RBCS
TIME TESTED.
TESTING IMPROVED.
www.RBCS-US.com

Component Outsourcing, Quality Risks, and Testing: Factors and Strategies for Project Managers

More and more projects involve more integration of custom developed or commercial-off-the-shelf (COTS) components, rather than in-house development or enhancement of software. In effect, these two approaches constitute direct or indirect outsourcing of some or all of the development work for a system, respectively.

While some project managers see such outsourcing of development as *reducing* the overall risk, each integrated component can bring with it significantly *increased* risks to system quality. In this article, I'll explain the factors that lead to these risks, and then strategies you can use to manage them.

I'll illustrate the factors and the strategies with a hypothetical project. In this project, assume you are the project manager for a bank that is creating a Web application that allows homeowners to apply for a home equity loan on the Internet. You have two component vendors. You buy a COTS database management system from one vendor. You hire an outsourced custom development organization to develop the Web pages, the business logic on the servers, and the database schemas and commands to manage the data. Let's see how you can recognize the factors that create quality risks, and the strategies you can use to manage those risks.

Quality Risk Factors in Integration

Figure 1 shows four factors that lead to increased quality risk for a system. Working clockwise from the upper left, let's take a look at each factor that can increase risk to system quality.

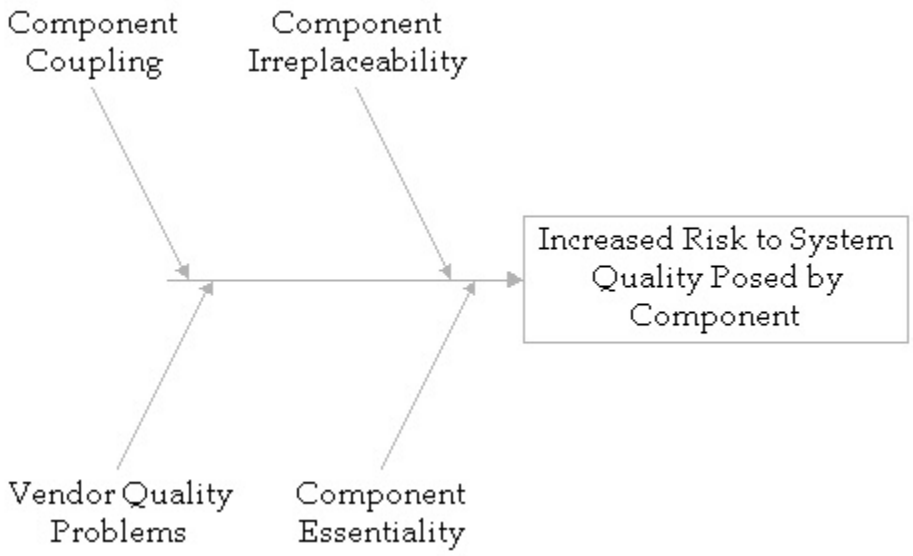

Figure 1: Sources of risk in system integration

One factor that increases quality risk is coupling, which creates a strong interaction with the system—or consequence to the system—when the component fails. For example, suppose the customer table on the Web application database becomes locked and inaccessible under normal load. In such a case, most of the other components of the system, being unable to access customer information, will also fail. The database is strongly coupled to the rest of the system.

Another factor that increases risks is irreplaceability, when there are few similar components available or the replacement is costly or requires a long-lead time. If such a component creates quality problems for your system, you are stuck with them. For example, the database package you choose might be replaceable, provided that you don't do anything non-standard with it. However, the development organization will want to be paid for the custom-developed Web application, and, should you choose to try to replace it, off-the-shelf products might not exist.

Yet another factor that increases risks is essentiality, where some key feature or features of the system will be unavailable if the component does not work properly. For example, suppose you planned to include a pop-up loan planner on the first page of your application to allow customers to evaluate some payment scenarios. Should that component not work, you can still deliver the major features of your application. A pop-up loan planner is not essential to your system. However, if the subsystem that accesses a credit bureau to check customer credit scores does not work, you cannot process loan applications, since checking credit scores is essential.

The final factor that increases risks is vendor quality problems, especially if accompanied by slow turnaround on bug fixes when you report problems. If there is a high likelihood of the vendor sending you a bad component, the level of risk to the quality of the entire system is higher. For example, if you buy a commercial database from a reputable, established vendor, or if you select a custom development organization with a proven track record, then you will probably have fewer problems. If you use a new open-source database that has never been used in commercial applications before, or if you use a newly-open custom development organization, then you will probably have more problems, particularly is there is no or poor component technical support.

So, you can see how these factors would exist and affect a typical data center application. For a weapons system where defense contractors intend to develop software to run on COTS platforms, the situation is similar, though the replaceability and vendor quality problems could be exacerbated by limited choices for components and vendors. How can you mitigate these risks? I have seen and used four typical strategies.

Trust Your Vendors

One strategy is simply to trust the vendor's component quality and testing, and assume they will deliver a sufficiently-good, more-or-less working component to you. This approach may sound silly and naïve, expressed in such words. However, project teams do this all the time. My suggestion is, if you choose to do so, do so with your eyes open. Understand the risks you are accepting. Allocate schedule time and finances as a contingency for poor component quality. The more coupled, essential, and irreplaceable the component, the greater the impact of such a situation.

To continue with our example, suppose that you plan to trust both the custom development organization and the database vendor. You could make such a decision rationally by checking the custom development organization's references, assuming they can provide references for customers who used them for projects that are very similar to yours in design and scale. The same is true for the database vendor, though you might have to do your own research if their sales and marketing staff cannot or will not provide references.

Let me mention that relying solely on an acceptance test is practically the same as trusting your partners in the custom development situation. For the COTS database, you could run an acceptance test at the beginning of the project for the database, using simple models to evaluate database performance, reliability, and data quality under your intended load conditions. However, for the custom-developed component, you'll have to wait until you receive the component until you can acceptance test it. If the component fails the acceptance test, what

options do you have? Even if the contract stipulates that you don't have to pay under these circumstances, there is a good chance of a lawsuit, and you also have the actual (and opportunity) costs associated with having to start over with a new custom development organization.

Manage Your Vendors

Another strategy is to integrate, track, and manage the vendor testing of their component as part of an overall, distributed test effort for the system. This involves up-front planning, along with having sufficient clout with the vendor to insist that they consider their test teams and test efforts subordinate to and contained within yours.

To continue with our hypothetical project, imagine that you are working at a large bank, and that the custom development organization is a small firm. They will probably be motivated to get and retain your business. They will be especially flexible if they think that you have particularly good testing processes and that they can learn something from you. In exchange for the effort you expend managing their testing, you will have early warning should quality problems emerge, and therefore you will have more options to deal with such an outcome.

Conversely, though, if you are buying the database from a large COTS vendor, they probably see your business as a small part of their larger product sales picture. They have their own test processes, product roadmap, and target release dates. It is highly unlikely that they will be receptive to offers—much less insistence—that you manage their testing operation.

Even with smaller COTS vendors, when they are selling a COTS component, they will want to sell you want they are offering. They will likely be averse to the possibility of an open-ended situation where you might redefine the component's requirements through expansive testing and ambiguous or evolving pass/fail criteria for the tests. I have seen more than one COTS vendor get burned by customers when they allowed this to happen.

Smart COTS vendors (large or small) would probably insist that this management of their testing, and any resulting bug fixes and change requests, be considered a customization of their component subject to time-and-materials billing. The only likely exceptions to such a condition would arise when the COTS vendor saw a strong possibility that working with you to fix problems and change the product would benefit their current or future customers sufficiently to justify the risks they would be taking.

Fix Your Vendor

Another option is to fix the component vendor testing or quality problems. In other words, you go into the situation expecting to either revamp the vendor's test and quality processes or build new test and quality processes for them from scratch. Both sides must expect that substantial effort, including product modifications, will result. Once again, a key assumption is that you have the clout to insist that you be allowed to go in and straighten out what's broken in their test and quality processes.

While this sounds daunting, on one project, my client hired me to do exactly that for a vendor. It worked out nicely. The vendor was paid for their part of the work, including the modifications. My client felt that the vendor brought enough technical innovation and capability to the project that managing the quality and testing problems for the vendor was worthwhile. With expectations aligned from the start, both sides were happy.

Going back to our hypothetical example, suppose you assess the outsource development organization before the project, and find their testing and quality processes lacking. They accept your assessment. You offer to help them fix the issues that were identified, and they accept that offer. If your assessment identified the major problems, *and* if you and the vendor can resolve those problems with the scope, budget, and schedule for the project, *and* if continuing to work with that vendor makes sense for other reasons, then this can succeed.

However, for the database vendor, it's difficult to imagine they would accede to the request for an assessment of their testing to begin with, not to mention allowing you to come in and implement changes to it. In fact, for any COTS vendor, the very fact that they might agree to such a request should set off alarm bells in your mind. You would have to ask yourself, do they actually have a COTS product to sell or are we dealing with a prototype masquerading as a product?

Test Your Vendor's Component

A final option, especially if you find yourself confronted by proof of incompetent testing by the vendor, is to disregard their testing, assume the component is coming to you untested, and retest the component. You'll have to allocate time and effort for this, and deal with the fact that the vendor will likely push to have every bug report you submit reclassified as a change request except in the most egregious cases. You also have to ask yourself if the vendor might decide, at some point, to cut their losses and disengage from the project. You'll want to make sure you have contingency plans in place should that happen.

I've had to do this for clients on system testing projects, notably on one project when a vendor sold my client a mail server component that was seriously buggy.

We became aware of the problems by a series of misadventures where promised deliverables continued to show up late, and even then with substantive bugs as well as fit-and-finish problems that gradually eroded our confidence in them. Eventually, the component did work and was included in the system, but the entire process took a couple months, not the one-week deliver-and-integrate that was in the project plan. Fortunately, slack elsewhere in the schedule prevented this from becoming a project-endangering episode.

Returning once again to our example, suppose that you become aware, through the delivery of poor-quality early prototypes from the custom development organization, that serious quality problems exist. You can no longer trust their testing. There's not much point in managing a test process that is clearly broken. There's no time remaining in your schedule to go in and fix their testing process. So, if you intend to stick with this vendor, you'll need to start a serious testing effort to take over where they have failed.

Suppose you become aware of similar problems with the database vendor. You can confront the vendor with the problems. However, if they delivered something to you with the assertion it would work, can you really trust them to resolve the problems now? Would they be likely to let you manage their testing? If you try to do the testing yourself, do you think they will fix the problems you find? Most likely, if the component is not essential, you are best off omitting it, or, if the component is replaceable, you are best off replacing it.

Whether for a COTS component or a custom developed component, these are clearly nasty scenarios, and at some point you'd have to ask yourself how you managed to get into such trouble. If you ran acceptance tests on a COTS component, why did they fail to identify the problems? If you thoroughly vetted your custom developer, why did they prove incompetent? How should your quality risk mitigation strategy for outsourced components change for future projects? These are all good questions, but ones to save for the project retrospective. During the project, the focus must be on achieving the best-possible outcome.

Implications, Considerations, and Success

All of these options have potentially serious political implications. Should problems arise, the vendor is unlikely to accept your assertion that their testing is incompetent or quality unacceptable. They might well attack your credibility. If a senior manager made the choice to use that vendor—and it might been an expensive choice—that person might side with the vendor against your assertion.

So, you'll need to bring data to the discussion about these strategies if the triggering conditions arise during the project. Better yet, if you're dealing with a custom developed component, see if you can influence the contract negotiations

up front to require the vendor to submit their tests and their test results, along with giving you the chance to perform sufficient acceptance testing by your team prior to payment. Build sufficient contingency plans into your schedule, including allowing for replacement of the vendor during the project if things start looking bad. Make sure the vendor understands that you're paying attention to quality, and that payment depends on quality delivery. It's amazing how motivational that can be for vendors!

For COTS components, you'll want to arrange a careful component selection process, including vendor research, talking to references, and acceptance testing as described above using carefully design tests. Identify alternative sources if possible. Consider the possibility and the consequence of omitting the component if it is not essential.

Finally, let me mention one last important consideration. With the risks to system quality managed at the component level, it's still possible to make a serious mistake in the area of testing. Remember that even the best-tested and highest-quality components might not work well in the particular environment in which you intend to use them. So, plan on integration testing and system testing the integrated system yourself.

Integration of COTS and outsourced custom developed software is a smart choice for many organizations. It is a trend that continues to grow as organizations gain experience with it. To ensure success on your next integration project, consider the factors that create quality risk in such scenarios. Select strategies that mitigate those risks. Build risk mitigation and contingency plans into your project plan. If you do these things, and execute the project carefully, with an eye on testing and quality, you can control the risks and reduce the likelihood and impact of component quality problems.

Author Bio

With a quarter-century of software and systems engineering experience, Rex Black is President of RBCS, Inc., a leader in software, hardware, and systems testing. For more than a dozen years, RBCS has provided its worldwide clientele with training, assessment, consulting, and outsourcing services. RBCS has over 100 clients spanning 25 countries on six continents, including Adobe, ASB Bank, Bank One, Computer Associates, Cisco, Comverse, Dell, the US Department of Defense, Hitachi, NDS, and Schlumberger. He has written six books on software testing, including *Managing the Testing Process*, *Critical Testing Processes, Pragmatic Software Testing*, *Software Testing Foundations*, Advanced Software Testing Volume I, and Advanced Software Testing Volume II, which have appeared in English, Japanese, Chinese, Hebrew, and Russian editions. About 50,000 copies of these books have been sold around the world. He has written about 30 articles,

presented hundreds of papers, workshops, and seminars, and given over a twenty keynote speeches at US and international conferences and events. Rex is the President of the International Software Testing Qualifications Board.

Acknowledgement

Portions of this article previously appeared in *Pragmatic Software Testing*.