Shoestring Manual Testing

Rex Black: President and Principal Consultant
RBCS, Inc., San Antonio, TX

Key Words: Testing, manual testing, test management, staffing, test engineers, test technicians, schedule, inexpensive, budget, automation, hiring, training.

Audience: Test managers, test leads, project managers, test engineers, recruiters, trainers.

Introduction

In the Test Manager's perfect world, her team would be staffed entirely by expert software engineering professionals. These test engineers could converse intelligently one minute with development engineers about code coverage and memory leaks; the next minute talk with technical support agents about the customer's experience of quality; and one minute later speak to marketeers about trade-offs between features, schedule, budget, and quality in the upcoming release. Such engineers would spend their time programming unit and integration test stubs and scaffolds for structural testing, scripting automated behavioral tests at the GUI level, and creating load generators and performance probes. Manual testing would be used sparingly to fill the gaps in such tests, for example by producing interesting error conditions and checking the system's response. The Test Manager would have ample staffing budget to hire such peer-level test engineers, and would have early involvement in the development effort. He could focus on verifying quality risk coverage using techniques like failure mode and effect analysis, analyzing and reporting defect management metrics, and setting up career development plans for his engineers.

This paper is not for the lucky few Test Managers who work in that world, but rather for the rest of us. On most of my test projects, I have to solve the following problems:

- The impracticality of extensive test automation.
- An insufficient number of test engineers.
- The collapse of an already-tight schedule as deliverables slip.

In this paper, I'll introduce you to the techniques I use to perform manual testing on a tight budget and schedule with a few test engineers and a larger group of test technicians.

The Need for and Challenges of Economical Manual Testing

Let's start by agreeing about what manual testing is and isn't. I define manual testing as developing and executing tests that rely primarily on direct human interaction throughout the entire test case, especially in terms of evaluating correctness and ascertaining test status (pass, fail, warn, etc.) By contrast, I define automated testing as developing and executing tests that can run unattended, including comparing actual to expected behaviors and logging status. Manual testing can include the use of tools to create certain test

conditions, such as background loads, error conditions, and the like, or to capture performance statistics or internal system states. If testers fire off test scripts and walk away, returning hours later to check results logs, they're performing automated testing. If testers enter data at input devices and actively observe output devices, that's manual testing. (See figure 1.)

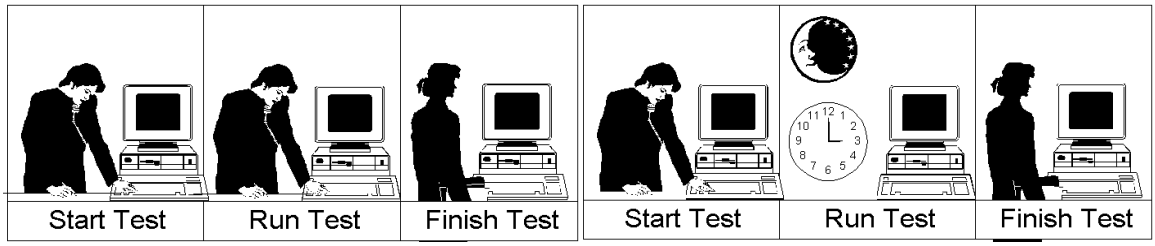

Figure 1: Manual Testing and Automated Testing

Complete test automation seems the ideal solution to tight schedules and budgets. A couple test engineers can start tests, let them run, stop the tests, and analyze results. Their time would be spent on critical tasks like reproducing, isolating, and reporting bugs, with a minimal effort expended actually to run the tests.

Alas, this oasis of easy living in the test desert is a mirage. On a typical test project, many obstacles can bar the path to total test automation. First, the system under test is subject to frequent change during the usual development project. Automated tests may require extensive and frequent updates, due to some tweak to the user interface, the core logic, an API, or the like. Second, developing automated tests takes a serious investment of time and money; on a tight schedule with limited budgets, a test team can only automate a few tests, which can create dangerous coverage gaps. Third, you may find a lack of suitable or affordable test automation tools for your specific configuration on the market, especially if you're working with a cutting-edge product that uses new technologies. No matter what your configuration, you will almost certainly have to develop your own test scaffolds and harnesses for any custom APIs you plan to test automatically. Fourth, you may not have appropriately skilled staff on your test team to pull off an automation project. Even with training, these skills take time and experience to acquire, so, unless you can retain a test automation consultant, you probably don't have the right talent available unless you're already doing a lot of test automation, perhaps on other projects. And finally, you may find that the critical quality risks for your product—the tests you most need to run—are not amenable to automated testing. For example, configuration, compatibility, and user interface testing tend to require a lot of manual interaction.

For these and other reasons, you'll find yourself managing a significant amount of manual testing. But that's no cakewalk either. How do you staff the manual test team? What do test technicians, new to the field of software testing, need to know? Can you accelerate your schedule and maximize the use of scarce equipment with manual testing? What is the career path for your manual test technicians to become test engineers? What pitfalls await the test manager in when running manual tests? Finally, can you sell management on your plan? The remainder of this paper will address these issues.

Sizing the Manual Test Team

The first step in staffing a manual testing team is figuring out how many people you need. Usually you will want to run a fixed set of tests in a fixed period of time. Understanding the requisite staffing levels can be quite easy or extremely difficult, depending on whether you know three salient facts about each of your test cases:

1. How many person-hours of effort are involved.
2. How long (wall-clock hours) the test case takes to run.
3. What dependencies exist between test cases.

If you have your tests well-defined, with this information available for each, you can put together a Gantt chart and resource plan quickly using software like Microsoft Project and other project planning programs.

As you prepare your plan, make sure you estimate the amount of overhead and lost time that will occur in the course of running your tests, due to factors like:

- Filing bug reports;
- Reporting test case status;
- Communication, e-mail, and management leadership;
- Breaks;
- Waiting for blocking issues to be resolved or actively helping to debug them.

For the first four categories, a good rule of thumb is that each technician has (potentially) six testing hours in an eight- to ten-hour day. The last category, down-time, will vary significantly, especially towards the beginning of your test effort, depending on the quality of the system under test when testing begins. I have seen situations as bad as 75% down-time or worse (sic) when very little meaningful unit testing occurred before we received a build for testing, with 25% being a good estimate when software entered the test process reasonably well qualified.

Figure 2 shows an example Gantt chart for one full pass of all tests for a manual test effort with five test technicians and two test engineers. I have omitted the test development effort, but that is likely to be extensive when doing lots of manual tests. Also note that the test team accepts one build only per pass, at the beginning of the pass. Figure 3 shows the test team organizational chart. The test engineers provide technical leadership to the test technicians, making sure they execute the tests and report their results properly.

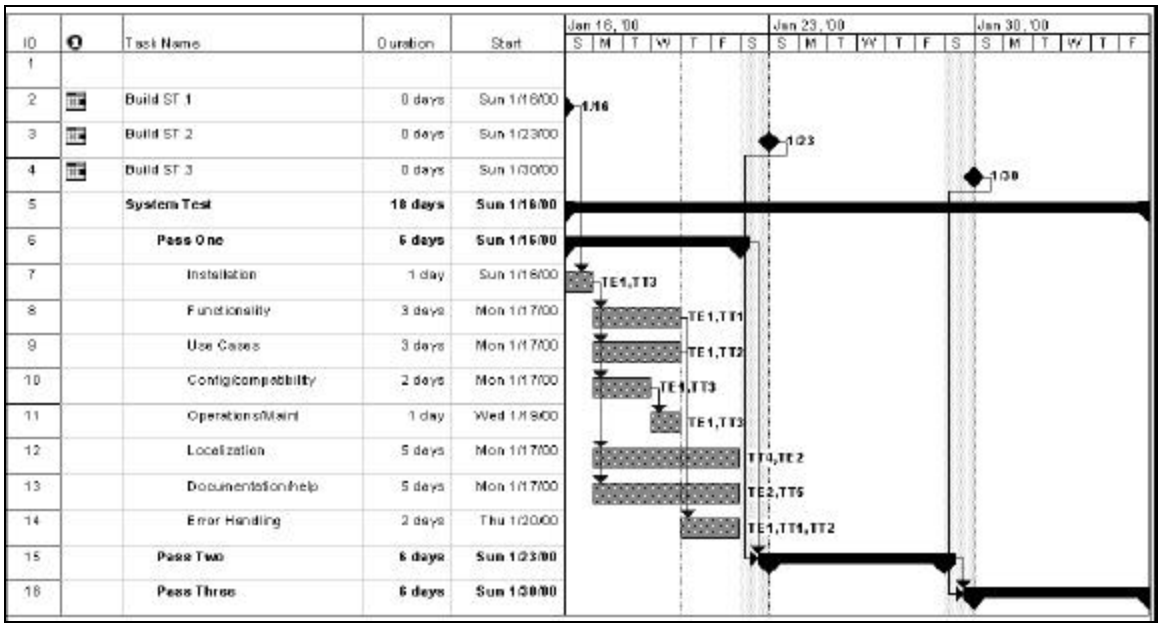

Figure 2: A Manual Test Gantt Chart

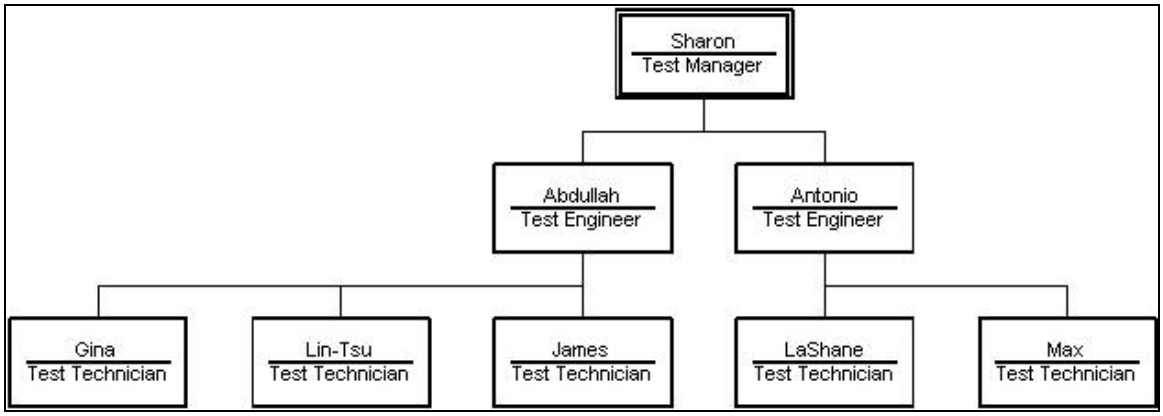

Figure 3: A Manual Test Team Organizational Chart

Hiring Good Test Technicians

Once you have an idea of how many people you need, it's time to start looking for candidates. The only limit on where to find good technicians is our imaginations, but I have found the following four sources particularly fruitful, if I'm careful about certain unique issues for each.

1. College and technical school students. Some of my favorite test technicians have been current or recently graduated two-year, four-year, and technical school degree students. I have had especially good luck with engineering and CS majors. However, you do need to make sure that enrolled students can commit enough time, especially when finals rolls around.
2. Technical and customer support staff. These folks are great because they understand problems from a customer's perspective, and have first-hand experience with what a

test escape means. They have to adapt, though, to just identifying problems rather than trying to solve them.

3. Moonlighters. These people can bring valuable experience from their day jobs, especially if they use the kind of software you're testing or work in some related field. Remember, though, that staying awake and focused after a full day's work is beyond the capacity of some people, and that your work will not be the first priority should conflicts arise.
4. Data entry personnel. Clerks, word processors, and call center agents are usually experienced computer users, and adeptly perform tests that require accurate entry of many screens of data. You must look carefully for the curiosity to dig into problems, for those who lack it do a poor job at reporting bugs.

By placing employment advertisements in your local paper, by posting job notices at colleges, universities, and technical schools, and by calling temporary staffing agencies, I am usually able to locate plenty of qualified candidates. I have found that I can staff test technicians easily and reliably at rates of up to one new technician per test engineer per week.

Training the Test Technicians

Bringing on inexperienced testers as key contributors to your team means that you need to provide appropriate training. Some of this training will be unique to the system under test and the environment in which you work. Some of it is intertwined with the inevitable logistics of hiring, such as obtaining a network login, getting a badge and dealing with security, using e-mail, and navigating the telephone system. Plan on having some approach—either documented processes or assigned mentors—to making sure the new staff learn "how things work around here."

Regardless of your specific environment, though, all neophyte test technicians will need to learn two processes. The first of these is how to begin, execute, and finish a manual test case. This task has both internal and external facets. In terms of internal considerations, you must teach the test technicians what level of ambiguity exists in the documented test cases, what amount of ad hoc exploration (if any) you expect when running test cases, the way to determine how long a test case should take to run, what states (pass, fail, warn, block, etc.) the tech can assign to a test case, and so forth. External considerations include how to prevent test execution overlaps and gaps, how to update test case status in a timely and consistent fashion, who assigns test cases, and how to handle dependencies between test cases. The specific answers to these questions will depend on your test case repository, your status reporting obligations, and your intellectual approach to testing. You can—and probably should—document a process for manual test case execution; one or two pages should suffice.

The second universal process is bug reporting. A good bug report communicates an issue effectively to the test team, the developers, other technical contributors, peer-level managers, and senior management, while poorly-written bug reports, no matter how serious the underlying problem, often fail to get attention, leaving the failure languishing. To ensure outstanding bug reports, I use a ten-step process:

1. Structure. A tester who uses a deliberate, careful approach to testing, and takes careful notes, tends to have a good idea of what's going on with the system under test. When failures occur, he knows when the first signs of failure manifested themselves.
2. Reproduce. The tester should check reproducibility of a failure before writing a bug report. If the problem doesn't recur, she should still write the bug report, but she must note the sporadic nature of the behavior. A good rule of thumb is three attempts to recreate the failure before writing the report.
3. Isolate. After reproducing the failure, the tester should then proceed to isolate the bug. This refers to changing certain variables, such as system configuration, that may alter the symptom of the failure. This information gives developers a head start on debugging.
4. Generalize. After the tester has an isolated and reproducible failure, he should try to generalize the problem. Does the same failure occur in other modules or locations? Can he find more severe consequences of the same fault?
5. Compare. If a tester has previously verified the underlying test condition in the test case that found the bug, the tester should check these prior results to see if the condition passed in earlier runs. If so, then the bug is likely a case of regression, where a once-working feature now fails. Note that test conditions often occur in more than one test case, so this step can involves more work than just checking past runs of the same test case. In addition, the test should check the behavior on a reference platform, if any exists.
6. Summarize. The first line of the bug report, the failure summary, is the most critical. The tester should spend some time thinking through how the failure observed will affect the customer. This not only allows the tester to write a bug report that hooks the reader and communicates clearly to management, but also helps with setting bug priority.
7. Condense. With a first draft of the bug report written, the tester should reread it, focusing on eliminating extraneous steps or words. You don't want cryptic commentary, but the report should not wear out its welcome by droning on endlessly about irrelevant details or steps that need not be performed to repeat the failure.
8. Disambiguate. In addition to eliminating wordiness, the tester should go through the report to make sure it is not subject to misinterpretation. Some words or phrases are vague, misleading, or subjective, and should be avoided. Clear, indisputable statements of fact are the goal.
9. Neutralize. Being the bearer of bad news presents the tester with the challenge of delicate presentation. Bug reports should be fair-minded in their wording. Attacking individual developers, criticizing the underlying error, attempting humor, or using sarcasm can create ill will with developers and divert attention from the bigger goal, increasing the quality of the product. The wise tester confines her bug reports to statements of fact.
10. Review. Once the tester feels he has written the best bug report he can, he should submit it to one or more test peers for a review. The reviewing peers should make suggestions, ask clarifying questions, and even, if appropriate, challenge the tester's

assertion that the behavior is buggy. The test team should only submit excellent bug reports, given the time constraints appropriate to the priority of the bug.

On my test teams, we write bug reports that are accurate, concise, thoroughly-edited, well-conceived, high-quality technical documents. Such documents are effective tools to communicate our findings to all interested parties.

In many cases, this training is on-the-job, because test technicians often join once testing has started in an effort to catch up or accelerate the test schedule. In such circumstances, you should avoid bringing on more people than your test engineers and you can handle in the midst of testing crunch mode. Even though test technicians ramp quickly, there is a negative impact on team productivity for the first couple weeks as new hires learn their duties and test team processes from more experienced test team members.

Using Multiple Shifts

By "multiple shifts" I mean employing night (from about 4 PM to 1 AM) and/or graveyard (from about 12 AM to 9 AM) testers, in addition to the day shift (from about 8 AM to about 5 PM) testers. In some cases, you may want to use multiple shifts. Some reasons to do so are:

- Schedule. To accelerate the schedule, you need to get more work done in one day. Often, you can only have so many people in the test lab at one time, so adding extra shifts offers you the only option.
- Resources. Even if you're not constrained by lab space, you may have a shortage of hardware platforms. For example, on a project where hand-made engineering prototypes are used, the allocation of such devices for testing will be very limited. Utilizing these scarce resources sixteen or even twenty-four hours per day may be necessary.
- Staff. In some cases, test technicians will want to work an off-hours shift. For example, moonlighters and students may find this most convenient. Allowing a night shift will provide you with good technicians you otherwise couldn't hire.
- Politics. As pointless as it may seem, many of us have probably worked on projects where a macho geek chic prevailed. These are the projects where managers boast about whose staff works the longest hours. Multiple shifts allow you to keep your individual test technician's hours to a civilized eight to ten per day, while still claiming sixteen- or twenty-four hour-per-day test coverage.

Regardless of *why* you implement a multi-shift approach, the benefits accrue in all areas.

You need to keep a few facts in mind when implementing a multi-shift approach. Tell people, during the interview, which shift they're interviewing for. Plan on paying an "undesirable hours" bonus like two extra billable hours per shift worked or the like. Finally, make sure to allow an hour or two of shift hand-off time for communications, question, and even a formal shift meeting. Remember, too, that you will have to work extra hard to build a sense of team togetherness in a group that spends only an hour or so together each day.

Technical and Managerial Pitfalls of Manual Testing

Technically, just as not all tests are suitable for automation, not all tests are a good fit for manual testing. For example, statistically valid performance tests require precise levels of load. Stability and reliability testing requires subjecting a system continuously to representative usage conditions. Table 1 shows a list of the kinds of tests that are and aren't suitable for manual testing. Beware of using manual testing inappropriately, because you can create an impression that your testing is covering more quality risks than it actually does. This can lead to test escapes—situations where you could have reasonably been expected to have found bugs that you missed—which can damage your credibility as a test manager.

Suitable for Manual Testing	Not Suitable for Manual Testing
• Functional • Use cases (user scenarios) • Error handling and recovery • Localization • User interface • Configurations and compatibility • Operations and maintenance • Date and time handling • Installation, conversion, and setup testing • Documentation and help screens	• Monkey (or random) • Load, volume, and capacity • Reliability and stability (MTBF) • Code coverage • Performance • Standards compliance

Table 1: Test Suitability for Manual Testing

Managerial pitfalls are, as usual, more subtle and often harder to deal with than the technical challenges. For the most part, manual testing as described in this paper doesn't present any challenges beyond what every manager faces, but I can think of three issues that are unique. First and foremost, make sure that you monitor the performance and behavior of inexperienced technicians. When you hire a test engineer with five, ten, or twenty years of work experience, you can assume that, along with being fit for the job, he knows how to comport himself professionally and what level of output will be expected. If you're giving someone their first job, you're taking a bigger risk. Some of these junior people won't work out, not always for reasons strictly related to ability to do the work. Steel yourself for the reality that eventually, you'll have to reassign or even fire a technician.

Second, if you choose to use off-hours shifts, do pay attention to what goes on. Monitor the level of output carefully; some people over-estimate their ability to be productive at night. Moonlighters may be burnt out from a long day at their "real job." Also, remember personnel dynamics: Out of sight isn't out of mind for long when conflicts arise.

Third, some types of testing require extensive domain knowledge. Your junior test technicians—unless they happen to have subject matter expertise from a previous career—probably won't work for these situations. For example, statistical analysis packages or electronic circuit simulation tools don't lend themselves to novice use.

However, even with these complex products, you can have inexperienced test technicians run some tests; e.g., anyone can test the installation of a Windows package, no matter how complex the program being installed.

Fourth, some of your test technicians will join with aspirations of moving on in the company. If you fail to provide a career path for them, the most talented will leave to find other jobs. This is a waste, because these people have valuable domain knowledge and testing experience by the time they leave. Groom promising technicians for advancement as test engineers, or, if they have other long-term goals, like marketing or development, help them find a place in those organizations. Please note, however, that I'm not advocating turning your test team into a boot camp from whence people graduate to more desirable positions in the company. You can manage the number of test technicians that leave your team by hiring people at all levels who want to remain in the testing and quality fields.

A Quick Case Study

Let's look at a practical example. This real-life manual test effort, shown in figure 4, occurred on an Internet appliance test project. We hired six test technicians, some of whom stayed on after the project was over to become test engineers. The technicians worked in two shifts, one day shift from nine to six, one evening shift from three to midnight. Two test engineers provided technical guidance to these technicians, assigning them test cases, monitoring their test work, and summarizing their status. The two test toolsmiths worked for the server test engineer, creating and running load scripts that produced stress, capacity, and volume conditions similar to actual field scenarios while the test technicians were running tests. I, as the test manager, oversaw the entire operation, managing the team, our interaction with peer-level managers, and our reports and escalation to senior managers. (What I call "inward," "outward," and "upward" management, respectively.)

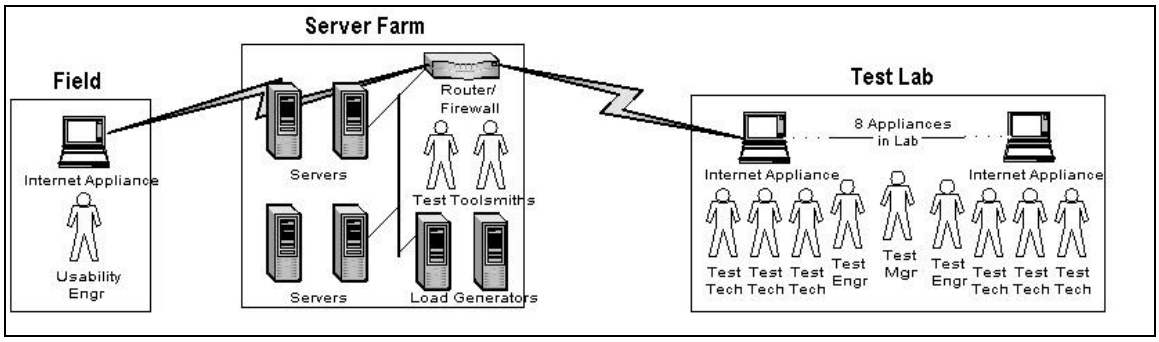

Figure 4: An Internet Appliance Manual Test Team

System Test was organized into a series of five two-week passes. During each pass, we accepted a new software build from the release management engineer every Sunday evening. This exposed us to some regression risk during the pass, as tests that passed during the first week might have failed had we run them the second week. However, we felt accepting a weekly build was a good compromise between focusing on preventing regression and making sure we verified bug fixes (confirmation testing) quickly.

With this test organization, we ran about 250 to 300 manual test cases per two-week System Test pass. The total effort to run these test cases was about 300 to 400 person-hours. Adding the 35 or so hours of confirmation testing the first day of each pass, this results in about 350 to 475 person-hours of effort per test pass; this is consistent with the six-day-weeks worked by most of the test team during System Test.

We also ran subsets of the test suites for special purposes. We had two- and three-day long Smoke Tests designed to ensure the sanity of a build before beginning a full pass of all test suites. In addition, we ran a one-week Validation Test against the final build before release, to check high-risk regression items. All the test cases in these pruned test groups were selected based on their previous ability to find high-risk bugs.

Selling Management on Manual Testing

Your management will likely become big boosters of manual test efforts, once you explain to the benefits of manual testing. You should put together a business case for the approach, including the following factors:

- Flexibility and responsiveness. Because of the junior stature and limited duties of test technicians, you can hire and train them more rapidly than test engineers, and some will accept short-term positions.
- Cost and coverage. Test technicians are much cheaper than test engineers, sometimes as much as four-to-one. This allows the test budget to provide for broader test coverage than if you used all test engineers.
- Schedule. By using lots of technicians, the time required to run a pass of all the test suites is reduced. As discussed above, system utilization may increase. These factors can accelerate the release schedule.
- Staff retention. Test engineers typically do not want to spend all their time in repetitive manual testing. Allowing them to write the tests and provide the technical leadership keeps the manual test effort challenging for them.
- Doing good while doing well. By hiring college students, technical school graduates, and other people new to software development, your company is helping jump-start careers. As much as the company benefits, the technicians do also.

Having done your homework on the size and staffing of the test team already, you can come to your manager will a comprehensive manual testing plan. For example, you can tell her something like, “I plan to hire seven test technicians. This is based on a complete pass in one week, with about 200 testing hours plan for a complete pass of the System Test suite. I have ten good resumes already based on some ads at local campuses and tech schools. I can handle a 15% productivity hit during the ramp-up period, so I’ll stagger the hiring over three weeks.” Having thought through the issues usually helps facilitate management acceptance.

Some managers may ask you, “What about automation?” For these skeptics, you should revisit the reasons why you needed to take a manual approach in the first place. Make a business case for using a manual approach. This is especially easy when you must create your own automation tools or when the system is changing so rapidly that test suite

maintenance will eat up any advantages offered by automation. Finally, you can remind the automation-obsessed that your manual test cases will translate well into automated ones when the time is ripe.

Conclusions

While the ideal test world would involve a small team of test engineers writing and running automated tests, this is often not possible and even more often not practical. Fortunately, the Test Manager can use test technicians to make manual testing effective and economical, and prevent staff burn-out. Good manual testing starts with careful planning of the test process and test cases, and just as automated testing has limitations, you can't use manual testing for everything. You must also build the team carefully, selecting the right test technicians. Once the technicians are hired, you and your test engineers will need to teach them how to run test cases and how to report bugs, along with the other typical "way things work around here" lessons that all new hires need. On the dozens of projects on which I have used manual testing extensively, I have found that it solves problem for me *and* for management. Applying the ideas presented in this paper will help you perform effective manual testing on a shoestring budget.

Presenter Biography

Rex Black has spent almost two decades in the computer industry, with most of that time in testing and quality assurance. He is the President and Principal Consultant of Rex Black Consulting Services, Inc., an international software and hardware testing and quality assurance consultancy. His clients include Dell, SunSoft, Netpliance, Clarion, General Electric, First USA, and others. For these clients, he has managed a number of test projects with both automated and manual suites. His work with these clients has taken him to Taiwan, Hong Kong, Japan, the U.K., Canada, Germany, Holland, France, Spain, Switzerland, and Italy, along with locations throughout the United States.

Rex is the author of *Managing the Testing Process*, published by Microsoft Press. This book is a comprehensive introduction to the complex and challenging job of managing software and hardware development test projects. He frequently presents original papers at conferences such as Practical Software Quality Techniques, Quality Week, Software Testing and Analysis Review, and the World Congress on Software Quality. Rex is a trainer for the International Institute of Software Testing, specializing in test and quality assurance management.

Before becoming a consultant in 1994, Rex worked as Quality Assurance Manager at Locus Computing and IQ Software, doing operating system testing for IBM and database and data warehouse query and analysis tool testing. He also spent three years as a Project Manager at NTS/XXCAL, an independent computer testing lab. Prior to that, he worked as a programmer and system administrator.

Rex holds a B.Sc. in Computer Science and Engineering from UCLA. He belongs to the Association for Computer Machinery and the American Society for Quality.

You can reach him at:

Mail: 31520 Beck Road
Bulverde, TX, 78163-3911

Phone: + 1 (830) 438-4830

E-mail: rex_black@rexblackconsulting.com

Web: www.rexblackconsulting.com