13

Assess Quality: Execute Test Cases

Finally the time has come to do some testing. During the test execution process, the testers will use whatever test system we have put together to assess the quality of the test releases as we receive them. This is both a potentially exciting and a potentially scary point in the project because the quality of the system is unknown until the testers start pounding on it. Furthermore, the test team generally has a unique charter here, that of looking for ways to make the system under test get as far *out of control* and *away from the expected* as possible.

In addition, while all managers are expected to manage unexpected events within their areas of responsibility, the test manager is the only manager in the software development organization whose area of responsibility is *searching for* the unexpected. This search takes the form of running a set of tests. How does a test team take these tests, run them against an installed system, and capture information about the system under test, the test cases, and the test execution process itself?

A Test Execution Process

Let's start with a generic test execution process, shown in Process 9. This process has a number of intersections with other processes. Step 3.D, reporting quality problems in the system under test, follows the process outlined in Chapter 14. Step 3.E, fixing test system problems, follows the process outlined in Chapters 10 and 11. Finally, step 7, reporting test cycle findings and status, follows the process discussed in Chapter 15.[1]

1. I thank Tim Koomen for reminding me of some of the test configuration management elements in the process.

Step #	Step	Done?
1.	Based on risk prioritization, project constraints, and any other pertinent considerations, select the test suites (from the test set) that should be run in this test cycle.	☐
2.	Assign the test cases in each test suite to testers for execution.	☐
3.	Execute the test cases, report bugs, and capture information about the tests continuously, taking into account previous test results for each subsequent test.	☐
3.A	Put the system under test and the test system into appropriate initial states. If these initial states are useful across multiple tests or multiple iterations of this test, save the initial states for subsequent reuse.	☐
3.B	Through data inputs and other stimuli, provoke the system under test into a desired test condition.	☐
3.C	Observe and evaluate the resulting outputs, behaviors, and states. Research any deviations from expected results.	☐
3.D	If appropriate, report problems in the system under test.	☐
3.E	If appropriate, report and/or resolve problems in the test system.	☐
3.F	Capture and report information about the test just executed.	☐
4.	Resolve blocking issues as they arise.	☐
5.	Report status, adjust assignments, and reconsider plans and priorities daily.	☐
6.	If appropriate, eliminate unrealizable or redundant tests in reverse-priority order (drop the lowest-priority tests first, the highest-priority tests last).	☐
7.	Periodically report test cycle findings and status.	☐
8.	Check any status documents, initial states, updated testware or other test system elements, or other useful permanent records produced into the project library or configuration management system. Place the item(s) under change control.	☐

Test Set

All the test suites that the test team plans to run during the current test phase (or level).

TEST TRACKING WORKSHEETS

In this chapter, I'll illustrate planning and tracking test execution through the use of what I call a test tracking spreadsheet.[2] The essential portions of such a spreadsheet are the test case summary worksheet and the test suite summary worksheet.

Figure 13-1 shows a test case summary worksheet as it would appear at the beginning of pass 5, once Jamal Brown, the test manager, has finished selecting the test suites and assigning each test case to a tester. The Assigned To column indicates the responsible tester. The Test ID column puts a unique four-digit number on each test. The Test Suite/Case column shows a short name for each test suite and, within each test suite, a short name for each test case. The Plan Date column shows the date by which Jamal wants the tester to have run the test case. The Plan Effort column indicates the number of person-hours Jamal estimates the tester will spend running each test case.

As the testers run each test case, they capture and report information about each test. One of the most essential bits of information is the state of the test.

- *Queue*—The test case is ready to run, assigned to a tester for execution in this test pass.
- *In Progress*—The test is currently running.

	A	B	C	E	F	G	H	I	J	K	M	N	O	P
1			Sumatra Test Case Summary											
2			Integration and System Test Pass 5 (12/16-12/27)											
3														
4	Assigned	Test			Test	Bug	Bug	Run	Plan	Act	Plan	Actual	Test	
5	To	ID	Test Suite/Case	Status	Environment	ID	RPN	By	Date	Date	Effort	Effort	Duration	Comment
6														
7		*1.000*	*Functionality*											
8	DLK	1.001	File						12/16		2.0			
9	JW	1.002	Edit						12/16		0.5			
10	BEZ	1.003	Font						12/16		0.5			
11	DLK	1.004	Tables						12/17		3.0			
12	BEZ	1.005	Printing						12/16		2.0			
13	LJ	1.006	Managed Files						12/16		2.5			
14	LJ	1.007	Non-Managed Files						12/17		4.0			
15	BEZ	1.008	DMS Check-In New						12/17		2.5			
16	BEZ	1.009	DMS Check-In Existing						12/17		3.0			
17	BEZ	1.010	DMS Check-Out						12/18		3.5			
18	LTW	1.011	DMS Load/Migrate Online						12/16		3.5			
19	HS	1.012	DMS Load/Migrate NOL						12/16		3.0			
20	HS	1.013	DMS Load/Migrate Offline						12/17		3.5			
21	HS	1.014	DMS File Information						12/17		2.5			
22			**Suite Summary**						**12/18**	**1/0**	**36.0**	**0.0**	**0.0**	

FIGURE 13-1 INITIAL TEST CASE SUMMARY FOR PASS 5

2. See my previous book, *Managing the Testing Process, Second Edition*, Chapter 5, for a detailed explanation of this tool.

- *Block*—Something prevented the tester from executing the test.
- *Skip*—For some reason, the tester decided not to execute the test.
- *Pass*—The tester ran the entire test and observed only expected results, states, and behaviors.
- *Fail*—The tester ran the entire test and observed one or more unexpected results, states, or behaviors seriously compromise the quality of the system with respect to the objective of the test.
- *Warn*—The tester ran the entire test and observed one or more unexpected results, states, or behaviors, but the underlying quality of the system with respect to the objective of the test was not seriously compromised.
- *Closed*—After being marked in a Fail or Warn status in the first cycle of a test pass, the next test release included a fix to the bug that resolved the problems with this test case.

In the course of running all the planned test cases in a test pass, testers move through the test lifecycle shown in Figure 13-2.[3]

In addition to the test case status, testers gather other important information. The Test Environment column (see Figure 13-1) is where they indicate what hardware,

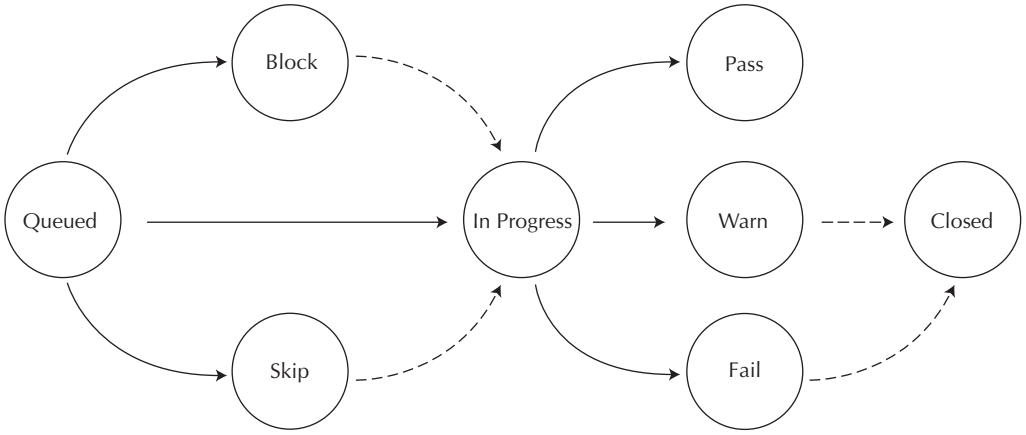

FIGURE 13-2 TEST CASE STATES AND LIFECYCLE

3. Some people use simpler sets of test states. Johanna Rothman, for example, mentioned to me that she does not use Block, Warn, or Closed, but instead uses Fail, In Progress, and Pass, respectively.

software, and other items they used to run the test. Bug ID and Bug RPN capture information about any bugs found (see Chapter 14). In the Run By column each tester enters his or her initials. Act Date is when the tester ran the test case, while Actual Effort is how many person-hours he or she expended, and Test Duration is the clock time required to run the test. (Test duration and effort can vary for automated tests or for manual tests that involve two testers.)

To get a better sense of what's going on across all the test suites—and to gather useful information for metrics (see Chapter 15)—I analyze the test status and test effort information in a test suite summary worksheet, as shown in Figure 13-3. This worksheet shows a summary for each test suite and for the entire test set. I classify as *fulfilled* those tests either run or skipped. (I count tests in a Warn or Closed state in the Pass column for simplicity.) I classify as *unfulfilled* those tests blocked or not yet completed. This allows me to analyze where I stand in terms of getting through all the tests.

The earned value section at the right analyzes whether we have gotten through the right number of tests in each test suite, given the actual effort expended. For example, if we have run seven out of 14 test cases, we have executed 50% of the tests (which would appear in the % Exec column). Suppose we have expended 40 person-hours out of a total of 50 planned person-hours? In that case, we've burned through 80% of the planned effort (which would appear in the % Effort column). This indicates that we are

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Sumatra Test Suite Summary														
2	Integration and System Test Pass 5 (12/16-12/27)														
3															
4		Total	Planned Tests Fulfilled				Weighted	Planned Tests Unfulfilled				Earned Value			
5	Suite	Cases	Count	Skip	Pass	Fail	Failure	Count	Queued	IP	Block	Pln Hrs	Act Hrs	% Effort	% Exec
6															
7	Functionality	14	0	0	0	0	0.00	14	14	0	0	36.00	0.00	0%	0%
8	Performance, Load, Capacity, and Volume	5	0	0	0	0	0.00	5	5	0	0	7.00	0.00	0%	0%
9	Reliability/Stability	2	2	2	0	0	0.00	0	0	0	0	0.00	0.00	0%	0%
10	Error Handling and Recovery	3	0	0	0	0	0.00	3	3	0	0	12.50	0.00	0%	0%
11	Installation, Maintenance, and Operations	4	0	0	0	0	0.00	4	4	0	0	72.00	0.00	0%	0%
12	Localization	8	0	0	0	0	0.00	8	8	0	0	128.00	0.00	0%	0%
13	Security and Privacy	4	0	0	0	0	0.00	4	4	0	0	17.00	0.00	0%	0%
14	Documentation	3	0	0	0	0	0.00	3	3	0	0	28.00	0.00	0%	0%
15	Integration	4	0	0	0	0	0.00	4	4	0	0	8.00	0.00	0%	0%
16	Usability Study	2	0	0	0	0	0.00	2	2	0	0	16.00	0.00	0%	0%
17	Exploratory	6	0	0	0	0	0.00	6	6	0	0	12.00	0.00	0%	0%
18															
19	Total	55	2	2	0	0	0.00	53	53	0	0	336.50	0.00	0%	0%
20	By Pct		4%	4%	0%	0%	N/A	96%	96%	0%	0%				

FIGURE 13-3 INITIAL TEST SUITE SUMMARY FOR PASS 5

unlikely to complete the remaining seven tests in the time scheduled (assuming that all the test cases are roughly the same size and disregarding priority issues). If the pattern holds across all the test suites, then we will run out of test execution time before we complete all the tests planned for the test pass. So, earned value gives the test lead or manager the opportunity to adjust the plan for the test pass before that happens.

THE ATTACK ON THE BIG BUILD

For their first activity every week, the Sumatra test team ran confirmation tests against the bug fixes in the new test release, as described in the Sumatra test plan. In the case of the Big Build, they had a lot of bugs to test. While the test team focused on confirmation testing, Jamal worked out the details of testing for the coming week.

Since this was increment 3, that meant that the first run of the localization and the usability test suites—both run by external test labs—would start this week. Jamal had asked Lin-Tsu to coordinate those activities with the test labs and had made a point to remind her of that last week. She had touched base with the labs on Tuesday and had scheduled delivery of the test release for Monday afternoon with both facilities. That would give her time to make sure the test release was ready before they started testing it.

The Big Build was going to be a big job, Jamal realized as he looked at the test case tracking worksheet. The only test suite he planned to skip during this test pass was the Reliability/Stability suite, but that meant that Emma Moorhouse, one of the automated test engineers, would run the Performance, Load, Capacity, and Volume suite, which was a larger effort.

The fact that the Christmas holiday fell right in the middle of the test pass also created a certain challenge. However, Jamal had leveraged his multicultural test team to avoid a completely dead week. He had already arranged for Lin-Tsu Wu, his manual test engineer, and John Woo, one of his test technicians, to take off the Chinese Lunar New Year holiday early next year instead of Christmas. Dinesh Kumar, one of the automated test engineers, and Hemamalini Shashidar, another test technician, would get an extra week on top of two weeks of vacation in April following the release to visit family in India and Bangladesh. Binyamin Zuckerman had visited his family in New York during Hanukkah. This way, each tester got to enjoy their own religious and cultural holidays, and he'd spread the effects of these holidays across many test passes. Now he'd reap the benefit, having a team of five over the Christmas week, down by just two people from the usual seven.

For the moment, though, Jamal focused on mapping out the test cases he needed the team to run during the coming test cycle this week. The first order of business, from a risk perspective, would be running tests that had previously been blocked, especially the functionality tests. Some of the changes dropping in meant that privacy, security, and error handling tests were also important.

Step #	Step	Done?
1.	Based on risk prioritization, project constraints, and any other pertinent considerations, select the test suites (from the test set) that should be run in this test cycle.	☑

Based on this prioritization, Jamal went through each test suite and assigned a planned completion date and a tester to each test case. Based on previous test passes, he also revised the planned hours for each test case. A partial snapshot of the test case tracking worksheet he came up with as part of this plan is shown in Figure 13-1.

Step #	Step	Done?
2.	Assign the test cases in each test suite to testers for execution.	☑

After completing the test assignments, Jamal printed seven copies of the test tracking worksheet for pass 5. Feeling that a group synchronization session was in order—and knowing that nothing makes for a well-received meeting like a good meal—he then sent an e-mail to the test team.

```
All—

The Big Build is here. The time has come for the most important
and challenging test pass of the project. To make sure we're all
in sync, please meet me in the test lab for a test pass kick-off
lunch at noon. I'll be handing out test case tracking worksheets
with the test case assignments, planned hours, and planned dates
for this test pass. To ensure promptness, I'll pick up lunch from
our favorite Indian restaurant, with the usual mild, vegetarian,
and kosher options available. Remember: those who live for the
nuclear hot chicken vindaloo and the aachar—like Emma and I do—
should show up on time or we WILL eat it all!

Regards,

Jamal
```

The meeting went swimmingly. There was a lot of work to do in a short period, but thanks to Jamal's up-front holiday planning, the workload was not overwhelming, even with Christmas. Perhaps more importantly, everyone on the team was eager to get their hands on the Big Build, a test release that was (almost) feature complete. As Luis Jacinto, one of the test technicians, put it, "Finally, a brimming mug of Sumatra, not one of those tiny little Italian espresso cups."

"Okay," Jamal said as the meeting—and the meal—wound down, "sally forth and find many bugs. I know they're hiding in the Big Build. I'll see you all at 4:00 PM for the shift handoff and afternoon status debriefing."

The testers went their separate ways to run their assigned tests. Luis Jacinto started his work with the managed file functionality test. The setup step of this test case involved starting a new instance of SpeedyWriter on a test client and then logging in to the document management system. The first step of the test involved creating a new file, entering and operating on some text in that file, and then checking the file into the document management system.

Step #	Step	Done?
3.A	Put the system under test and the test system into appropriate initial states.	☑
3.B	Through data inputs and other stimuli, provoke the system under test into a desired test condition.	☑

At this point, Luis discovered that the file-creation functionality was seriously broken, corrupting all new files when certain font operations were performed. He found that the only workaround was to check the file into the document management system immediately upon creation, before starting to edit it. In addition to running the planned test, Luis spent some time researching and writing the bug report. This took quite a bit longer than the scheduled two-and-a-half hours, but he wanted to spend the time necessary to isolate and document such a critical bug. Before moving on to the next test case, he used a pencil to write down the status information associated with running this test case in the test case tracking worksheet.

Step #	Step	Done?
3.C	Observe and evaluate the resulting outputs, behaviors, and states. Research any deviations from expected results.	☑
3.D	If appropriate, report problems in the quality of the system under test.	☑
3.F	Capture and report information about the test just executed.	☑

Luis then proceeded to the nonmanaged file test. However, before he could get too far into it, a system administrator inadvertently shut down Tobruk, the Linux Web server Luis was using. After some initial confusion, he got the server rebooted, but close to an hour evaporated as he tracked down this problem.

Step #	Step	Done?
4.	Resolve blocking issues as they arise.	☑

After the daily debriefing with the rest of the test team, Luis spent another couple of hours testing, then an hour reading and responding to various e-mails, before calling a ten-hour day to a close. He spent a few minutes talking to Binyamin Zuckerman, one of the evening shift test technicians, about the font-related bug he had found, since he had noticed from the test tracking worksheet that Binyamin was scheduled to run the font test that evening. They also talked a bit about Binyamin's recent trip to New York, especially his visit to the reconstruction site in Lower Manhattan. Finally, Luis headed out for a late dinner with some friends in downtown Austin.

Unfortunately, although Luis had closed the loop with Binyamin, he had not talked to Emma Moorhouse all day. That wouldn't have been a problem, except that when Luis sent his bug report out via e-mail for test-team-wide review, Emma was busy getting the automated Solaris performance tests running, so she didn't read it. And that wouldn't have been a problem either, except that the test scripts happened to depend on the correct behavior of exactly what Luis had carefully described as severely broken in his bug report.

So, when Emma left on Monday night, the Solaris performance tests were giving invalid results, but she didn't know it. (She hadn't implemented the features of the tool that would allow the scripts to send her a page if it was behaving incorrectly.) It wasn't until Tuesday morning that she discovered the oversight. Emma was frustrated and chagrined. First she vented to her coworkers and then she admitted to Jamal what had happened. Jamal responded with a worldly shrug and raised eyebrow, saying, "The joys of automated testing." Relieved but not surprised by Jamal's understanding outlook, Emma then reworked the test scripts in order to restart them for a run on Tuesday night.

Step #	Step	Done?
3.E	If appropriate, report and/or resolve problems in the quality of the test system.	☑

Based on the incident that occurred, Jamal realized that he needed to reinforce a process that had grown up informally over time. He sent out the following e-mail, marked urgent, to the test team.

All—

From the beginning of testing back in October, we've had an informal process of sending bug reports out via e-mail to the entire team for review and informational purposes. I think this process has been very helpful, both in fostering a high degree of professionalism in our bug reporting and in keeping us informed.

As a manager, part of my job is to institutionalize good ideas as part of our regular process. I've made an error in this regard, as I may have communicated the idea that reading those bug report e-mails is a low-priority activity, something that we can do once a day or even skip entirely if we're busy. However, keeping abreast of each other's test results—and considering how those results might affect our own testing work—is actually critical to effective and efficient testing.

To the extent that I led us to believe that reviewing each other's bug reports is a low-priority activity, I made a management mistake that I will now correct. Please consider reading each other's bug reports to be a central and recurring part of your daily work. Please make it a regular process, prior to starting the execution of any test case, to check your e-mail and read any e-mail from fellow testers related to bug reports. When sending out such e-mails, please make it a practice to have the first word of the subject line be "BUG:" followed by the bug ID and as much of the bug report summary as will fit in the e-mail subject line.

Some of you are already following the practices outlined in this e-mail. However, because of my failure to enunciate this policy at the start of test execution, some of you are probably not. In those cases, this new practice represents an increase in workload. If you believe that this change in the test execution process represents a significant increase in workload that will prevent you from completing previously assigned tasks by their agreed-upon dates, please feel free to come see me to discuss adjusting dates and/or assignments.

Finally, let me make clear that I am not sending out this e-mail to cast aspersions on anyone's work or to call into question anyone's judgment. I am extremely happy with the work everyone is doing on this challenging project. My objective is to correct a

```
management oversight on my part, and to take this opportunity to
improve the process. This incremental increase in workload will,
in the long run, save us all time, promoting our efficiency and
effectiveness.

Regards,

Jamal
```

Step #	Step	Done?
4.	Execute the test cases, report bugs, and capture information about the tests continuously, taking into account previous test results for each subsequent test.	☑

After the performance testing miscue was resolved, on Tuesday the testing continued at a quick, deliberate, yet careful pace. Luis continued his nonmanaged file tests in the morning, having a very productive testing session with that test case. He found six separate problems and filed six reports. Also, the simulated server crash the day before, while inadvertent, exposed a serious problem he otherwise would have missed. So, after discussing the matter with Lin-Tsu, he spent 30 minutes updating one of the error handling test cases to reflect the new condition. As it turned out, the nonmanaged file tests weren't totally completed until Wednesday morning.

On Wednesday afternoon at 4:00 PM, just as on every afternoon, Jamal sat down with the whole test team to debrief and transition from the day to the evening shift. One by one, he asked the testers to tell him which tests they'd run and how many person-hours each had taken, and to review for him all bugs found for each test. As each tester reported their status, Jamal updated the test case summary worksheet (see Figure 13-4). Once it was all over, Jamal updated the project metrics using the test suite summary worksheet (see Figure 13-5). This meeting usually took about 45 minutes, but with all the testing and excitement during the Big Build, they were lasting about an hour this week.

Following the team debriefing, Jamal looked at the updated test case summary and test suite summary. He realized that the security and privacy testing was now at risk for this test cycle. Binyamin was bogged down in a slew of important functionality bugs. John Woo, another evening test technician, had found a number of important problems in the documentation. Lin-Tsu was tied up with managing the localization and usability testing efforts. (These had become a larger than planned drain on her time because of regression bugs in the installation process.) This meant that everyone scheduled to run security and privacy test cases was tied up elsewhere.

	A	B	C	E	F	G	H	I	J	K	L	M	N	O	P
1			Sumatra Test Case Summary												
2			Integration and System Test Pass 5 (12/16-12/27)												
3															
4	Assigned	Test			Test	Bug	Bug	Run	Plan	Act	Test	Plan	Actual	Test	
5	To	ID	Test Suite/Case	Status	Environment	ID	RPN	By	Date	Date	Effort	Effort	Effort	Duration	Comment
6															
7		*1.000*	*Functionality*												
8	DLK	1.001	File	Fail	SW/H2/B1	1832	1	DLK	12/16	12/16	2.0	2.0	3.0	4.0	
9	JW	1.002	Edit	Warn	SW/H2/B1			DLK	12/16	12/16	0.5	0.5	1.0	3.0	
10	BEZ	1.003	Font	Fail	SS/H2/B7	1832	1	BEZ	12/16	12/16	0.5	0.5	2.0	4.0	
11	DLK	1.004	Tables	Pass	SS/H2/B5			DLK	12/17	12/17	3.0	3.0	3.0	5.0	
12	BEZ	1.005	Printing	Pass	SW/H1/B6			BEZ	12/16	12/16	2.0	2.0	2.0	4.0	
13	LJ	1.006	Managed Files	Fail	SW/H1/B8	1832	1	LJ	12/16	12/16	2.5	2.5	3.0	3.0	
14	LJ	1.007	Non-Managed Files	Fail	SS/H2/B8	1832	1	LJ	12/17	12/18	4.0	4.0	6.0	6.0	
15						1837	6								
16						1838	25								
17						1840	6								
18						1845	3								
19						1846	2								
20						1847	8								
21	BEZ	1.008	DMS Check-In New	Fail	SS/H1/B6	1836	1	BEZ	12/17	12/17	2.5	2.5	6.5	5.0	
22						1832	1								
23						1841	5								
24						1844	15								
25	BEZ	1.009	DMS Check-In Existing						12/17		3.0	3.0			
26	BEZ	1.010	DMS Check-Out						12/18		3.5	3.5			
27	LTW	1.011	DMS Load/Migrate Online	Fail	SS/H1/B2	1833	2	LTW	12/16	12/16	3.5	3.5	4.0	4.0	
28	HS	1.012	DMS Load/Migrate NOL	Fail	SS/H2/B3	1833	2	HS	12/16	12/16	3.0	3.0	3.0	3.0	
29	HS	1.013	DMS Load/Migrate Offline	Fail	SE/H2/B3	1833	2	HS	12/17	12/17	3.5	3.5	3.5	3.5	
30	HS	1.014	DMS File Information	Fail	SW/H1/B3	1833	2	HS	12/17	12/17	2.5	2.5	3.0	3.0	
31			**Suite Summary**						**12/18**	**12/18**		**36.0**	**40.0**	**47.5**	

FIGURE 13-4 THIRD-DAY TEST CASE SUMMARY FOR PASS 5

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1			Sumatra Test Suite Summary												
2			Integration and System Test Pass 5 (12/16-12/27)												
3															
4		Total	Planned Tests Fulfilled				Weighted	Planned Tests Unfulfilled				Earned Value			
5	Suite	Cases	Count	Skip	Pass	Fail	Failure	Count	Queued	IP	Block	Pln Hrs	Act Hrs	% Effort	% Exec
6															
7	**Functionality**	14	12	0	3	9	9.60	2	2	0	0	36.00	40.00	111%	86%
8	**Performance, Load, Capacity, and Volume**	5	2	0	1	1	2.83	3	3	0	0	7.00	5.50	79%	40%
9	**Reliability/Stability**	2	2	2	0	0	0.00	0	0	0	0	0.00	0.00	0%	0%
10	**Error Handling and Recovery**	3	0	0	0	0	0.00	3	3	0	0	12.50	0.00	0%	0%
11	**Installation, Maintenance, and Operations**	4	0	0	0	0	0.00	4	4	0	0	72.00	0.00	0%	0%
12	**Localization**	8	1	0	0	1	0.50	7	0	7	0	128.00	22.00	17%	13%
13	**Security and Privacy**	4	1	0	1	0	1.00	3	3	0	0	17.00	4.50	26%	25%
14	**Documentation**	3	0	0	0	0	0.00	3	3	0	0	28.00	0.00	0%	0%
15	**Integration**	4	2	0	2	0	0.00	2	2	0	0	8.00	3.00	38%	50%
16	**Usability Study**	2	0	0	0	0	0.00	2	0	2	0	16.00	0.00	0%	0%
17	**Exploratory**	6	0	0	0	0	0.00	6	6	0	0	12.00	0.00	0%	0%
18															
19	**Total**	55	20	2	7	11	13.93	35	26	9	0	336.50	75.00	22%	34%
20	**By Pct**		36%	4%	13%	20%	N/A	64%	47%	16%	0%				

FIGURE 13-5 THIRD-DAY TEST SUITE SUMMARY FOR PASS 5

That, Jamal realized, leaves me no choice but to call on the Reserve Tester: Jamal Brown. Try as he might, though, he knew he wouldn't get through 17 hours of tests in the next three days, not with the other fires burning in the test arena. He replanned the testing as shown in Figure 13-6.

Step #	Step	Done?
5.	Report status, adjust assignments, and reconsider plans and priorities daily.	☑
6.	If appropriate, eliminate unrealizable or redundant tests in reverse-priority order (drop the lowest-priority tests first, the highest-priority tests last).	☑

Thursday and Friday proceeded almost to plan, with only minor delays. Lin-Tsu and Emma, having a few tests to wind down on Saturday morning, agreed to meet Jamal in the test lab around 10:00 AM to put the cycle to bed. Dale, the test technician who ordinarily smoke tested the test releases, offered to come in as well to get the next cycle's release installed for a quick start on Monday morning.

Jamal consulted with his test engineers. Since the test technicians had been working long hours during the week, Lin-Tsu, Emma, Dinesh, and Jamal decided not to ask them to come in on Saturday to try to get ahead of the game. As Jamal said to his three test engineers over steaming cappuccinos, "No point in trying to sprint the marathon here. We have three months to go, and I'm guessing we'll need what's in those fellas' reserve tanks between now and then."

Jamal caught up with Dale shortly afterward. "Dale," he said, "I appreciate the offer, but I'd like to take a rain check and perhaps ask you to pull a weekend later in the project. Instead, why don't you just give me that magic checklist you have, and I'll smoke test the build myself?"

	A	B	C	E	F	G	H	I	J	K	L	M	N	O
1			Sumatra Test Case Summary											
2			Integration and System Test Pass 5 (12/16-12/27)											
3														
4	Assigned	Test			Test	Bug	Bug	Run	Plan	Act	Plan	Actual	Test	
5	To	ID	Test Suite/Case	Status	Environment	ID	RPN	By	Date	Date	Effort	Effort	Duration	Comment
70		*7.000*	*Security and Privacy*											
71	BEZ	7.001	Legal Actions Blocked						12/20		5.5			Replan: JHB 12/23
72	LTW	7.002	Illegal Actions Allowed						12/19		3.5			Replan: JHB 12/20
73	JW	7.003	Logins/Passwords						12/18		3.5			Replan: JHB 12/19
74	JW	7.004	Sharing						12/17		4.5			Replan: JHB 12/18
75			**Suite Summary**						**12/20**	**1/0**	**17.0**	**0.0**	**0.0**	

FIGURE 13-6 TEST CASE SUMMARY SHOWING THE SECURITY AND PRIVACY TEST SUITE REPLAN

Dale smiled, "Sure. I had other things in mind anyway, but I thought I'd offer."

Jamal then coordinated with Petra, the system administrator who usually performed the test release installation, to come in over the weekend. She agreed, perhaps more willing than she otherwise would have been, after hearing that Jamal would be there to work with her, smoke testing the build himself.

Finally, on Saturday afternoon, the testing cycle was done and the new release smoke tested. With the first cycle of pass 5 in, Jamal finalized the metrics and reports for the pass. (In Chapter 15, we'll see how Jamal presents these findings in the Monday Sumatra project status meeting.) He then checked the various documents he'd updated and created into the project repository.

Step #	Step	Done?
7.	Periodically report test cycle findings and status.	☑
8.	Check any status documents or other useful permanent records produced into the project library or configuration management system. Place the document under change control.	☑

Recognize a Good Test Execution Process

Like the tip of an iceberg, the test execution process is the part that shows, but it floats on all the good work in the preceding eight processes. What are other test team behaviors and achievements that distinguish a good test execution process?

Find the Scary Stuff First

When thinking through the test execution process, I use various sources of information to make an educated guess about where the nastiest bugs live, and I test those areas first. When I'm first starting test execution, I use the quality risk analysis discussed in Chapter 2 to guide where I test first. Later, as I get some hands-on testing experience with the system, I have a better idea of the real risks from where I've found bugs—and where I haven't even tested yet. I can use this knowledge to fine-tune my judgment of where to test next. (However, I have to be careful not to let strictly technical considerations—e.g., where we've found bugs—blind me to the fact that customer usage and other business risks need to influence test prioritization, too.) In the case study, Jamal initially focused testing on previously blocked function-

ality tests, along with privacy, security, and error handling tests affected by features new to the latest release.

I'm also concerned about two special kinds of risks to quality: those related to bug fixes that don't actually resolve the problem and bug fixes that create new problems. Those risks are the domain of confirmation testing and regression testing, respectively. Because bug fixes that don't resolve problems are especially problematic, I like to run those tests first in each test cycle.

I extensively discussed strategies for dealing with regression risk in Chapter 6, but here let me point out that test execution is the time to put those strategies into action. One problem can arise for regression risk strategies that involve repeating tests, in that this approach can conflict with focusing on new tests and confirming bug fixes. I do not recommend throwing out your carefully considered regression strategy at the first sign of project trouble, but I allow myself to reconsider an approach that involves heavy repetition of tests if test results show regression to be a low risk, while other risks turn out to be much higher than originally believed.

Another example of scary stuff we want to find is the stuff we don't expect. A carefully planned, methodical set of test processes such as the one I've outlined in this book does help prevent classic testing blunders.[4] However, it can also create dangerous blind spots. A mistaken assumption during quality risk analysis can ripple through the entire test estimation, planning, and test system development processes, creating large, unseen holes in places that actually are important to test. Simply looking at the bugs you do find to figure out where more bugs exist won't solve this problem, because it tells you nothing about the bugs you aren't finding because you're not testing those areas at all. This is why I now augment many of my test plans to include exploratory testing as part of the overall strategy. In the case study, Jamal included exploratory testing in the estimate discussed in Chapter 3 and in his plan discussed in Chapter 6.[5]

Finally, testers should look for bugs beyond their immediate testing objectives. Especially with precisely scripted manual or automated testing, we can sometimes fall into the trap of ignoring bugs because they don't relate directly to what we're testing. For example, if I'm testing file-open functionality, and I notice a problem with

4. For another perspective on testing blunders, see Brian Marick's paper "Classic Testing Mistakes" at www.testing.com.
5. I recommend Kaner, Falk, and Nguyen's *Testing Computer Software, Second Edition*, and Kaner, Bach, and Pettichord's *Lessons Learned in Software Testing* for more information on exploratory testing techniques, along with James Bach's Web site, www.satisfice.com.

the file-import utility, even if that's the subject of another test, I should spend a little time tracking down and reporting the bug. This is especially true if the in-passing bug I notice *is not covered* by another test. This relates to the topic of opportunity testing, which I discussed in Chapter 2. In general, it's a good idea for testers to take the opportunity to find and report other bugs, provided it doesn't distract from the other testing goals. Part of a tester's job is to represent the quality expectations and experiences of a reasonable user. When I test software, I always look for behavior that if I were a reasonable user, I would find disturbing, confusing, incorrect, or otherwise troublesome. As a test manager, I continually remind my test team to do the same thing.

PRODUCE, GATHER, AND DISSEMINATE VALUABLE INFORMATION

Just as drug companies test drugs and engineers test engineering materials, we test software in order to learn something about it. The objective of the process is information. So, it makes sense to design our experiments—our tests—in such a way that we maximize the useful information gathered and get that information into the hands of people who can use it.

The primary kind of information dissemination is external. While we'll look at these topics more closely in Chapters 14 and 15, the main objective of testing is to produce information that the project team can use to make smart decisions about quality. The bug triage committee or change control board can prioritize bugs and decide which bugs to fix (see Chapter 16). The developers can decide how to fix those bugs. The management team can decide what course corrections to make in the project's direction. To enable these smart decisions, a good test execution process should provide for capture of test findings in clear reports and for circulation of these findings to the appropriate parties. In the case study, the testers entered bug reports immediately, and the test manager produced a weekly status report of the test cycle findings for management. As discussed in the case study in Chapter 12, Jamal also attends weekly change control board meetings that discuss and prioritize bug reports.

Another kind of information dissemination is internal. The results of all the previous tests can have some bearing on the subsequent tests. A positive example of this arose when Luis talked to Binyamin about the font bug he'd found. However, we also saw a negative example when Emma chose not to read Luis's bug report.

This process breakdown resulted in two negative outcomes. First was rework. Emma had to expend effort to rerun the Solaris performance test, and machine time was lost. Second, and perhaps more seriously, it introduced a one-day delay in finding four

performance-related bugs. You might say, "Oh, well, what does a day matter?" But a project is nothing but a sequence of days, and days add up to weeks after a while.[6] Research on defect removal models has shown that finding and fixing bugs earlier reduces the total number of bugs found, resulting in savings in both money and time.[7] Using peer reviews of bug reports and test status reports can reduce these kinds of miscommunications, but the test manager needs to make sure that testers spend time on these tasks. Jamal's e-mail to his test team is an example of the kind of process that can work and the kind of management direction that can put it in place.[8]

As this example shows, this internal information is not just about the current project but can lead to improving the test system and the test process. You could say that while the test system tests the system under test through the test execution process, the system under test also tests the test system and the test execution process. This means that astute testers and test managers can glean information about the quality of the test system and the test execution process and find opportunities to improve them. In the case study, Luis and Lin-Tsu captured an improvement to one test case based on a serendipitous bug discovery. During initial test selection and assignment, Jamal adjusted the test execution times based on the previous test cycle.

As a final kind of useful internal information, in this case from a perspective of managing the work to be done, the test lead or test manager will want to make sure that the test team is completing all the planned testing activities. Here the earned value section of the test suite summary comes in. In the case study, Jamal looked at the earned value situation and saw that many of the test suites were taking longer than planned. This led him to analyze specific test suites remaining to be done. He realized that certain tests might not be completed soon enough or even at all without some remedial action. A good test process allows this kind of management perspective to see these situations as they develop, and take steps to mitigate the risks.

---

6. Fred Brooks discussed the danger of such small slips and the corrosive effects such attitudes about small slips have on a project over 25 years ago in *The Mythical Man-Month*. More recently, Tom DeMarco wrote in *The Deadline* that "there are an infinite number of ways to lose a day on a project, but not one single way to get a lost day back." Jealously guard your team's time and energy early in the project when there seems to be plenty of it, and you'll be glad you did later when time is tight and energy is flagging.
7. See, for example, Stephen Kan's book *Metrics and Models in Software Quality Engineering, Second Edition.*
8. Notice that Jamal recognized that the failure arose from a problem in the process, not a problem in the people. I mentioned Deming's red bead experiment earlier in this book as an example of why management needs to take responsibility for process deficiencies. This is another example. Rather than castigating Emma for the natural outcome of following the process as she understood it, Jamal accepted responsibility for the incident and fixed the process.

CORRECTLY INTERPRET TEST RESULTS

Of course, a characteristic that distinguishes valuable information from worthless noise is accuracy. In the great majority of cases, the test results we report should be correct. In the test execution process described earlier, substeps 3.C, 3.D, and 3.E ask the tester to check into discrepancies in test results carefully and distinguish between bugs in the system under test and bugs in the test system. Defining three distinct steps—as opposed to one step that says, "File a bug report if the expected and actual results don't match"—draws attention to the need to correctly interpret the test results.

Test execution is a human process. Since we are humans, our powers of observation and judgment are sometimes flawed. Our expectations of correct behavior are sometimes wrong. Inscrutable systems under test sometimes conceal evidence of correct and incorrect behavior. We might mistake a problem in the test environment, scripts, tools, or other parts of the test system for a problem in the system under test. For these and other reasons, we might report a failure when the program behaves correctly or, worse yet, vice versa (see Table 13-1).

These kinds of errors shouldn't surprise us. To see why, let's zoom in on a graphical view of step 3 of the test execution process. Figure 13-7 shows such a view. With all these different moving parts, misinterpretation is going to be an issue. Add to that the fact that other testers and project team members are likely to interact with the test system and the system under test while we're running tests, and such mistakes become all the more likely.[9]

To deal with tester observation and judgment problems, I have had senior testers review junior testers' results and assign the same test cases to different testers during

	Tester Reports . . .	
Behavior Is . . .	**Pass**	**Fail**
Correct	Increased confidence	Misplaced concern
Incorrect	False confidence	Useful diagnostic information

TABLE 13-1 THE OUTCOME OF VALID AND INVALID TEST RESULT INTERPRETATION

9. One of the reviewers, Steve Splaine, pointed out to me that this is especially true "if no effort has been made to partition [or] coordinate the use...of shared tests and data." This is certainly the case, and the potential problems become all the more acute the larger the team and the more complex the test environment.

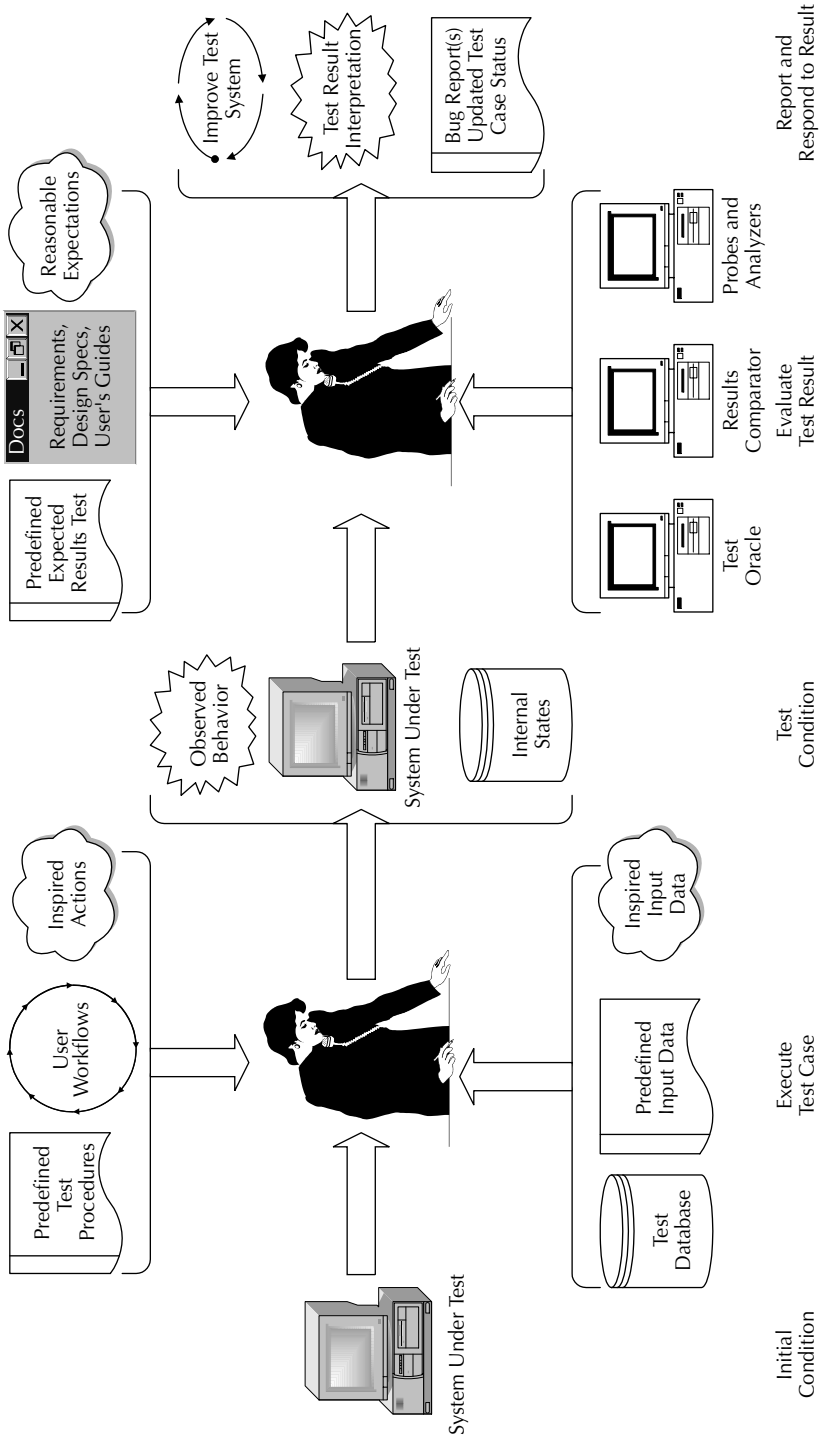

FIGURE 13-7 TEST CASE EXECUTION

subsequent cycles. Completely resolving the problem of recognizing correct and incorrect behavior—the oracle problem discussed in Chapter 10—depends on having unambiguous requirements and design specifications and a foolproof way of predicting the right response to any stimulus. When I lack these items—as I typically do—I generally err on the side of reporting bugs. (That said, see Chapter 14 for ways to deal with the political ramifications of reporting spurious bugs, and Chapter 16 for ideas about how a good change management process can resolves questions about gray-area bug reports.) The inscrutable software issue boils down to writing testable software. Only senior management can resolve this issue by ensuring test involvement during the program design and development.

What should testers do when they find a problem in the test system during test execution? The process I defined earlier calls for immediate resolution of the broken test. Some people would advocate putting the test system through the same process as the system under test: Report the bug in the tests, let the bug triage committee decide whether to fix it, gather metrics on quality problems with the test system—the whole nine yards. In other words, since the test system tests the system under test and the system under test tests the test system, this symmetry means we should follow the process when the shoe's on the other foot.

On most of the projects I've worked on, this process wouldn't make sense. After all, let's get back to the question of quality. If we accept that the customer or user is the final arbiter of quality, then the system under test and the test system have distinctly different customers and users. The system under test is an external deliverable, designed to solve a real-world problem for the actual paying customers and users. The test system is typically an internal deliverable used by the same people who developed it to discharge their testing responsibilities. The system under test is a product. The test system is a tool that helps us make the product.

That said, sometimes the people creating the test cases are not the same people who are running them. For example, consultancies sometimes build test systems and tools for companies. Also, in some cases, quality problems in the test system could have unacceptable consequences—e.g., in safety-critical systems. In such cases, instituting a testing process for the test system may add the level of formality needed to ensure that a quality test system is delivered.

Finally, testers sometimes can't know with complete certainty which of the four quadrants in Table 13-1 they're in without spending an inappropriate amount of time. Researching bugs and chasing down other unusual or promising (for bug discovery) behaviors involve unpredictable amounts of effort. So, the actual number of person-hours required to complete a set of tests can deviate considerably from the plan.

Testers need to remain cognizant of where their time is being spent. Test leads and managers must help their teams balance the need to make forward progress through the planned test cases against the need to know the meaning of their test findings with total certainty.

Not all bugs deserve the same level of research. An intermittent bug where an obscure error message appears in the wrong font once every ten or so times it appears can perhaps be safely ignored so as not to delay other tests for hours. However, an intermittent bug that sporadically corrupts a shared database may warrant postponing much of the test cycle until the failure has been isolated and the problem is reproducible.

EXHIBIT THE PROPER ATTITUDES

The topic of correct result interpretation brings us to attitudes. A good testing process should reinforce the proper attitudes of good testers, and the proper attitudes of the testers likewise enable a good testing process to produce useful results. As I mentioned in Chapter 8, the proper attitudes of a good tester include professional pessimism, balanced curiosity, and focus. Because testers need to remain skeptical and play the professional pessimist's role in the development project, I advocate an active bias toward reporting a bug when testing. This has the following implications.

- If in doubt, the testers assume that the observed behavior is incorrect until they satisfy themselves otherwise.
- The testers report as a bug any situation where the on-screen help, user's guides, or any other documentation indicates that correct behavior differs from that observed.
- The testers *always* report any event leading to catastrophic behaviors, no matter how intermittent or irreproducible. Such behaviors include any loss of data, a crash, freeze, or hang, or any other reliability, data quality, or severe performance degradation problem involving the system under test, the host system, any cohabiting software, or any interfacing or communicating system.
- Finally, the testers report any circumstance where the system under test does not conform to reasonable expectations of program behavior or quality, or is otherwise confusing, misleading, or ambiguous.

This way of thinking, however, must be balanced against where you are in the project. In the early stages of testing, system quality is often very bad. Testers should focus their efforts at this stage on finding the scary stuff, not worrying about misspellings in the user's guide, for example. Once the system improves, what is often criticized as nitpicky test result interpretation is actually an appropriate attempt by

testers to polish the fit and finish of the product. We should not be ashamed to file low-priority bug reports at the appropriate time in the project. This is a form of balanced curiosity.

Focus implies that good testers are alert for discrepancies in interpretations of what it would mean for the system to be *behaving correctly*. Such discrepancies between interpretations can arise not just between a tester and a programmer, but also between two programmers writing communicating components, which is the source of most bugs found during a well-executed integration test phase. A classic example of this kind of bug occurred on a NASA Mars mission where confusion arose about the use of metric or English units for data, particularly in the measurement of velocity and braking force. The use of different units by different programmers resulted in the total loss of the mission—and any valuable information the mission might have generated—when the lander hit the Martian surface at high speed and presumably disintegrated.

In such cases, the bug exists not in one component or the other, but in the interaction between components. Before a single line of buggy code existed, though, the bug simultaneously existed in the heads of two different human beings. Had the right person—a tester, perhaps—at the right time—during requirements specification—thought to ask both of those people the right question—"About the units: English or metric?"—the entire unfortunate scenario could have been avoided.

So, testers must learn to be vigilant for the signs of disagreement or different understandings within the programming organizations. Sometimes asking "dumb questions" such as "I'm sorry, but I'm confused; are we using metric units or English units on this project?" can stimulate discussions that surface these misunderstandings before they result in dangerous, expensive, hard-to-diagnose bugs during system test—or, worse yet, later. This is a major source of value for test organizations. As Fred Brooks wrote in the first edition of *The Mythical Man-Month* over 25 years ago—about projects that had happened in the 1960s—"Long before any code exists, the specification must be handed to an outside testing group to be scrutinized for completeness and clarity. As [V. A.] Vyssotsky [of Bell Lab's Safeguard Project] says, the developers themselves cannot do this: 'They won't tell you they don't understand it; they will happily invent their way through the gaps and obscurities.' "[10]

A bug is a bug is a bug, whether it lives in a component, lives in the intersection of all the components, lives in the whole system, lives in the design, or lives in the requirements. The most effective test teams I've worked on have had implicit or explicit per-

10. See *The Mythical Man-Month*, page 142.

mission to report any of these kinds of bug and have been involved early enough to do some good in each area.

PROCEED EFFICIENTLY

To some extent, an efficient test process relies on an efficient test system. This brings us back to the test system development issues discuss in Chapter 11. For example, minimizing test case dependencies and sharing of test data sets prevents unnecessary rework, delays, and possible misinterpretation of test results.

However, efficiency considerations also arise during test execution, even with the most efficient test system. As a test manager, I want the minimal amount of overlapping work and avoidable delays in testing. Part of this involves making clear to each tester which tests they should run and when. In the case study, Jamal used the Assigned To column in the test case summary worksheet to identify an owner for each test. He used the Plan Date column to order the tests for maximum efficiency and to generate information in the right order. Both assignments and planned dates changed for some test cases as the cycle proceeded—e.g., when Jamal took over the security and privacy tests. In such situations, the test lead or manager also needs to communicate changes in assignment and planned dates to the test team.

This may sound like a trivial matter, but it's easy for confusion to creep in here, with two serious consequences. The mismanagement of two precious resources (tester effort and schedule time) entrusted to the test manager is bad enough. Frequently, test managers will require yet more overtime from their team to rescue the schedule because of such errors, which constitutes a failure of leadership that detracts from the managers' ability to get the best work from the team.[11]

This points out the need for the test team to allow the process and the planned tests to adapt to evolving circumstances. Part of testing is encountering unexpected behaviors and researching those anomalies to create a detailed, actionable bug report (see Chapter 14). Since the behaviors *are* unexpected, and since we can't predict with 100% confidence how many bugs we'll find, the test process generates its own fluid state of affairs. In addition, external events such as delayed builds, adjustments in build content, and shifts in management priorities bring change to bear. Testers can't prevent these changes, so the test process must accommodate them. In the case study,

11. Disrespecting the team's time is a management deadly sin. Once I witnessed the resignation—in a profane, abrupt, and emotional outburst in the middle of a meeting—of the lead system architect on a project after such an incident. For more insights on this and other management failures, see Tom DeMarco and Tim Lister's book *Peopleware*.

Jamal adapted to changes from within—a miscommunication on bug findings—as well as changes from without—a high level of bugginess in SpeedyWriter and a system administrator's negligent shutdown of a test server. Jamal adjusted the test case assignments, added extra resources, changed the planned execution dates, and pushed lower-priority test cases out to the following test cycle.

As an aside, consider the waste and delay that arose in the case study when the system administrator outside the testing team brought down the server Luis was using for a test. Test environment configuration management is a topic I dealt with at length in my first book, *Managing the Testing Process*, but let me point out in the context of test execution that the test team simply must have complete control of the test environment during this period.

There's a major breakdown in the process somewhere when the kind of error described in the case study happens more than once or maybe twice during a test project. In situations like the Sumatra test team's, where outside people must have administrator permissions on the test environment because they support it, it can be difficult to ensure that no one modifies the test environment without clearance from the test manager or test lead.

On rare occasions in situations like this, people entrusted to support the test environment prove to be more of a hindrance than a help. I've had instances where system administrators took hardware out of the test lab even though it was clearly and exclusively earmarked for the test effort. In such cases, it's better to try to move the test environment support function into the test team, even if you must reduce the scope of your testing to do it, and simply deny logical (i.e., electronic) and physical access to the test equipment to anyone outside the test team. This can entail secret passwords, firewalls, locks, separate test labs, card keys, and so forth. Otherwise, the test project may end up suffering the death of 10,000 cuts as little incidents like "borrowed" hardware and debugging in the test environment gradually reduce testing efficiency to zero.

Finally, an efficient test process should have the lowest possible overhead. In other words, we should devote the most time possible to testing and other activities that generate value for the organization, and the least time possible on everything else. Notice that the daily debriefing meetings, reading each other's bug reports, and other informal test activities that foster communication and sharing of knowledge are not overheads, they are ways to promote efficiency. However, following an inappropriately complex or onerous process, using documentation-heavy processes where the information will not be used later, having standing meetings where one person reports status to the supervisor while everyone else just waits their turn to talk—all these are examples of overhead.

HANDLE CHALLENGES

Since the test execution process is subject to all sorts of unforeseeable and external pressures, there will be challenges. Let's look at a few of these in the following subsections.

DEALING WITH EXTENDED SHIFTS AND OUTSOURCED TESTING

Back in Chapter 6, Jamal made a decision to use an evening shift. He made that decision because of constraints on the test environment. This is not an uncommon reason for choosing to run an evening—or even a graveyard or weekend—testing shift. I chose to run evening and weekend shifts once when I had a shortage of rare engineering prototype Internet appliances and a lot of tests to run in a short period. When testing must be done on production mainframe systems that simply can't be dedicated to testing during normal business hours, sometimes most testing has to happen during off-hours. On another occasion, the need to keep a temperamental stress test running 24 hours a day meant that we needed to baby-sit the tests much of that time. When I was a project manager for a testing lab, we commonly used both evening and weekend shifts.

Extended shifts are not terribly difficult to manage, as long as you keep a few realities in mind. The primary issue is communication. It's harder for people who aren't in the same place at the same time to communicate, and test execution (along with bug reporting) is definitely part of the testing process where communication between the test team members is critical. In the case study, Luis and Emma had a communication breakdown on the same shift, which shows how easily miscommunications happen. Evening shifts limit communication options, and graveyard and weekend shifts limit them even further. The test team will need to make every effort to promote communication—and the test manager will need to put processes in place to ensure it. On an Internet appliance project, we had a two-hour overlap period from 3:00 PM to 5:00 PM every day. During this period the evening shift was expected to read e-mail from that day and ask follow-up questions before the day shift left.

Extended Shifts

Any off-hours work done by people other than those working during normal business hours (approximately 8:00 AM to 5:00 PM). Typical extended shifts include *evening shift* (approximately 4:00 PM to 1:00 AM), *graveyard shift* (approximately midnight to 9:00 AM), and *weekend* or *holiday shift* (any of the previous shifts, but on nonworkdays). Contrast *overtime*, which is work beyond 40 hours per week, but which can involve any shift.

Part of that communication is status gathering. Daily debriefings or status meetings that happen during a structured shift handoff period help. On the Internet appliance project, we held the handoff meeting at around 4:00 PM. During that period, we talked about the testing that had gone on that day and the evening before, and we updated the test case tracking worksheets.

Another issue is scheduling. Everyone will need to know what shift they're on and on which days. The simplest case is that people work the same shift for the whole project. However, this is not always possible. Some testers may see extended shift work as very onerous, and some kind of shift rotation might be necessary. My experience has been that bonuses paid to evening and weekend shift crews tended to make the extended shifts desirable for certain people, ameliorating this problem.

Another issue is support. Assuming that the test team does not administer the test network, you'll need to line up system administration or network operations support during off-hours. This means support staff must give the test team pager numbers, home phone numbers, cell phone numbers, and the like. They must carry those pagers and cell phones at all times, answer their home phone day and night, and in general agree that their evenings or even early mornings can be interrupted by an urgent test team request. I've found this to be more challenging on my projects than lining up testers willing to work extended shifts. In the case study, Jamal was careful to deal with this issue during the test planning process. If you wait until test execution to iron this out, you can expect to encounter resentment, destructive literal compliance, and other forms of active or passive rebellion, including outright refusal.

Finally, consider logistics. People working extended shifts will need badges with off-hours access. Air conditioning units must be left running during the shift. Alarm codes must be provided. Security guards must be notified. Phone switches and PBXes may need to be configured to allow off-hours usage. Facilities people must be willing to help resolve logistical issues that arise.

Outsourcing some or all of a test effort generally creates the same issues as extended test shifts, only more so. Communication is often entirely via phone and e-mail in these cases except for the occasional on-site visit. Even during on-site visits, it is rare that everyone at the test lab comes to the client's site—or vice versa. In addition, on-site visits can be more about management presentations and sales pitches than nuts-and-bolts tactical interaction between the individual contributors. E-mail and phone communications are often complicated by time zones, which limit the periods of overlapping work and thus the possibility of dialogue on a real-time, interactive, as-needed basis. Test support personnel, such as system administrators, generally do not have a responsibility to provide support for outsource test facilities.

Because of the broad client base of most outsource testing organizations, the specific expertise required to support the system will not be present in their facility. During testing, both minor and serious glitches in installation, maintenance, or operations arise, but the right expert—often wandering the halls not far from the in-house test team—can resolve these in minutes. However, such glitches can render the system unavailable for testing for hours or even days in an outsource test lab. I've seen this happen on more than one testing project.

This section is not a dig against independent testing labs and consultancies—after all, I run one. Nor am I denigrating the value of extended shifts—I use them. However, it's important to plan tests knowing that we will find and run into problems. A test plan that assumes all the tests will pass and the system will work just fine is bound to break down. If you plan your use of extended shifts and outsource testing recognizing the need to handle communication, support, and logistics problems that arise from complex or buggy products, these options will serve you well. In addition, the test team that amplifies its efficiency and effectiveness by leveraging extended shifts and outsource test teams will reap valuable political benefits from being able to achieve better testing in a quicker way, thus building credibility.

ACCOMMODATING HOLIDAYS, CULTURES, AND VACATIONS

Test execution periods—the time of greatest pressure on many projects—often span holidays. Indeed, in the diverse cultural mix present in many companies around the world these days, you might be hard pressed to find a test execution period that didn't span a holiday for one or more of the people on the project.

As Jamal showed in the case study, though, a culturally diverse team is actually a strength, not a weakness, in this regard. On homogeneous projects, everyone will want to take the same holidays off. On diverse projects, many people might be happy to trade holidays like Christmas for their own cultural or religious holidays more attuned to their needs.

Vacations, while not occurring synchronously, do present a challenge in terms of downtime, especially with key players. This can be a problem with tight or high-risk projects, and executive management sometimes cancels all vacation during these times. However, the general case is that someone or another will take a vacation during a long project. Realistically, the other option is that people will burn out and quit.

Whatever the cause of reduced staffing, the watchword is advance notice. I've found that as long as I knew far enough in advance about holidays, cultural requirements,

and vacations, I could plan around them. During test execution, it's important to remember which weeks will be short-staffed. The test manager can then shift testing work around those weeks.

CAPTURING THE HISTORY ON THE GROUND

Capturing historical information is especially critical when you have outside dependencies. If you can't get through your planned testing because the test environment is down, that may not be your fault. Perhaps the operations team—charged with supporting the test environment—isn't responding to pages and calls for help. Perhaps competing priorities prevent them from responding as agreed in the test plan. In that case, management needs to understand what these competing priorities are doing to the test effort.

Of course, it is possible that outside teams upon whom you depend are actually opposed to the success of the testing effort or the project as a whole. For obvious political reasons, keeping accurate logs in such situations is a survival issue. While the black arts of corporate politics are beyond the scope of this book, in addition to keeping detailed logs of how external dependencies affected test execution, certainly you'll want to deal with this situation with care and adequate documentation.

IMPLEMENT IMPROVEMENTS

While test execution is tightly bound to the context of the testing effort and the rest of the project, there are some things you can do to fine-tune test execution itself in many cases.

1. Assess where you are and what portions of the test execution process are not under control, including other processes discussed in this book. Maybe some of the sources of chaos are external. Can you figure out a way to reduce the leakage of external chaos into your testing effort?
2. Ensure that you have some idea of what you intend to test and how long it will take to perform those tests, broken down into bite-sized pieces. Two- to four-hour test tasks are sizes that I find very manageable. This need not take the form of detailed, scripted test cases where you count the number of minutes required on average to execute each detailed, unambiguous step. Simply bounding a test charter (e.g., test file manipulation capabilities) with a planned duration (e.g., spend two hours testing file manipulation capabilities) can suffice.

3. Put in place some way of tracking test cases—either using the test case summary and test suite summary worksheets I showed earlier or some other mechanism—both in terms of duration and schedule and in terms of findings. Again, you can start with something very simple, and you may find that this simple technique suffices.
4. Use the effort data you gather in tracking test cases, along with your predictions about the bugs found at various points in a test phase, to predict the amount of time you should allocate for dealing with bugs. A classic pitfall of testing is to fail to anticipate the presence of bugs in the system under test. This results in scheduling tests as if they will pass, which underestimates the effort and duration, often severely.[12]
5. Deal with the need for internal communication and information gathering. A lot of testing problems arise when the testers don't know what one another is doing.

All these steps can be important way stations on the road to getting your test execution processes under control.

Competent testers execute tests with discipline and imagination, and the competent test manager manages the test execution process crisply. The test team adapts and responds to unexpected events and findings, and adds value consistently for the organization. The spotlight is on the test team during test execution.

With the end of this chapter, we wind down our discussion of the Perform step in the test process. Deft performance of the tests generates crisp, complete, and accurate information about the quality of the system. This is the service and product we can deliver to developers, to peer-level managers, to project managers, and even to senior executives.

The delivery of this product and service provides the project team with an opportunity to perfect the system. It also provides us with an opportunity to perfect the test system. In the final chapters of this book, we'll look at reporting results, managing changes to the system, and, ultimately, improving the testing itself.

12. The habit of planning tests assuming those tests will fail, not pass, has become so ingrained in my thinking that I forgot to mention it initially. I thank Tim Koomen for pointing out this omission.

Step #	Step	Done?
3.A	Perform: Do the testing and gather the results.	☑
3.D	Acquire and install a test release consisting of some or all of the components in the system under test.	☑
3.F	Assign, track, and manage the set of test cases to be run against each test release.	☑