Test Estimation

Tools and techniques for realistic predictions of your test effort

by Rex Black

As a test manager, development manager, or test lead, at the beginning of many projects, you probably find yourself confronted with the question, "How long will it take—and what resources do you need—to test this system?" Maybe your boss asked you. Or maybe your boss just gave you an end date, and you're wondering if you can actually hit that date. Or maybe you're bidding on a project for an important client. Or maybe you are putting together a proposal for a new in-house project, and the stakeholders for that project need to know how long the testing will take. No matter what the impetus, it is important to know how to estimate testing projects.

An estimate should accurately predict and guide the project's future. Such an estimate is

1. Realistic. It includes all the tasks that you can reasonably anticipate. It forecasts what, based on our current knowledge, is most likely to happen. It reveals the risks to a test project so you can take steps to mitigate them.
2. Actionable. I want clear ownership of tasks by committed individual contributors. I want to see assigned resources and known dependencies.
3. Flexible. What if deadline and resources constraints are immovable? The estimate must accommodate project realities.

Some Estimation Terminology

Project: Temporary endeavor undertaken to create or provide service or product.

Test subproject: The subset of the project performed by the test team to provide test services to the project.

Project schedule (a.k.a., work-breakdown-structure): Hierarchical decomposition of project into phases, activities, and tasks with resources and dependencies.

Check the Project Management Institute's Web site (www.pmi.org) for more useful definitions and ideas about project management and estimation.

Divide and Conquer

One excellent estimation tool is a *work-breakdown-structure.* A work-breakdown-structure (WBS) is a hierarchical decomposition of a project (in this case, the test effort) into stages, activities, and tasks. For testing projects, start with the following stages:

- Planning
- Staffing (if applicable)
- Test environment acquisition and configuration
- Test development
- Test execution (including find / fix / retest cycles)

Those with programming backgrounds may be familiar with the technique *divide and conquer*. A WBS works in a similar way. Once you have determined the stages, *divide* them into ever-smaller chunks of work, ultimately down to the level of one person over a short period of time (one to five business days.) Then *conquer* the estimation problem by understanding how long (duration) and how much work (effort) each task will take. Because the overall effort and duration estimate derives from the lowest-level constituent tasks, such calculations are called *bottom-up estimates*.

To measure when tasks are completely finished during the project, it helps to ensure that each task produces a key deliverable, or at least a first draft or some measurable piece of a key deliverable. These deliverables may be internal to the test team, like test cases, test tools, or test data. They may be deliverables *in to* the test team, like the first feature-complete test release, the unit test results, or the configuration of the test environments. They may be deliverables *out to* the project team, like test plans, bug reporting systems, and test results. These deliverables are often inputs to subsequent tasks.

Let's use a hypothetical case study to bring these dry ideas to life. Step into the shoes of the Test Manager on the "Grays and Blues" project. This project team is building an exciting, first-person action game based on the history of the United States' Civil War. Your Test Team consists of two skilled test engineers and five junior but eager test technicians. You will run the System Test phase of the project—the Development Team owns the Component and Integration Phases—which is proceeding under the "V model" software development lifecycle methodology. (I talk more about testing and development lifecycles in my book, *Managing the Testing Process*.) The test strategy is to use scripted, manual test cases with some automated load and reliability tests.

As the test manager, you sit down to create a WBS for the Grays and Blues test project. You can use a project management tool like Microsoft Project, or you can tape 3x5 index cards (or stick Post-It Notes™) to a white board. (Often, people who use the index card technique will enter their estimates into a project management tool after they're done.) Initially, you might come up with the WBS shown in Figure 1.

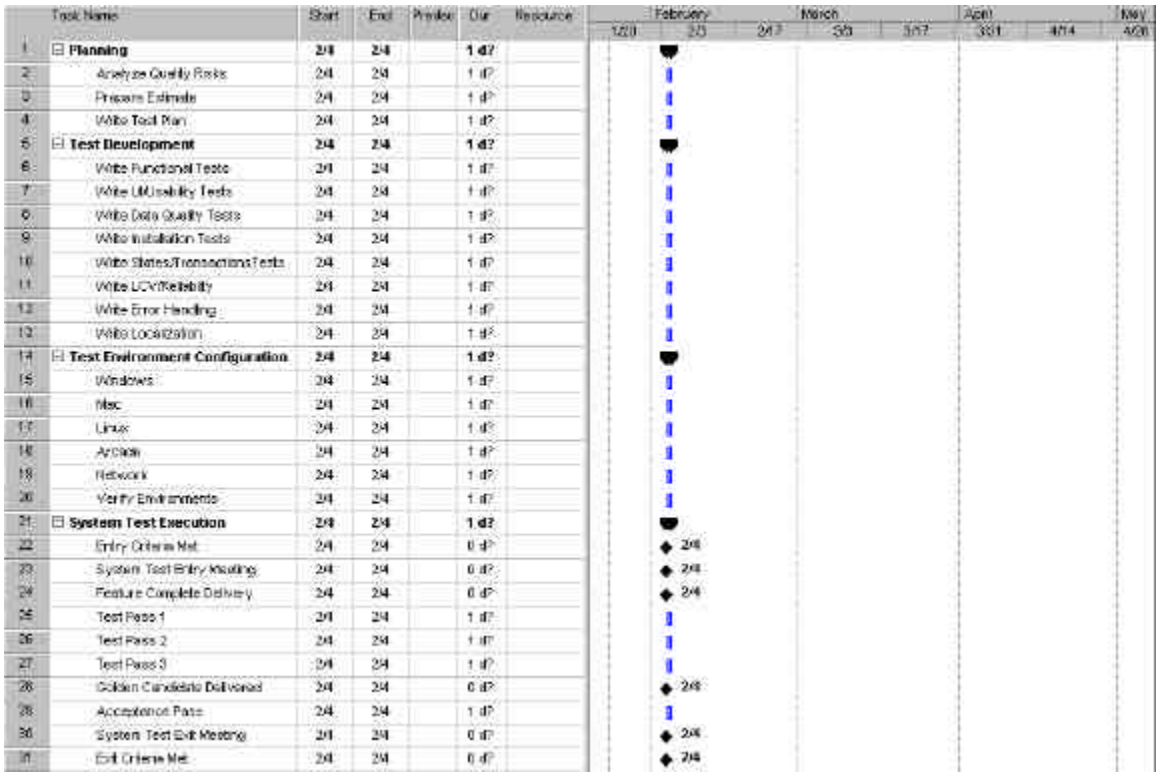

Figure 1: An Initial Work-Breakdown-Structure

Unite and Estimate

Decomposing the work into tasks can be done alone, but accurate predictions about how long those tasks will take requires the collective wisdom of the technically competent people on the team: those who know how long the tasks really take because they've done them before. (Some managers like to gather the team together to develop the initial WBS; others bring the team a first draft WBS that serves as a springboard for the discussion. It's up to you.)

Using the whole team leverages varied and extensive experience. On a team-building note, involving the team in estimation sends a powerful message of trust and collegiality. It also builds commitment to the estimate.

There are three major techniques for team estimation. One is the *Delphic Oracle*, where each team member estimates how long each task will take. During estimate review, the high and low estimators for each task explain their reasoning. The low estimator may point out an optimization or trick that can speed up a task, like using a random number generator, a spreadsheet, and cut-and-paste to generate large amounts of test data rather than entering it by hand. The high estimator may point out a likely delay, like the process of getting imported hardware prototypes through customs. The estimation process is then repeated twice more, taking the high and low estimators' points into account each time. The average for each task at the end of the last iteration is the estimate.

The next approach, called the *Three Point*, involves asking people not for one estimate, but for three estimates of each task. The first number is the best case estimate; i.e., assume everything goes well. The second number is the worst-case estimate; i.e., assume our worst fears are realized. The third number is the expected-case estimate. The average of the expected cases is the final estimate, but the best case and worst-case estimates are documented to understand the accuracy of the estimate and to feed into the test planning and risk management processes.

Finally, the combination of Delphic Oracle and Three Point techniques is called the *Wideband* technique. Team members give three numbers. The low and high estimators for each of the three numbers on each task explain their estimate. The process is repeated twice, then the averages of the expected case estimates become the final estimate. The average best- and worst-case numbers for each task become the range.

Names like "Delphic Oracle" serve as reminders that you are trying to foretell the future. Risky business, that. As the project proceeds, you will learn new things that would have affected the estimate. Changes will happen. Flaky new technology won't work right the first time. So, it's a good idea to include some contingency time—some slack—into your schedule, especially the riskiest tasks. A rule of thumb that's worked for me is 20%, but you'll want to look back at past estimates and see how far off initial estimates were from final end dates to come up with a good rule of thumb for you.

Hop back into your Grays and Blues shoes again. Suppose you sit down with your team and go through the WBS, assigning durations and effort to each task using the Wideband technique. In the process, you find a missing activity, Smoke Test. In addition, the exercise of discussing the highest worst-case estimates identifies a number of key risks to the testing project that you will want to put contingency plans in place for. The sized estimate is shown in Figure 2.

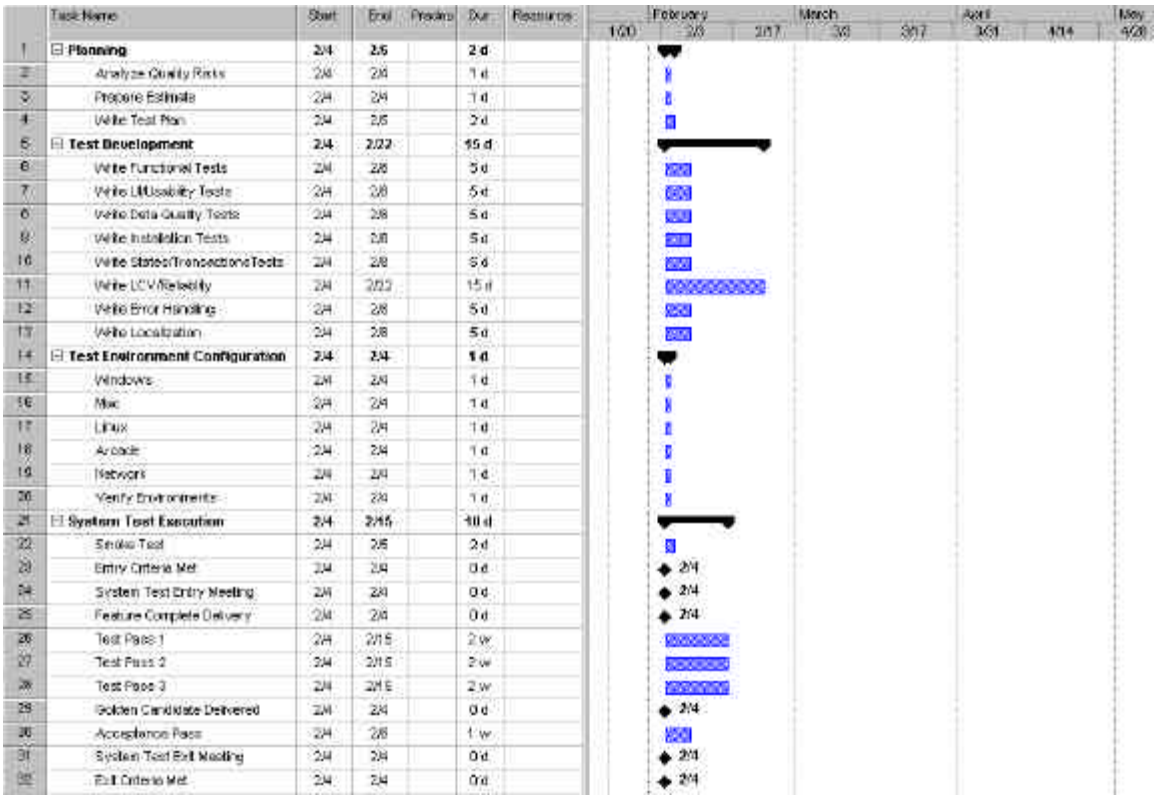

Figure 2: An Estimate with Tasks Sized Using the Wideband Technique

It Depends

The project shown in Figure 2 looks very strange. It seems like, given enough people and other resources, it could finish in fifteen days, the length of the longest task, Write LCV / Reliability (LCV is an abbreviation in this figure for Load, Capacity, and Volume.). But are resources the only constraint on your schedule? Of course not! Some tasks have dependencies on other tasks. The tasks that come before are called predecessor tasks, and the ones that follow are successor tasks. Successor tasks cannot start—or at least cannot complete—until their predecessor task(s) are at some stage of completion.

Of course some deliverables cannot be started—or finished—until others are finished. Dependencies often arise through the deliverables. For example, you might want a completed, approved test plan before you start test development. This is an internal test team deliverable, and is called a *finish-to-start* dependency. As another example, suppose you decide that you want to continue system testing three weeks beyond both the completion of new feature development and also the delivery of the last major change for testing. In that case, you have a *finish-to-finish* dependency.

If you're using the 3x5 or PostIt Note™ method, you and the team can use the following approach to identify the dependencies:

1. Stick all the tasks that have no dependencies to the whiteboard first, at the far left edge.
2. Now, add the tasks that depend in some way on the tasks currently on the whiteboard (and only those tasks). Tape or stick them to the right of those tasks upon which they depend. With a marker, draw a line representing the dependency that connects the successor and predecessor tasks. Identify what kind of dependency it is.
3. Repeat step two until you and the team have identified all dependencies.

For small projects, you might rather have everyone huddle around a PC and plug the data directly into the project management tool.

If you have used the 3x5/whiteboard approach, now is the time to enter the information into a project management tool, because the next step is to look at the *critical paths*. The critical paths are those sequences where delay, slippage, or exceeding the allotted time for any task along the sequence will cause a day-for-day delay in the project end date. *Near critical paths* are those where a delay of a day or two might not affect the schedule, but significant delays will. For example, those tasks that affect phase entry or exit criteria are often on the critical path, as many dependencies tend to converge around phase entry and exit dates. External dependencies are another frequent source of delay for projects. Analyzing the critical paths identifies those tasks that are at the highest risk of delaying the schedule. Such tasks require careful management and attention during the project.

No Free Lunch

As part of the team estimation exercise, resources are assigned to tasks (some managers come to the team estimation meeting with pre-determined resources for some tasks). The exact set of resources required is very project-specific, of course, but generally fall into the categories of

- people
- test environments
- test tools and testware.

People resources include test engineers and test technicians, whether contractors or employees, as well as outside test resources like test labs and vendor test groups. Remember that using a less-skilled person to accomplish a particular task—due to a shortage of a particular skillset, perhaps—will increase the effort and duration of that task, so be ready to revise the estimate based on the people assigned. Also, keep in mind that a person's skill with a given task or tool determines the accuracy of the estimate. Unless, for each task on your schedule,

you have at least one person on your team who knows how to perform that task, the estimate will be subject to delays for those tasks for which skilled people are unavailable. If you find yourself understaffed or without skilled people in certain areas, you'll need to talk to your manager about hiring a new tester, bringing in a contractor, or reducing the scope of the test effort.

Test environment resources include hardware, software, networks, facilities, and so forth. You needn't account for every pen and piece of paper, usually, but it's especially important to include expensive or long-lead-time items like large servers and lab space.

Test tools and testware include custom (new or reused) test data, test cases, test scripts (manual or automated), and test harnesses, along with the widely-known commercial test tools. In many cases, test tools and testware are deliverables from the early stages of the test project.

Back in the land of Grays and Blues, our WBS with dependencies and resources assigned might appear as shown in Figure 3.

As the test manager for Grays and Blues, you've taken care to avoid some common estimation pitfalls, like

- assuming that two people can finish a task in half the time of one.
- overloading test tools (e.g., insufficient licenses) or the test environment (e.g., performance and load testing on the same systems as functional testing).
- forgetting to include time and resources to set up and support the environments and tools.

These mistakes can cause lengthy delays on projects when critical dependencies delay successor tasks.

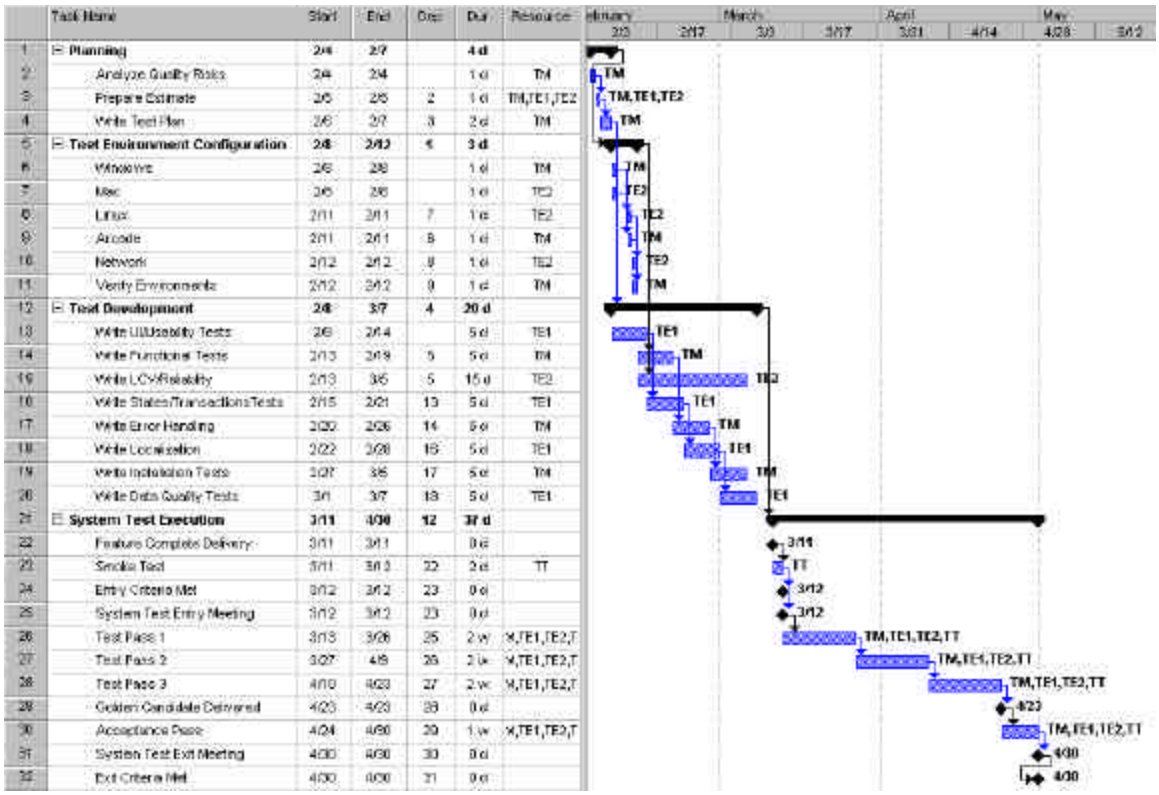

Figure 3: A WBS with Dependencies and Resources Assigned

Estimating Test Execution

The test execution stage is often particularly hard to estimate. How long it will take to finish running the tests is really a function of two questions: 1) how long will it take to run each planned test (scripted or exploratory) at least once, and 2) when will we be done finding bugs, fixing them, and confirming they're fixed?

To deal with the first question first, you have to know three things to estimate the planned test time.

1. How many total person-hours of testing have you planned? Suppose you have estimated the number of hours required for each Gray and Blue test case. You come up with a total of 280 hours of effort.
2. How many raw person-hours of staff time do you have available per week? Our Gray and Blue test team includes 7 people, and let's assume they work 40 hours per week, so we have 280 person-hours available.
3. What percentage of the time a tester is on site is spent, on average, running test cases? (Testers often must attend meetings, confirm bug closure, update test scripts, read e-mail, and do other productive things that are not related to planned test execution during each week.) Assuming our Gray and Blue testers spend 50% of their time testing, we have 140 person-hours of testing per week.

This would mean that we need two weeks to run each test once.

Now to the harder question: How long to find all the bugs? Stephen Kan in *Metrics and Models in Software Quality Engineering* and Pankoj Jalote in *CMM in Practice* both advance what are known as defect removal models (see StickyNotes for more on those references). A simplified version of building such a model is as follows.

First, we need to have some prediction of the *total number of bugs*. Function point and lines of code counts are commonly used, but might be beyond your process capability. Suppose you have the total number of person-hours estimated for the project. If so, can you look back at previous projects and see how many person-hours were estimated, and how many bugs you ultimately found? If not, maybe you have some other historical data you can use, like the average number of defects found per feature or per programmer. The idea is to create a simple mathematical model—using a spreadsheet, say—that calculates, using a metric or two that you have during the estimation period, the total number of bugs ultimately found during testing.

Second, given a prediction of the total number of bugs, *how long will it take to find them*? Again, with historical data, perhaps we can calculate what percentage of the remaining bugs are typically found each week during system test. Now, how long to fix them all and confirm them fixed? Again, based on historical data, what percentage of the open bugs get fixed each week? As you derive absolute numbers from the percentages, take care to check these numbers against the capability of the test and development teams. If you project a peak bug find rate of 200 per week for your Gray and Blue team of 7, ask yourself, “Can the testers really report over five bugs per day?” Now, add this information to your spreadsheet, building a simple model for the cumulative bug open and close numbers.

If, as the Grays and Blues test manager, you come up with a defect removal model that predicts the find and fix rates shown in Figure , then the test execution schedule in Figure 3 is reasonable. The more historical data you have for projects like your current project, the more accurate these predictions will be. For instance, using historical data, one of my clients predicts total bug reports with 10% accuracy on projects as long as two years. (You can download the spreadsheet used to generate this chart from my Web site as part of the templates that accompany *Managing the Testing Process*. See StickyNotes for details.)

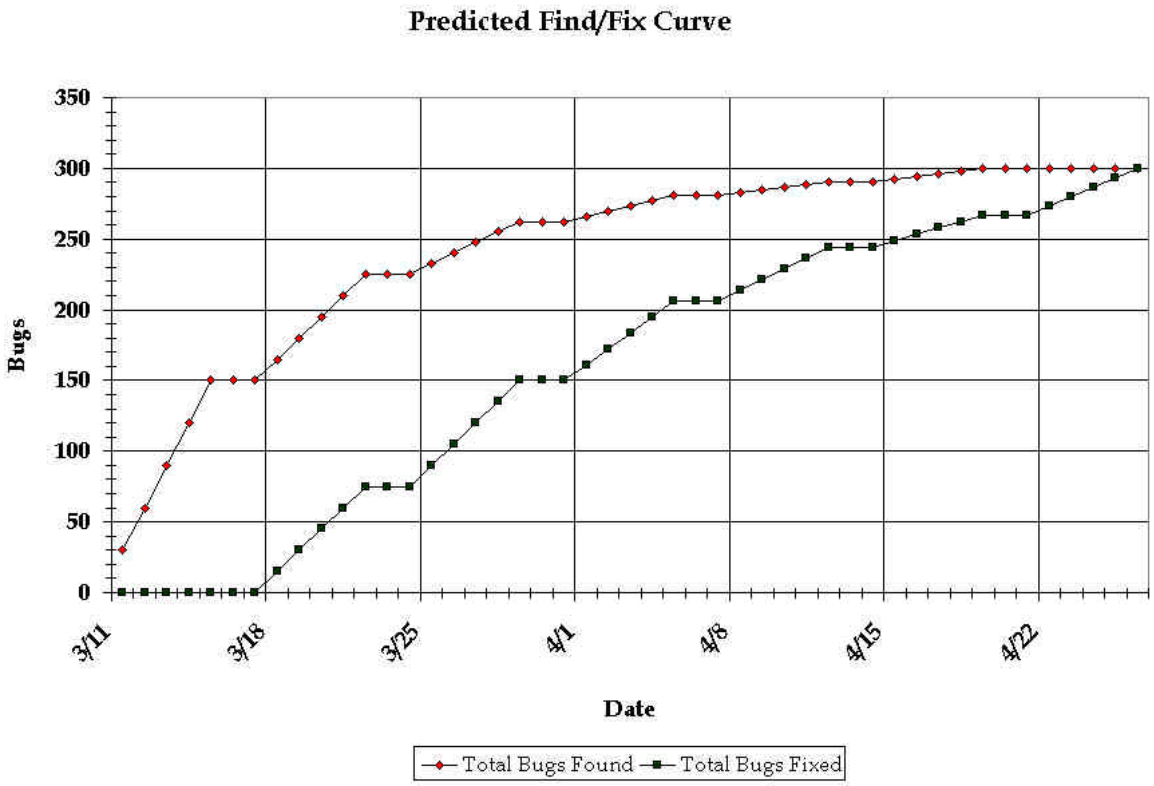

Figure 4: A Predicted Find/Fix Chart, Based on Historical Data

Factors that Influence Testing Estimates

System engineering—including the testing—is a complex, high-risk, human endeavor. As such, it's important to combine good estimation techniques with an understanding of the factors that can influence effort, time, dependencies, and resources. Some of these factors—processes, tools, test environment, skills, team composition and management—can act to slow down or speed up the schedule, while others, when present, can only slow things down.

When preparing a test estimate, it's important for the test manager and those on the test team who help with estimation to consider how each of these factors will affect the estimate. Forgetting just one of these factors can turn a realistic estimate into an unrealistic one. For an explanation of factors that can effect test estimates, see my article "Factors affecting test estimation," published on StickyMinds.com.

But We Don't Have Until April 30th!

Suppose you propose your realistic, actionable schedule to management, and they say, "Make it shorter! We don't have that long!" Now what? Reluctantly or petulantly accept a date from management? That's not a good plan. How about some other options?

One thing you could do is relax your entry criteria for testing a bit. Oftentimes, people define entry criteria as, "one phase must end before the next begins." But suppose you waive the “feature complete” entry criteria for system testing and accept an “almost complete” release. If you did this in the Grays and Blues example, you could pull in your estimated test completion date by two weeks. That’s good; but people must understand that this overlap can increase risks to system quality. For one thing, can developers really fix bugs and finish the system at same time? If not, what will suffer, bugs or features? For another thing, testing an unready system often is less efficient, which might result in less—and less thorough—testing.

Another solution is to add staff. Suppose you can increase the test team and programming team staff so that you can run test passes in a week and fix bugs twice as fast, too? In the Grays and Blues example, this would pull in the estimated end date by five weeks. Now March 26 is the new end date, which is 40% earlier. But the human resource cost would increase significantly, perhaps even doubling. Furthermore, it’s not certain that you could hire new people and ramp up them up in time to make a positive contribution to this project.

An oft-proposed solution to schedule crunches is an arbitrary reduction in test execution time. “Hmm,” people think, “maybe we can accept a lesser level of quality?” Okay, suppose you cut test execution in half? This eliminates the scary staffing budget increase, but will you *really* be done testing? Cutting test execution is the most risky strategy for schedule compression. In terms of schedule, what if the system is not good enough at end date? Now you have a schedule slip right at the end. In terms of budget, this schedule slip occurs when the project’s daily cost is at the highest point. In terms of quality, bugs will be found late in testing, leaving no time to remove them. You might have to drop buggy functions late in the game, after you’ve already expended most of the time and money to implement those features.

If you’re going to take risks with quality, take those risks with your eyes open. To do so, carefully consider the comparative levels of risk associated with reducing the testing effort. One option is to eliminate whole areas of test coverage. Identify the lowest-risk areas in the scope of testing, and then either drop coverage or test in those areas only if you can leverage other testing that brings you to that area anyway. Another option is to reduce the extent of testing across the board. Identify the least risky of the highest-risk areas, and adopt a balanced (broad, not deep) test approach. You can also postpone automation of some tests or use outsourcing, especially test labs, to reduce test environment costs. Whatever technique you choose, the idea is to pick creative techniques in a proactive way to reduce the time required for testing.

Instead of cutting testing, you could cut the features in the product. For Grays and Blues, suppose you drop significant chunks of game functionality, like

multiplayer and arcade, Mac, and Linux platforms? This reduces the test effort by about 25%. So, depending on effect on development, you could be done by March 26. This can work, but it's also true that releasing lots of small releases increases the quality risks related to regression. (The limited amount of testing time associated with a small maintenance release makes missing bugs in unchanged areas more likely.) Therefore, these features should be bundled with next regular release.

What *Not* to Propose

While there are a few other approaches to dealing with schedule pressure, I don't recommend any of them. One is weekend or overtime work. If people were machines, you could use 7-day workweeks to shrink schedules by 40%. Now, people will have to work weekends and overtime occasionally. However, it should seldom be part of the schedule or estimate, but rather an attempt to catch up if you fall behind at some point. As Tom DeMarco suggested in *The Deadline*, extended overtime will burn people out and reduce their productivity. The only benefit of long periods of overtime is the cynical one, which is to make managers appear blameless when schedules are missed.

Another risky estimation technique is setting a tight schedule as a "stretch goal" for testers. To have a 50/50 chance of on-time project completion, each task (especially critical and near-critical path tasks) must have a 50/50 chance of finishing on time. Otherwise, you will probably fall further and further behind. Tight schedules only make sense if you believe in what's called "Theory X" management. This theory says that people will only do their best work if cajoled, exhorted, and pressured by their manager. As an alternative, consider "Theory Y." This theory says that people want to do their best work, and it's the manager's job to enable, support, and nurture that work. If theory Y applies to your team, then stretch goals are a losing strategy. Also, as a practical matter, even if heroics and sacrifice lead to on-time delivery for one project, what are the long-term quality and motivation impacts?

Realistic, Actionable Estimates

In a successful project, schedule, budget, features, and quality—the four "moving parts" in a system development effort—converge as the release date approaches. Realistic, actionable estimates lay the foundation for this kind of project success. The best practices of project estimation and management can help you develop a good estimate. Such an estimate is complete and accurate, captures and balances risks, has committed team and individual ownership, and takes into account dependencies and critical path. Such an estimate gives executives and the project management team options that allow them to balance competing risks. Working together, through smart trade-offs in the context of a good estimate, you can guide your project to success.

