

Managing COTS Test Efforts, In Three Parts – *It's Like Conducting an Orchestra!*

by: *Virginia Reynolds*
Insight Quality President

Managing COTS Test Efforts, In Three Parts - It's Like Conducting an Orchestra!

Imagine this:

You are a testing professional newly hired by a mid-to-large size company. Any industry. You are responsible for the testing of all of the company's internal corporate systems---HRIS, Finance, ERP, CRM, CLM, Sales Management, and Knowledge Management to name a few. You have just been tasked with providing the testing strategy and test management for a large-scale commercial-off-the-shelf (COTS) implementation. At first you think, "Hmm. A COTS vendor application that has already been developed, tested, boxed up, and released?!? This job is going to be a piece of cake!"

Not so fast, buckaroo.

This is where I found myself earlier in my career. I took a role within the Corporate Systems – IT division of a private mid-size company, where I was responsible for the testing of all corporate applications, which were mostly COTS systems. The corporate systems department focused on implementing new systems, upgrading existing systems, and integrating multiple systems. Not only did I learn how to test the apps---mostly from trial and error, oops---but I also learned that testing COTS systems in a corporate systems integrated environment is quite a bit more complicated than I had originally thought.

Here are some of the lessons I learned as well as a high-level strategy that can be adopted to test almost any COTS system. The strategy I use is to divide the COTS test effort into separate "test tracks" where each test track is managed individually with its own team of test resources and its own project plan. The individual test tracks roll up into your master test effort, including the master test effort project plan. Yes, I said, "project plan." It is my experience that in order to manage test budgets, test timelines, and test resources on a large-scale, complex COTS test effort, it's important to view yourself not only as a tester, but as a Test Project Manager. Your role is really like conducting an orchestra. While each musician learns his part, it is the conductor's responsibility to bring all the musicians together and ensure they play cohesively to produce beautiful music.

I've organized this article into three parts: Part I is about how testing COTS systems is different than other test efforts; Part II focuses on the various test tracks I recommend for all COTS implementations; and Part III provides some keen insights into how to better manage the COTS test effort.

PART I – The Instruments: A COTS System

Before preparing to conduct an orchestra, it is important to gain an understanding about the instruments involved. For our purposes, we're going to discuss one instrument, a generic COTS system.

Lesson #1: With COTS systems, you are not testing the actual product itself as if you worked for the vendor company that developed it.

As we all know, software never arrives 100% defect-free. If you have worked for a software vendor in the testing division, then you also know that there is a balancing act between the quality of the system and time-to-market delivery. With vendor products, many times the time-to-market aspect outweighs the value of a truly vetted application.



Knowing this, you are going to want to retest the entire COTS application. I know I did.

Here's your first lesson: Don't do it.

This is a hard pill to swallow but there are bigger fish to fry, as I'll soon explain. You have to assume that the "vanilla application"---the functionality straight out of the box---generally functions as developed and designed.

Lesson #2: Your company will never use the COTS system's functionality straight out-of-the-box.

Most of the time, someone in the corporate systems organization selects the vendor's product based on a needs analysis (High-Level Requirements) and a gap analysis. The results of the gap analysis identify where the vendor's product meets the organization's needs as well as where the vendor's application does not meet those needs. Where the requirements are easily met, we say that the vendor's "vanilla application" handles the need. Where the requirements are not so easily met, there are generally two options. These compensate for the gap between what the vanilla application *can* do and what the organization *needs* the application to do. If there is a built-in GUI Administration module, it should be used by a Product Administrator to configure the application to meet the business's needs. If there is no GUI module or it is not sufficient, then you will need the vendor to customize the application to meet your needs. This is where the vendor goes into the application's core and/or database and refactors code—and possibly rewrites entire modules---specific to the needs of the organization.

From a testing perspective, the high priority areas of a COTS application are in these two areas, where the product has been configured and/or where the code has been customized. These areas will be the focus of one of the testing tracks that will comprise your overall test strategy: System Testing.

With that said, in order to prioritize the functional (system) testing, it's imperative that the test team has a clear understanding of which requirements are met with the vanilla functionality, which are met with product configuration, and which are met with custom coding.

Lesson #3: It's difficult to identify which requirements were met by which means: core vanilla product, configured product, or custom-coding within the core product.

Most organizations I have worked with---mid-size, not-for-profit, Fortune 10---just do not document this information well. It is, therefore, in the hands of the test team to collaborate with the vendor, the implementation team analysts, the project manager, and anyone else to investigate and acquire this information. If you do not collaborate with all these players, you will not know where the product has the greatest potential for defects and therefore is at greater risk, and you will not know where to prioritize your testing to mitigate that risk. It is extremely important that you set aside time for the test team to research and analyze this information if it is not readily available.

As a side-note, many vendors have their own product test team in place for testing their internal product releases. Once the product is implemented at a client site, however, there is an Implementation Team provided by the vendor that is responsible for all configurations and customizations. More often than not, the Implementation Team does not have a professional tester as part of the team. Frequently, it is this Implementation Team that conducts any testing rather than the vendor's in-house test team. Hence, the vendor's professional testers are not testing the configured and customized code for the client; they are instead testing the plain



vanilla version, which places the customized product at a higher risk. This is just one more reason to ensure your test team identifies areas of higher risk within the product.

PART II – Identifying the Sections of the “Testing” Symphony Orchestra

There are many components that comprise a symphony orchestra: woodwinds, percussion, brass, and string instruments. Some symphonies don't have a brass section and are simply named an “orchestra” (losing the word “symphony”). Some are even smaller, consisting of less than forty players, and are called “chamber orchestras.”

It is important to identify what instruments and sections your testing orchestra will contain from the get-go.

Lesson #4: Most COTS systems are not standalone systems, rather they integrate with one or many other systems.

If you are leading the overall testing strategy of the COTS implementation, then you need to look beyond just functional testing of the application. That's just one section within the orchestra. You also need to know the full scope of the integrations with other systems and plan on testing these as well, both from a functional standpoint and from a data accuracy and correctness point of view.

Unless the organization's IT processes are robust or the organization has implemented a service-oriented architecture (SOA), there are typically numerous methods utilized for moving data between systems. These means of data transition from the COTS system to existing systems will have to be developed in order to integrate the COTS system with other existing applications within the corporate systems environment. You can think of the systems as verses within a musical composition and the backend data integrations as bridges between the verses that allow data to flow from one verse to another. Because these backend data integrations consist of new or updated code (ETLs, APIs, .csv, etc.), the integrations should be identified as high-risk areas for potential defects.

Identifying all of the data integrations between the COTS system and other systems in the early, upfront test strategy is key in planning out the scope and approach of the test effort. This should be the focus of another test track within your overall test effort: Integration Testing.

Lesson #5: It's difficult to identify all of the backend data integrations between the COTS system and existing systems from a Big Picture perspective.

I have found that even the most senior technical architects never have all of the information that they need, from a testing perspective, in a single, tidy document. Many times, I have also found that there is not a single technical owner (or architect) of all integrations. Instead these integrations are allocated among numerous technical managers, thereby making the acquisition of consistent information a challenge. As such, there is typically a lack of consistency among the deliverables provided by various technical managers.

I always allocate time to create a Big Picture Information/Data Architecture & Workflow diagram, specifically from a testing perspective. This is an invaluable document for the test team. As an added bonus, I have found that it instantly gives me a little more “street cred” with the technical folks both on the vendor side and within the corporate systems development team.



What does the diagram contain? I create a visual representation of the end-to-end process using simple boxes and arrows. Each box represents a system (not necessarily the server or hardware breakdown—we'll do that later when we discuss environments) and each arrow represents a set of data either being sent or received between systems (the backend data integrations). I try to situate the COTS system in the middle of the diagram and place the other systems around it. The arrowheads show the direction of the data flow between systems. The arrow's pattern (solid, dashed, etc.) identifies the method in which the data is sent (nightly batch or real-time data transmission, for example). The arrow's color identifies the metadata (e.g., order data, employee record). I then layer text over each arrow line to label the frequency of data delivery (e.g., real-time, nightly batch--time TBD, user-triggered pull, Mondays at 8am EST).

This Big Picture Data Flow Diagram will help you see the forest through the trees, as the saying goes. Additionally, you will need it when you plan End-to-End Data Flow Testing, which I'll get to in a bit.

Lesson #6: Integration Testing requires testers with specialized skills.

Each type of testing requires Testers who have specialized skills and knowledge. Testers are not always good at testing everything. Time and again, I have seen the senior management team hire a front-end GUI functional tester to design and execute backend database tests. Typically this does not work well. It's comparable to hiring an accountant as an investment banker. The assumption is that if they both work in finance, their roles must be interchangeable, right? Wrong. This applies to different types of testers as well.

Backend integration testing requires testers with specialized skills, such as understanding how to access and view tables in a database, how to write database queries (e.g., SQL statements), and how to design and execute tests without a front-end application GUI. The Integration Tester tests data scenarios that might not necessarily be triggered by an end user but rather by backend system processes. The Integration Tester also needs to be comfortable with creating test harnesses that simulate data integrations, such as SOAP or writing XML. This is a different skill set than is needed to test a front-end GUI application where the tester is simulating mostly user-triggered actions within the system.

The thing to note is that when you are planning the test effort for a COTS implementation, it's important to know from the start that you will need to augment your test team with testers who have a specialized skill set in order to test the backend data integrations. Don't wait until it's too late to make the business case for these necessary resources.

Lesson #7: If the COTS system integrates with more than 1 system, including a data warehouse and a reporting suite, then separate the data warehouse and reports as their own test tracks from all other backend integrations.

Integration between a COTS system and a data warehouse is typically fairly complex. In addition, the end users typically desire a number of new and/or updated reports that contain data sourced from the COTS system and pulled into the data warehouse.

As I mentioned previously, backend integration testing requires a special skill set. Testing Business Intelligence (BI) applications, data warehouses, and reports requires an even more niche skill set. If your COTS implementation contains a data warehouse integration, I suggest that you identify the need for a special testing resource with experience specifically in data warehousing. Understanding the lingo as well as how to test data transformations from source systems through ETLs to data marts and to reports is a highly data intensive effort.



Also, make sure to add this test effort as its own test track to your overall test strategy: Data Warehousing & Reporting.

Lesson #8: Performance Testing on a COTS system is like nothing you've ever done before.

COTS systems are typically heavily data-dependent. When talking about ERP, CRM, HRIS, Sales and Order Management systems, it's all-data, all the time. If you are creating a performance test scenario suite on a data-heavy COTS system, your test scenarios might require many prerequisite data needs that are challenging to design and create.

Most of the time, COTS performance tests don't focus on scaling up to millions of users, like web apps. For many COTS systems, there is a set number of users and set number of "per user" licenses supplied for the vendor software. Yes, part of the performance testing is ensuring that the volume of users can be accommodated without degradation of the system. But most of the time, the focus is on how quickly data transactions can occur under "normal use" and what the maximum volume of transactions is before system degradation occurs.

For example, let's say one of your performance scenarios is validating the volume of "order approval" within a sales system whereby the "order approval" is triggered by an order price total exceeding \$30,000. Let's say the requirement is that the COTS system needs to process a minimum of 1000 order approvals in less than 30 seconds and without taxing the servers.

The first thing that jumps out is that you need to have 1000 orders in the COTS system. AND these orders need to be for a product or service whereby the order cost total is in excess of \$30,000. AND these 1000 orders need to be in a state whereby they have moved through the prior steps of said workflow and are in a state that is ready to be approved. That's a lot of prerequisite test data set-up. And that's just one iteration through this one test script, let alone multiple times. We haven't even talked about the other scenarios, or the use of a specialized performance testing tool.

Performance testing on COTS systems requires detailed test data planning as well as the ability to work with DBAs and Networking Teams. Again, like data warehousing and backend integration testing, you should plan on having specialized performance testers who are familiar with working on COTS systems and will work on the Performance Test Track. This need should be identified upfront.

Lesson #9: If the COTS system is replacing another legacy system, or multiple legacy systems, then you should include a Data Migration & Data Conversion Track into your test effort.

This is an extremely important test track that quite often gets neglected by the test team. While it is true that data migrations and conversions can be complex and technical, that does not preclude the need for stringent testing on the data itself. If you want to ensure that you get accurate data, then this testing is critical.

Data Migration is required when an existing legacy system is being retired and its functionality is being replicated in the new COTS system implementation. From an end user perspective, ideally the user is performing his or her job in the legacy system on a Friday. When the user comes back to work on Monday, the shiny new COTS system is ready for him or her to use, containing all of the records and accounts that the user needs. Voila! It's as if a magician pulled the new system out of a hat.



For example, let's say an organization is sunsetting an old customer relationship management (CRM) system and replacing it with a new COTS system---a big, fancy, all-in-one enterprise resource planning (ERP) system that includes a CRM module. Well, Sally the Sales Manager has years' worth of data in the CRM system, both historical and reference data as well as active or in-process data, such as accounts, opportunities, leads, etc.

In order to do her job well, Sally can't just waltz into work on Monday and start using an empty ERP system without any of her data from the old CRM application. No, Sally needs to have all of her opportunities and historical data for forecasting right there in the new ERP system, ready to go. This is called data migration---moving and mapping data objects from an old system to a new system, while remembering that it's not a simple data dump. The new system uses a completely different means to store, retrieve, and display Sally's data, so there is an element of migrating the data that needs to be considered to make it useful. This means plenty of opportunities for defects.

In addition, there is an element of data conversion that may be required. This is where data from the legacy system might need to be massaged, or converted, in order for the data to be used in the new system. For example, in the legacy CRM system, Sally might have an opportunity for a sale of widgets. That opportunity object typically has a number of different object states (e.g., new, open, approved, hold, order). In the new system, this same opportunity workflow might have 10 states rather than 5 states and it has been decided that any opportunity objects that have a status of "open" in the legacy CRM system need to be mapped to 1 of 3 different states in the new system, based on other data factors. So, the data needs to be massaged in order for the data to be useful in the new system. Again, this situation is rife with opportunities for defects.

By now, you know what I'm going to recommend, right? You guessed it! You need yet another specialized team of testing resources to focus on data conversions and migrations. The data migration requirements are typically the most poorly documented of all of the development efforts, making this a big challenge.

These tests need to be designed for two efforts. The first test effort focuses on functional testing of the interface. For example, is the data converting and mapping correctly, as well as accurately, without data loss or duplication? The second test effort is designed to validate the real-time conversion & migration within a specific (typically short) timeframe. Does the test execution and validation occur in minutes and hours rather than days and weeks?

At this point, I bet you are thinking that your test team is getting fairly specialized, right? Good. It should be. Just remember: you shouldn't have an accountant acting as an investment banker and vice versa.

Lesson #10: End-to-End Data Flow Testing Is a Really Good Idea

I define End-to-End data flow testing as being when data is created and/or used to simulate real-world usage of the COTS system and all of its integrations with other systems. Ideally, this test effort occurs after the COTS system has been system tested and after all of the individual integrations between the COTS systems and other systems have been executed to an acceptable standard.

End-to-end testing helps to identify data scenarios that might cause unexpected results as well as uncover data accuracy and correctness issues that were not diagnosed during integration testing. More often than you would expect, defects arise not due to the incorrect functionality



of the COTS system, but due to the mishandling of unexpected data--types, formats, and character lengths. Sometimes the requirements and design specs do not provide the full scope of information needed in order to adequately validate the system and integrations. Sometimes it isn't until end-to-end testing that a data defect rears its ugly head for the first time. I always say, "Better late than in Production!"

Planning an End-to-End test is extremely labor intensive. It requires the test designer to document specific data inputs and the expected result for each input. Ideally, the data output created by executing a test case is then used as the data input for another test case, and then this process is repeated (also known as "systematic testing"). In essence, you are building a very long linear test scenario. Computing and calculating the expected data results in a long scenario, with data inputs sourced from multiple systems, for example, is difficult and tedious.

An example of a long, data-dependent end-to-end scenario would be as follows:

Create Order Scenario

- 1) Create a customer account in a CRM system
- 2) Validate the backend data integration between CRM & Order system (did your customer account make it into the Order system properly?)
- 3) Create the actual order for the customer you created within the Order system
- 4) Calculate sales tax based on the geographic location of the customer's shipping location
- 5) Approve the order "behind the scenes" in the Order system
- 6) Enter payment data (such as a credit card)
- 7) Simulate the Payment system backend data integration to approve the credit card authorization (was the credit card data sent and received properly?)
- 8) Simulate the Fulfillment system integration that receives the order to be filled
- 9) Simulate the Inventory system integration to reduce inventory by the items purchased
- 10) Simulate the receipt of payment into the company's Financial system

This exact scenario can be executed a million times with various data combinations that, depending on the business rules and requirements, could generate vastly different expected results.

As you can see, designing the data scenarios with accurate data inputs and expected data outputs is a tedious and detailed process. It requires a test designer who understands the Big Picture of the full COTS implementation. This includes the COTS system itself, the business use cases, the full integration workflow between 10 or more systems (this is where that Big Picture Visual Diagram mentioned in Lesson #5 comes in handy), and real-world data scenarios. It also helps to be extremely detail-oriented. Finding a seasoned tester with all of these skills is typically a challenge. It's rare that any other member of the IT project team would have this breadth of understanding of the entire implementation as well as understand the detailed data elements across all integrated systems.

I don't even need to state this, do I? But I will anyway. You need to identify the need for this senior, seasoned, specialized tester as part of your test strategy as well as create an End-To-End Data Flow test track as part of your overall test strategy.

Lesson #11: Determine if User Acceptance Testing (UAT) Is Managed Within Your Purview or If It's the Responsibility of the Business End Users



I have worked in organizations whereby the end users desired my help in planning the logistics and scope of their UAT effort as well as help in ensuring it runs smoothly. Sometimes I've acted as the liaison between the end users and IT. In one Fortune 10 organization, surprisingly, the end users balked when they saw how organized and practical my UAT approach was---they had always just set a few hours aside to allow the end users to conduct some exploratory testing and provide a simple "thumbs up" or "thumbs down."

It is imperative that you identify what your involvement is with UAT at the onset of designing your COTS Test Strategy, or you will be left scrambling to coordinate a fairly complex effort at the last minute. I know that I'm beginning to sound like a broken record, but if UAT is under your purview, then you want to assign the right person to work with end users. In addition to being knowledgeable about testing methodology, it is critical that the tester's personality works symbiotically with the end users.

I can't stress this last statement enough. Properly managing the Business Users' expectations throughout UAT is essential to having a successful UAT. Your organization's end users can identify a million issues, but these may not be true defects and instead may be change requests or training issues. Whatever the issue, it is up to your test team to manage the users' expectations such that UAT does not turn into a large-scale development effort in the immediate timeframe before the scheduled launch date. Choose your test team liaison wisely.

PART III – The Symphony Orchestra Plays the Lincoln Center, or, What To Do When You're On Stage and Something Doesn't Go Quite As Planned

One of the most important qualities of a Test Project Manager is flexibility. More accurately, "smart flexibility." You have to be able to roll with the punches during a COTS implementation---when requirements change, when the test environments aren't available, when the vendor's development is late, or when the DBAs accidentally overwrite the test database snapshot losing weeks' worth of work. These things happen.

The greatest thing that I have learned is: be prepared for these types of events to occur. By being prepared, you'll be well-suited to handle the situation with grace and ease. Instead of complaining and acting out of frustration (Who, me...? Never...!), you will be brainstorming out-of-the-box solutions to help keep the test team and the project on track. (Did I really just use that old cliché?!)

Lesson #12: You're About to Walk Onstage When You Notice You Are Missing Your Violin

How can a musician play music on stage within the symphony if her violin is missing? Darn good question. It simply can't be done. Sure, the violinist can stand on stage, well-prepared with her sheets of music in hand. She might even have her bow. But the music and the bow have lost their value to the violinist if the violin itself is missing.

In the testing world, the violin is our environment where we execute tests on the system. It's imperative that an environment for testing is available. An environment can consist of one server, multiple servers, or many servers, depending on its purpose (e.g., sandbox, development, testing, staging, production). One of the main challenges within Corporate Systems environments is the cost associated with having multiple environments or multiple instances of the same application. In addition to the hardware cost to host multiple servers and the staff resources to support these, there is typically an even higher cost associated with the licensing of the COTS system.



Many vendors only allow a minimum number of non-production environment installations of their application at a client site. Many times, this may be limited to three non-production environments: a Development environment, testing environment, and staging environment. I can tell you right now that I've never seen a large-scale COTS implementation work well with so few environments.

Availability of testing resources is always a challenge. Within the projects I've worked on, the demand for test environments has always far exceeded the supply.

Most recently, I had a need for four concurrent test environments: performance, UAT, functional testing, and an integrated environment. Senior Management folks thought I was crazy...until I showed them my environment requirements, data constraints, security constraints, and time constraints. If they wanted testing done well and on time, this is what I needed. Guess who got her four dedicated environments?

The critical lesson I learned here is to define your environment requirements upfront in your strategy because you will likely need more than one dedicated test environment. These are some of the requirements that you should identify, per test effort track:

- 1) Hardware & Platform requirements: Does the test effort being conducted require that the hardware replicate the Production environment, which is critical for Performance testing? Or can the environment be a virtual environment (functional/system testing)?

You also need to document the specific versions of the database servers and O/S. Also, note versions of database servers and O/S. Server teams can get tricky in an effort to cut costs. Next thing you know, you're testing on a Linux server when the production server is Windows.

Also ensure that the database server for your COTS application is dedicated to your COTS application and not sharing with anything else. Never assume your test environment is dedicated to just your test effort.

- 2) Application requirements: Does the application under test need to be the latest and greatest version, even if the environment is updated daily? Or do you need to have more control over the environment such that it's stable and not updated for "x" amount of time in order to complete a cycle of testing?
- 3) Data requirements: Does the data in the system need to be controlled and known without any updates, deletes, or newly created records (data warehouse integration and reporting testing)? Or is the data not important in this environment (functional testing)?
- 4) Security requirements: Is your data a copy of sensitive production data that then needs to be massaged and masked before use (HRIS systems with payroll and SSN data)? Or does your data need to be an EXACT copy of production data in order to properly validate (legacy system data migration)?
- 5) Integration Requirements: Does the COTS system need to be integrated with another system in order to conduct testing (integration testing between COTS and System A)? Is



there a non-Production instance of System A that can be used and what are your requirements for that system?

- a. Important Lesson here: I once sent data from a COTS system to System A in the morning and planned on validating the data in System A in the afternoon. Well, System A was owned by a different Division and their process was different than ours---they refreshed all of their non-Production environments daily at noon with Production data. Guess what happened to my test data? Yep---gone! So it's extremely important that you not only identify requirements, but also constraints, especially when dealing with integrated environments.

Displaying the testing environment requirements with a visual aid is always helpful to me. With a little more elaboration, that visual diagram I mentioned in Lesson #5 is a good way to show the types of servers, platforms, and versions you will need to help identify your requirements.

The goal is to have the minimum number of environments, which allow you to meet the requirements for each test track within your test effort. That number must a) meet your requirements, b) be available, and c) be supported. If you have contradicting environment requirements for two or more test efforts, then you need to have separate environments for each test track, or, conduct the test efforts in series.

Due to cost, set-up, and support of multiple environments, it's imperative to define and gain buy-in on your needs. You don't want to be left on stage without your violin.

Lesson #13: Getting involved in Change, Risk, Configuration, and Environment Management

Change happens. There's no way around this. The sooner you accept this, the more successful you will be as a Test Project Manager.

Changes come in all shapes and sizes. There can be a requirements change---new requirements, removed requirements, changed existing requirements. There can be a resource change within your test team or within another project team. A budget change can occur, which means the management team is no longer able to purchase those four new dedicated testing servers they promised you. The end users' business can change, causing all sorts of downstream changes.

The important take-away here is to set yourself up in the Test Strategy as a member of the Change Control Board, or Change Management Team. If your organization doesn't empower you in that manner, then just request to be a non-voting, or non-opinion giving member---you just want to be kept in the loop in order to better manage risks and impacts that affect the overall project with regards to the test effort. The same applies to Configuration Management and Environment Management---get yourself involved, even if it's just a weekly meeting with the team that manages the environments.

In order to be successful, you need to be in a position to handle change, and you need to know when change is coming and to be able to readily assess the impact on you and your team. The important lesson here is to assess direct and indirect changes and how these could potentially affect your team.

For example, let's say the only server in the development environment dies. Does that directly impact your test team? It doesn't seem like it would, but it might. If the project team has exhausted its budget, then there might not be any available funds to purchase a new development server. And, if you have four servers dedicated to testing in your environment,



guess what might happen? You guessed it! You now have three servers. So, it's extremely important to keep yourself in the loop on all changes that occur on the project so that you can assess the potential impact and be prepared.

Should something like this happen to you, being prepared means being able to assess the situation with a level head. Never say, "We can't test now!" Instead, assess the impact on your test effort. What are the options? You probably can't utilize another environment concurrently, due to constraints. But perhaps you can propose to move the timeline of one test track to occur in series after another and share that test environment. If timelines are important, then perhaps you cut the scope of testing of both now-occurring-in-series test efforts in half? These are creative solutions that you need to provide as a Test Project Manager, presenting the risks and impacts of each one, and then have senior management select which option they prefer, and for which they are willing to take on the risk, if any, to the project.

Lesson #14: Push the Automated Functional Testing until after the initial go-live launch.

Since automated testing sounds great and sexy, you will be tempted to use it as early as possible. Resist this urge. In the initial launch of a new COTS system, it's just too early. The system is too volatile and the scope is still changing. Don't spend time scripting tests that will need to be rewritten day-to-day. I highly suggest sticking with manual testing for the initial launch. Otherwise you will be in a cycle of rework and rescripting over and over again, especially if the vendor is custom coding a significant amount of the application for your organization.

If you do want to get a "quick win" with your senior management by utilizing some test automation, use scripting to generate your test data, perhaps for performance testing. Use scripting to create a Smoke Test (or a Vendor Release Readiness Test). By doing so, you will be incorporating automated scripting into your project early-on while reducing the risk of rework to your test team.

Summary

COTS testing is a complicated and difficult test effort to manage well. It is important to view yourself not only as a Test Lead but as a specialized Project Manager. You **are** the Test Project Manager no matter what title you have. In order to be successful, you need to start by putting together a Big Picture Test Strategy. The Test Strategy needs to incorporate all of the test tracks, the specialized resources, and the dedicated environments. You need to be able to manage all of the moving parts and assess how any change impacts your world. Most of all, you need to be "smartly flexible." You need to be able to accept change and offer up creative solutions to challenges.

If you can do all of this, then your testing symphony orchestra will appear to perform seamlessly during the live concert. The audience will never know about the challenges overcome during rehearsals or that your violin was temporarily missing.

-Virginia Reynolds