# Test Release Processes

**Rex Black:** President and Principal Consultant
RBCS, Inc., Bulverde, TX

**Key Words:** Testing, critical test processes, test management, configuration management, build management, release management, source code control, software development projects, software maintenance projects, test automation, manual testing.

**Audience:** Test managers, test leads, project managers, test engineers, development managers, release managers, source code control engineers, build managers, configuration managers.

## Introduction

In the last issue, I introduced this series of articles on critical software testing processes. For our first article, let's look at the processes associated with an essential precondition for any test activities: creation, delivery, and installation of a software release into the test environment. Without software to test, there can be no software testing, but, as obvious as this is, many software development organizations do a poor job of managing test release processes. Often, the role of building a test release devolves into the test organization, which is generally ill-prepared to handle the tasks, and the processes are defined reactively—to put out fires—not proactively—to handle key roles optimally.

If you are a test engineer or test manager in this situation, you've probably been telling yourself, with every release, "There's gotta be a better way." Guess what? There is. This article will provide some suggestions on how to improve the processes to simplify your life as a test manager, reduce chaos and miscommunication, and allow you to deliver better test results.

## Definitions

To avoid using the same phrases for different concepts—and different phrases for the same concepts—let's start by agreeing on terminology. If your company uses different nomenclature, that's fine. There is no universally-accepted test glossary yet, so don't worry that your company is "using the wrong words," but do worry if you're doing the wrong things.

- Build, release, or test release: An installable software item transmitted to the test group for testing.
- Repository, library, or source code control system: The system—preferably a commercial or shareware software tool like Visual Source Safe™, ClearQuest™, CMS, or SCCS—that stores the source code, online help, HTML documents, audio files, and other files that, upon compilation and assembly, make up the installable software item.

- Build manager, source code control engineer, or release manager: The person responsible for creating the installable software item from source code.
- Build ID, release name, version number, or version. Some sequence of numbers and/or characters that uniquely identifies the release.
- Test cycle. A set of tests run against a given build.
- Test pass or test suite pass. A complete run of all test suites, either during one cycle or across multiple cycles.

Finally, to define the scope of this article, let me point out that I'm leaving aside the issue of how to decide when to release the software to customers or users. This is also a critical test process—and likely topic for a future article—but not the subject of this article.

## The Test Release Process

As a starting point, let's outline an idealized test release process:

1. Select the content (bug fixes, new features, and documentation) for a particular release.
2. Implement the changes required for the bug fixes and new features.
3. Fetch the source files from the repository; compile, link, and otherwise assemble the build; and, mark (in the build and in the repository) the build with a version number.
4. Smoke test the build. If the tests pass, continue with the next step; if the tests fail, figure out what went wrong, fix the problem, and return to the previous step. (See Kurt Wagner and Andy Roth's article, "Lessons Learned from Smoke Testing," in Volume 1, Issue 1, for more details on smoke testing, which is a critical software testing process in its own right.)
5. Create an installable media image of the build; package it appropriately; and, deliver it to the person responsible for installing it in the test lab.
6. Install the build in the test lab.
7. Smoke test the build in the lab environment. If the tests pass, begin the test cycle; if the tests fail, uninstall the build, resume the old test cycle, and return the build to the development team to start over at the first step.

Your process may differ significantly from this, in terms of having more or fewer steps. However, this is based on a process that I as a Test Manager saw work extremely well at a previous client. Such a process is both practical and productive.

## Test Release Process Quality Indicators

I asserted above that the process outlined was excellent. Why? Because it met what I consider to be key quality indicators of a good test release process. Based on my experience and suggestions from the tester and the test manager communities, I propose the following quality indicators.

***Q.I.1*** Predictable and timely releases.

***Q.I.2*** Simple installation/update/apply procedures.

***Q.I.3*** Simple uninstall/removal/unapply procedures.

***Q.I.4*** Testing the real process.

***Q.I.5*** Consistent version naming.

***Q.I.6*** Simple name interrogation.

***Q.I.7*** Documented content.

***Q.I.8*** Intelligent content selection.

***Q.I.9*** Coordination of data and code.

In the following sections, I'll discuss why each of these indicators matters, using case studies and comments from test professionals. (Below, I have grouped some related quality indicators together, for ease of discussion and to reduce repetition, but referred to this list by quality indicator number as above.)

## *Q.I.1:* Release Timing

Accepting a new release into testing has a number of implications for the test operation. The test manager must communicate to the test team what tests they are to run against this build. The release must be installed, and any necessary data files loaded, databases updated or changed, or dependent system configuration modifications made. In the case of client-server and other multiplatform test settings, installing a new release may involve installation on many computers, and the order of this installation can be significant. The test team must ensure that the build is ready for testing. If the build is not ready for testing, they must either make it ready—e.g., patching the software on the fly, fixing a heretofore undiscovered test environment configuration problem, etc.—or, often having failed in the attempt to ready the build, they must remove it from the system. Once the build is properly installed and the test environment ready, the test team often is mandated to proceed to confirmation testing—verifying that those bugs reportedly fixed in the release are indeed resolved—prior to resuming planned test activities. In some cases, a complete regression test—rerunning all previously-run tests or retesting all previously tested conditions—is required as a first priority. In situations where multiple test projects are active in the test lab at one time, the arrival of a build for an urgent project can trump testing of other projects, requiring some significant and risky context-switching on the part of all or some of the test team.

While not all test groups experience such extensive effects, many do. Under these circumstances, installing a new release is not a decision to take lightly. Since it interrupts and changes the planned test activities, most test managers find it undesirable to have releases show up at unpredictable times. Likewise, because of the overhead associated with installing a new build, having them arrive too frequently can seriously affect forward progress. However, because testing against a "stale" build can result in the reporting of already-resolved bugs, it is also possible to have build show up too rarely; the test manager may want to accept a build prior to running all the test cases.

Peggy Fouts, a Senior QA Architect at Compuware, agrees that release timing is a critical quality indicator. She mentions the difficulty of changing bad habits in this regard, but offers the following cautionary tale as motivation to do so. "Currently I am working for an organization where…the users have become accustomed to getting what they ask for as soon as they request it. Although this organization has set up some good structured testing and QA practices, they will not be able to move forward since random unscheduled releases keep the testing group from test case maintenance and process improvement activities. This is typical in my twelve-plus years of experience as a consultant."

The urgent "stop everything and jump on this test release" syndrome is a common one for test managers. Darcy Martin, Lead Test Engineer for Lexis-Nexis, shared the following anecdote. "I was just in a meeting this morning with a development manager. Our discussion was surrounding the inability of test to plan appropriately for emergency patch fix releases to client based software. In many cases this is unavoidable; a major customer reports a impact level 1 problem [so] we jump right on it and at times want to release the same day! We've decided to schedule monthly or bimonthly patch releases tentatively into the schedule for future planning in an attempt to alleviate (not remove) test resource planning conflicts." There's no real solution when two projects want to use the same resources at the same time, though plans can be put into place to reduce the impact.

Gary Dawson, Senior QA Engineer for MediaWay, pointed out the schedule-delaying effects of too-frequent builds. "I have seen scads of time lost on projects because gun-slinging Development VPs or Directors…decree that going fast is of the essence, and therefore there will be multiple builds [going into test] weekly…The initial result is that QA spends 75% (yes, I can document this) of its time just trying to install [the system under test] for the first 3 or 4 weeks of a project." Think of that for a minute: If you have eight person-weeks of testing planned for the first four weeks of your test project, you can complete only two person-weeks of it. Since test projects are often planned for two months or less, the test team is seriously behind schedule at the end of the first month, and the project team has only about a quarter the planned tests run. Unless most of your test cases are fluff, the results of which don't matter, then serious bugs remain undiscovered with half the planned testing time gone.

Barbara Ruf, Quality Director for Netperceptions, agrees with Gary and Peggy's assessment of this as a critical but difficult quality indicator for release management. "I have major problems with predictability and timing of releases. I have a tendency to reward ill-behaved projects that happen to be critical (such as in terms of revenue, customers, etc.) and accidentally punish well-run projects that get hit by the bad project train. This is something we are working on. We are trying to assign testing resources in advance by the calendar, and then tell other projects that spontaneously show up in timeframes that are already scheduled that they are out of luck. This can be dangerous, as then they will just ad hoc [test] something and release it anyway. So we work on being a sufficient value add that other groups regard it as unthinkable to avoid going through the formal test event."

While every test project is different, I like to have builds show up during the System Test Phase once every week. A good process might involve the delivery of a build from release management every Monday morning, with a target for completed installation and commencement of testing by Monday afternoon. As the build is being installed, the test manager can plan the upcoming test cycle based on the release content (see below), and summarize the results of the previous cycle for a test status meeting in the afternoon while the next cycle of testing gets underway. If possible, I avoid interrupting a test project currently in progress with testing for another test project.

As an example, on one project, the development team started buttoning up their development activities for the agreed-upon release content on Friday, and by Saturday night the code was ready. The release manager would come in Sunday morning, and we received a build every Sunday afternoon. By the time the weekend shift ended on Sunday evening, we had a new build installed and sanity tested, ready for testing on Monday morning. This was a very effective process for test releases.

## *Q.I.2, Q.I.3, & Q.I.4:* Install and Uninstall Processes, Including Real-Life Testing

The complexity of the install process is often a function of the system under test. A Windows application typically has a straightforward installation process, while complex IT environments with multiple servers, sometimes distributed across a wide-area network, can take person-hours of time to install. In all cases, though, the process should be made as simple as possible. Simpler processes are less prone to error, and therefore more likely to result in a testable configuration sooner. Also, the simpler the process is, the less likely it is to involve multiple people; the "too many cooks" effect always increases the difficulty, duration, and likelihood of failure for any activity. If the process is simple enough, the test team itself should do the installation.

In addition to simplicity, installing a test release into the test lab should kill two birds with one stone. At some point, the test team needs to verify the installation process itself. In the case of shrink-wrap software, this often involves testing the installation on a variety of system configurations. For IT software, the environment may not change, but variations like the order of executable installation or changes to the database, the use of patch, update, or fresh install processes, and the like creates multiple test conditions. So, the installation process should cover all these tests, thereby sparing the test team an extensive and redundant installation test effort.

The uninstall process is, in the best case, a variation of the install process. For Windows applications, for example, the Control Panel allows access to both functions, and InstallShield™ format packages support removal of files and registry changes as a menu selection. For Solaris, the package management utilities *pkgadd* and *pkgrm* provide like functions. In some situations, using a utility like Ghost™ can allow the test team to preserve clean images of systems to which they may apply any level of release. However, in some very complex IT environments backing out a change may be much more difficult than installing one, unless configuration management tools exist that can reliably undo any sequence of changes to a system. Nevertheless, the prudent test manager will lobby hard for an uninstall facility. Without such capabilities, a fatally flawed build that destroys the test systems can result in literally a week or more of downtime while the entire test environment is completely reinstalled, including the operating system, supporting software, and a previous version of the software under test.

On a Unix network operating system project long ago, I saw the install and uninstall process done very well. The hardware environment was a heterogeneous mainframe/PS-2 network with Ethernet and token-ring infrastructure. An integrated program, launched from one server, could: 1) install a fresh version of the operating system onto formatted hard disks on one or more servers or workstations; 2) update existing servers or workstations with updates or patches; or, 3) back-out updates and patches. During System Test, we varied the process used every week to cover the possible combinations, providing both a clean release process into test for the subsequent week, and, by the end of the System Test Phase, systematic testing of the install and uninstall facilities in the operating system.

## *Q.I.5 & Q.I.6:* Release Naming and Interrogation

In a well-run testing process, as a tester runs tests against a particular release in the test lab, she will, eventually, find a bug. She will then log the bug in a bug tracking system, usually some sort of database. Generally, these bug tracking systems include some sort of release name field, so the tester needs a way of identifying the offending release. Without this information, it becomes difficult for developers to locate the root cause among the moving parts in the code base, especially if the failure's symptom changes somewhat from one version to the next. In addition, managers may have trouble generating per-release bug-find-rate metrics.

For these reasons, a consistent naming convention is needed for any software released to test. The naming convention need not be meaningful. You could, conceivably, name releases Fred, Zedediah, and Hemamalini. However, I like to see sequence and simplicity in release naming schemes.

To start with sequence, we could use a naming pattern like A, B, C, …, or 0001, 0002, 0003… The former approach is rather weak, though, as it will either "roll over" or go to AA, AB, AC, …, after reaching Z. In one successful release management regime I have seen, my client used the four-digit number approach. Builds occurred nightly, each with an incremented build number, and were smoke tested and used by the developers as a base for further development. Once a week, the release manager delivered the nightly build to the test team. So, we received version IDs about seven numbers apart; e.g., 0103 for one test cycle, 0110 for the next test cycle, 0117 for the cycle after that, and so forth. I say "about seven numbers apart" because sometimes developers broke the build, and more than one build occurred on a given day.

This brings up a point about a naming pattern that sounds good on the surface but often doesn't work well. Danny Faught, Senior Consultant for Reliable Software Technologies, wrote to tell me that, in his experience, "the natural tendency is to use the date, so the issue bears some extra attention to make sure the labels are unique. On my last project, they used the date concatenated with the time (down to minutes), which seemed to be sufficient except when the developers patched the system without bumping the release label or checking in the changes." This is the path to bug reporting chaos and irreproducible failures, because two builds with identical release numbers are not the same and may exhibit different behaviors.

In terms of simplicity, note that the tester need not know down to the individual file level what the release names mean, as long as the developers can figure that out, usually with the help of their source code control system. I once managed testing for a system that had seven major components, and each component had a five-part release identifier. In other words, it took thirty-five numbers to describe the release name. Each of these numbers could be one, two, three or more digits and/or letters. Such an approach makes it difficult for testers to keep straight which release they're testing. It is best to have a single number, which the repository tool can then break down into individual component release IDs, and, if necessary, specific source file changes.

Now, a logical naming schema is a necessary condition for crisp release/bug information capture, but not a sufficient condition. In order to enter the release name against which a bug was found into the bug reporting system, the tester must know it or be able to find it. Ideally, one would like to interrogate the system; i.e., ask it a question that results in it giving its own name. For example, most Windows applications include an About screen, accessible through the Help pull-down menu, that will provide the version ID. As an example of a less user-friendly but still workable approach, in some Unix applications, coding standa rds call for inclusion of a release ID string in the executable that a tester can find using the *strings* utility. For either approach to work, of course, the build process must include either automated or manual processes for changing this identifier from one version to the next.

In some cases, one can't query the system itself, but rather the system that updates it. Let me illustrate this by example. I tested a client/server system once where the software that ran on the clients was "pushed" from a single server. This server had a database that included the client device identifier and the release ID for the set of bits last successfully updated to the device. By querying the server, I knew which version of software any given client was running. To revisit the install and uninstall topic for just a minute, this system also provided a simple command line interface that allowed me to change the release to be pushed to the client the next time it connected, both backward (uninstall) and forward (install), thus providing a nice, clean, consistent interface for install, uninstall, and interrogation activities. This example illustrates that, with some careful forethought, a simple yet powerful utility can handle many of the issues related to release management.

## *Q.I.7, Q.I.8, & Q.I.9:* Content Selection, Coordination, and Documentation

To illustrate the issues associated with test release content management and documentation, consider the following hypothetical situation. A System Test cycle begins against a given release. Over the course of a week of testing against this release, the test team reports thirty problems, each as a distinct bug report. In addition, sales, marketing, or the business sponsors log six enhancement requests; i.e., new features to tune the product for the customer or user base. Collectively, these documents create opportunities for the project team to improve the quality of the product, but the process must be managed carefully to ensure effectiveness and efficiency. Improving the quality of the product using these three dozen data items requires a process that deals with three questions in sequence.

1. How does the project management team select which of these thirty-six potential changes in behavior are candidates for inclusion in the next test release?
2. How does the development team notify the test team of which changes in behavior they believe were implemented through code changes?
3. How does the test team return status on success or failure— behavioral change implemented without associated regression, behavioral change implemented but with an associated regression, or behavioral change not implemented properly—of these code changes to the development?

These are test release content management and documentation questions.

Figure 1 below shows a process that I have seen work to handle these questions. In this process, the bug reports and enhancement requests comprise the inputs to a cross-functional meeting. At this meeting, the project management team, including development, testing, sales and marketing or the business sponsors, technical or customer support, and perhaps operations, will evaluate the costs and benefits of each proposed bug fix and potential enhancement.

Content selection affects both the system under test and the test process for a given cycle. For example, some missing features or bug fixes may take on special importance because they block critical tests. These tests, if delayed until the end of the testing process, may result in a last-minute discovery of show-stopper bugs. In addition, in some development lifecycle models, such as the evolutionary model, release content must be staged so that, at the end of each test cycle, a "shippable" product can be built.

At the end of this meeting, tentative agreement is reached on what changes will and won't be implemented. I say "tentative" for two reasons. First, while senior and executive management generally defers to the project management team to manage the bug and enhancement selection process, they also sometimes have pet features or pet peeves that—for political reasons if none other—must be implemented. Second, the development team can't always complete implementation of every change in time for the next release. Some changes take longer than others; some prove more difficult upon closer inspection than originally planned.

The development team will complete a significant chunk of the proposed changes, then deliver a test release as discussed above. However, as shown in Figure 1, the test team should also receive release notes. These release notes address question two above, which of the proposed changes the development team actually implemented. I have seen this process work well in a variety of ways. In situations where a robust bug and enhancement tracking tool is in place, the entire process can work through this tool. The development team, in the course of preparing a test release, marks as "ready for testing" the bug reports and enhancement requests associated with the changes in the release. The test manager, upon receiving the release, can run a report from this tool listing all those items in a "ready for testing" state. On some projects in the Unix and Web application worlds, I have seen development teams provide text-file release notes as part of the tar archive containing the release. These documents, often called "README" files, may refer to bug IDs, or just list the changes. Clearly, the more automation and tracing back to a managed list of changes the better, but the lack of such support tools shouldn't obscure the need to implement *something*. Indeed, in cases where the automated tools are missing, the manually produced release notes become all the more critical.

Closing the loop on these changes is the last step in this iterative process. With an automated bug and enhancement tracking system, this usually entails marking reports associated with the successful changes as "closed", indicating the problems found with the unsuccessful changes in their reports, and writing new reports for regressions or new bugs found. (More details on confirmation and regression testing and results reporting will wait until I tackle the test execution process in a subsequent column.) The closed reports are cause for celebration, while the reopened and new reports are inputs to the next round of content management and documentation activity; i.e., back to the top of Figure 1. This cycle continues until the project management team decides that the phase exit or release criteria are met. In the case of System Test, this is often a decision that the risk of delaying shipment for another fix/enhance/test period exceeds the risk involved of shipping with the known bugs and without any pending requested enhancements.

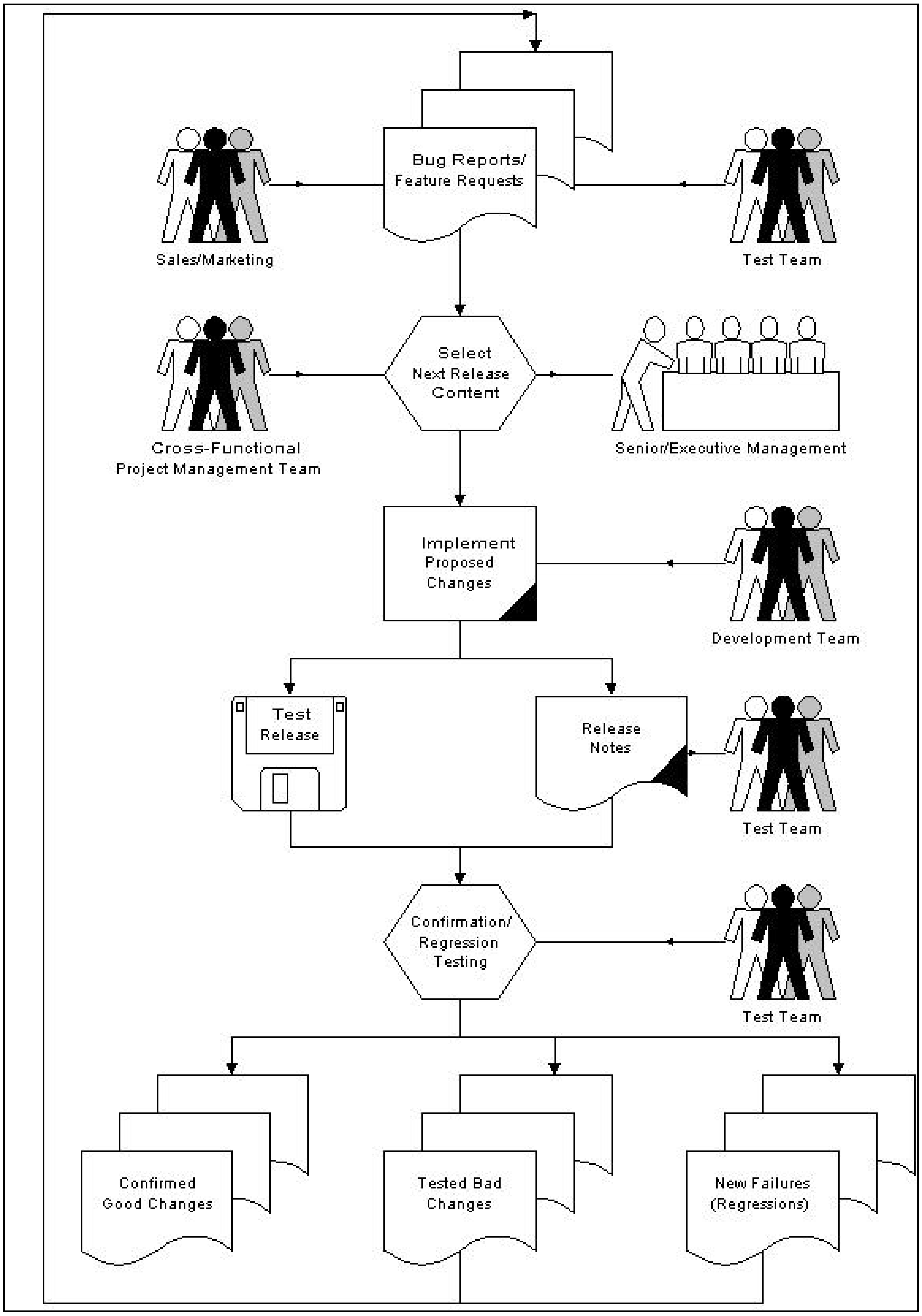
Bug Reports/ Feature Requests
Sales/Marketing
Test Team
Select Next Release Content
Cross-Functional Project Management Team
Senior/Executive Management
Implement Proposed Changes
Development Team
Test Release
Release Notes
Test Team
Confirmation/ Regression Testing
Test Team
Confirmed Good Changes
Tested Bad Changes
New Failures (Regressions)

**Figure 1: Managing and Documenting Test Release Content**

Tim Dyes, Software Engineer at GambroBCT, emphasized the importance of making intelligent decisions about release content. "Also key is the process to decide and control what goes into a release and what is deferred. It's helpful to have weekly reviews…to decide what goes in and what doesn't." Tim also recommends using these review meetings to assess progress towards test completion, offering some solid advice. "Release criteria and where you currently stand regarding [them] should be communicated frequently to the whole team."

Harvey Deutsch, Test Manager at Netpliance, Inc., an Internet appliance services company, agrees with Tim. He wrote to tell me, "Two of the most critical components of a high quality, on time release are: 1) Don't develop what you are not going to test, and, 2) Don't test what you are not going to release. In other words, cut early and often." Harvey's two points should probably be printed in large letters, framed, and hung prominently in the meeting room where the discussions about release content are held.

I should mention two facts in closing on this topic. First, what I discussed above is a test-centric view of what is often referred to as *change management*. To keep this article focused, I chose not to go off on a tangent about that extensive topic. However, it is an important topic for test managers, in much the same way that release management is: The test manager is usually on the receiving end of both processes, and is helped or hurt depending on whether the processes are done well or poorly. I will write a future article on this topic in this space, but, in the meantime, I recommend that curious—or concerned—test professionals continue learning about change management by reading Steve McConnell's excellent book, *Software Project Survival Guide*, especially Chapter Six, or by consulting his company's web site, www.construx.com.

Second, it's easy to overlook, but in situations where data and/or metadata have significant effects on system operation, don't forget to manage and document this as carefully as executable content. For example, I once worked on a project where one development team claimed their component of the system was feature complete, but they kept changing the schema of the master table that held the user information from one release to the next. This caused as much impact to testing as adding or removing features, since each field was associated with user interface changes and modifications to the workflows and transactions that affected the call center agents. Randall Becker, CEO of Nexbridge, Inc., mentioned this as a particular release management concern—and blind spot—for many big data-centric IT projects.

## Who Owns the Test Release Processes?

Having outlined a process and discussed each piece in detail, let's examine the process again, this time from the point of view of task ownership. In Table 1 below, I list the tasks shown earlier in this article, this time with associated owners. In some organizations, steps three through six live in a sort of no-man's-land; people seem to think that software magically migrates from the development team to the test team with no further effort expended. Because the test team needs this migration to occur to begin testing, of course, the result is predictable: the steps devolve upon the test team, which, as I noted in the introduction, is ill-prepared to take many of them on.

When I have seen the test release process work smoothly, steps three through six have the owners I propose in Table 1. The creation of the build, the initial smoke test of it, and the delivery of that build, in a customer-like format and via a customer-like process, belong to a special project team member. This person goes by many names; I have heard "build manager," "release manager," "source code control engineer," "source librarian," and even, sardonically, "build dude." (I like "build dude"—or "dudette"—because it highlights the poorly understood role played by this individual, and humorously reminds people not to underestimate his or her importance.)

To whom should the build dudette report? I'm not sure I care. I have seen the testing team report into a development services group that also included release management (and user documentation). I have also seen the release manager as a separate member of the development team. Both approaches worked.

Baring the presence of a dedicated build dudette on your project team, the next-most-logical place for these roles to live is in the development team. After all, in each release the development manager is proposing that the test team accept her team's work as ready for testing. Shouldn't she own the tasks associated with delivering that build, *demonstrably* ready for testing, to the test team?

Note, though, that I propose the test team own not just the initial smoke test prior to starting the test cycle, but also the installation. Why? Because, as I have mentioned earlier, I believe that the test team should independently evaluate the installation and uninstallation processes as the end-user, operations group, or customer will experience them. This is a test activity, because we're looking for bugs. In some complex settings, this may require special expertise in the test team. In one case, where server-side software was installed on Solaris supercomputers, my test team audited the install processes by reading the documentation and observing the steps performed by the network operations team. However, the ideal case is to have a tester on the team competent to test the installation and uninstallation processes armed with nothing other than the same checklists or documentation the end-user or customer will receive.

| Step | Tasks | Owner (Typical / Proposed) |
|---|---|---|
| 1. | Select the content. | Cross-functional project management team |
| 2. | Implement the changes. | Development team |
| 3. | Make the build. | Unclear<br>Release manager |
| 4. | Smoke test the build. | Unclear<br>Release manager |
| 5. | Package and deliver the build. | Unclear<br>Release manager |
| 6. | Install the build in the test lab. | Unclear<br>Test team |
| 7. | Smoke test the build in the test lab. | Test team. |

**Table 1: Test Release Process Tasks and Owners**

In addition to the inappropriateness of some of the test release tasks falling into the test team, Gary noted that having test assume ownership for steps three through five can adversely affect testing objectivity. “[It] is my experience that when release control is in QA the QA endeavor at that company is usually somewhat less effective than one would prefer. I believe that this is because it allows developers to relieve themselves of the responsibility to produce good code that *fits* into the code base. When developers know they have to make it fit in right they tend to be more judicious in how they check code in. I’ve always seen QA as ‘you provide a product and I’ll test it.’ When I get into the build process I then invariably find myself [handling the build process acceptably] but [losing] any desire to test [the end result]. One company I worked for had development doing programming, release handled by marketing, and QA reporting directly to the CEO. That was probably the best setup I've seen…”

## Moving Forward from Where You Are

So, having read this article, you hopefully have formed some ideas on changes you’d like to see in your company’s test release processes. However, Peggy, Darcy, and Barbara’s comments point out a general problem for the test manager. Instituting processes changes of any sort—especially those that require the cooperation of people outside the test area—is hard. When those changes can affect customer-delivery dates, the challenges are harder still. Barbara’s remarks highlight the ironic truth that one of the very factors that makes predictable and timely test releases important—the need to balance competing demands for scarce test resources—are undermined by the urgency of some release schedules.

This is usually a business-priority balancing act. As the test manager, you are there to manage business priorities, in your case by assessing risks to product quality. The business may choose to make snap decisions about what releases go into the test operation when, and that’s okay, provided that you have done a good job of explaining the costs. Don’t push back on taking an out-of-cycle release because it makes your life difficult; push back because it introduces inefficiencies—which you can document, right?—into your process that affect your ability to provide a desired assessment of quality at a given cost. In other words, you can’t add value through testing if all your resources get eaten up task-switching and installing (unready) builds.

Barbara’s point about making test a significant value-add to the process such that test gains credibility also came up in a discussion with Danny. In terms of selling test release processes improvements to peer managers, he wrote, “You have to extol the benefits to the whole organization of getting this additional installation support; e.g., the time (and thus money) that could be saved by having a faster installation process.” I couldn’t agree more. I like to phrase my lobbying for intelligent naming and trivial interrogation processes in terms of providing better service to development. I say things like, “It’s important for a professional test group to write world class bug reports, so we want to tell you exactly what releases have particular failures. To do this in a reliable fashion, we need your help creating good processes for naming and interrogation.” Likewise, with confirmation testing, I like to say that we can do a better, quicker job of confirming bugs as closed if we know exactly what release they arrived in.

In terms of the install and uninstall process I recommend, these methods are wildly divergent from what happens in the typical development setting. Developers often install releases onto their systems by direct migration of files into target directories and hands-on hacking of configuration files and database data and metadata. Instructions from a developer on how to install a build may include advice on how to edit the Windows registry or add an entry to the /etc/passwd file. Therefore, the install process will not naturally conform to the requirements I laid out above; you will often need to explain the need for process and structure to the development engineers, the development managers, and the release manager.

Sometimes, a less subtle and more hand-on approach is needed to motivate process change. Gary told me that sometimes "the only way to solve [build installation process problems] is to drag developers over to…walk through installs enough that they feel the pain too. They will invariably correct install problems when they realize that much of the test plan is being forfeited by time sucked into the install black hole." Gary's advice assumes that the development team owns the build delivery process; in your situation, the release manager may be the lucky girl or guy invited to participate in working through a broken installation process. Like me, Gary feels that the install process is getting more difficult—and critical. "In today's climate of client/server/web installs [with] multiple gotchas…the install process becomes even more important to saving time for projects."

Lastly, notice that you don't have to go it alone in advocating these changes, since, as Danny pointed out to me, "often…the changes you're asking for will also be required by your customer support team." As an example, he mentioned that the interrogation processes are "useful features to the customer." He concluded, "So get some allies to lobby with you." Good advice!

## And Don't Forget…

Finally, as Ross Collard, Principal Consultant of Collard and Company and the author of an upcoming multivolume opus on testing and quality assurance, reminded me, one shouldn't forget about tool support. Release management—whether into the test lab or to the customer—is something one can't do properly without a source code control system. A decent tool will support branching and merging code streams, name the versions, prevent conflicting modifications of the same file, and provide some interesting metrics on module changes, too.

One of my clients purchased a hardware item for integration into their product. This item had on-board firmware. As part of trying to learn about the version names of this firmware—for entry into the bug tracking system—I asked about the source repository. It turned out that this vendor used a directory on a Windows NT™ system and the Windows file Explorer to store and manage the firmware source. This is a bad idea for a number of reasons:

- No control over simultaneous modifications, unauthorized modifications, or modification dates.
- No way to reliably change version names when the code changes.
- No support for merging multiple changes back into a single source again.
- No support for backups.

There are just too many tools available commercially—and as freeware or shareware—to risk precious source code this way.

Also widely available through such channels are problem/enhancement tracking and management systems. The aspects of test release management associated with managing and documenting the release content are much easier to handle with automated tools, and much harder to do without. Creating your own tools might adapt better to your workflows, but don't underestimate the complexity of doing so.

The availability of tools for installing, uninstalling, and interrogating the software will depend considerably on your target platform. Windows and Unix platforms, including the Linux platforms, have a fairly broad assortment of such tools available, while less widely used platforms may not. Note also, though, that the right set of tools for this job depends not only on the target platform, but also the intended distribution process. While you want to keep these tasks simple, you *must* at some point test the user or customer install and uninstall mechanisms. Beware of falling into the trap of using some simple approach that is not the same as the field process.

Ross also brought up the importance of disciplined process in addition to tools, writing that "in many organizations, there is not an adequate assurance that the version of the system being released is the same as the one that was tested. The versions *could* be the same, but we don't know for sure. Either the programmers can have access to the system after the testing is done for last-minute 'minor' adjustments, or the version control/configuration management system is not powerful enough to detect all the changes, or the configuration of the test environment is not identical to that of the live operational environment." The latter case especially arises when development teams release software to the test organization that "works on their systems" but no one knows how to replicate the configuration of the development environment. This brings us to configuration management, another critical issues for testing. We'll examine this topic in a future article.

## Acknowledgements

I'd like to thank the following people for their insightful contributions to the development of this article: Randall Becker, Ross Collard, Gary Dawson, Harvey Deutsch, Tim Dyes, Danny Faught, Peggy Fouts, Darcy Martin, and Barbara Ruf.

## Author Biography

A Contributing Editor for the *Journal for Software Testing Professionals*, Rex Black (Rex_Black@RexBlackConsulting.com) is the President and Principal Consultant of Rex Black Consulting Services, Inc. (http://www.RexBlackConsulting.com), an international software and hardware testing and quality assurance consultancy. He and his consulting associates help their clients with implementation, knowledge transfer, and staffing for testing and quality assurance projects. His book, *Managing the Testing Process*, was published in June, 1999, by Microsoft Press.