CHAPTER 18
TESTING

Have you every wondered why a software program blows-up the first time it is used, even though those who developed the program insist they previously tested it? Multiply this situation by hundreds of ERP programs that were "tested," and then imagine the chaos at go-live. This scenario is not unusual.

In fact, inadequate testing is a double whammy. It results in many lingering software bugs that are eventually discover once the system is in production. It also leads to less software knowledge for the project team since testing is a significant part of learning. A team that does not understand the software means inadequate documentation of the new work procedures and poorly trained end-users. Once the system is live, lack of user knowledge creates as much confusion within the organization as software bugs.

Much has been written about software quality assurance (SQA) over the years. The purpose of this chapter is to draw upon some best practices that can be applied without the need to become a software quality expert.

Testing Phases

The project should include several types of testing such as the Conference Room Pilot (CRP), Integrated Conference Room Pilot (ICRP), Limited Parallel Pilot, User Acceptance Test, Volume/Stress Test, and a System Cutover Test. Within the CRP and ICRP test phases, there are several *rounds* of testing in each. Each round represents a greater scope and depth of testing.

Conference Room Pilot

The Conference Room Pilot is the first phase of formal testing. It is mainly concerned with testing the programs *within* each software module. As with any testing, this includes the standard programs that come with the package and all custom developed programs.

Each application team is responsible for testing their software module during CRP. The concept is to first test each module of the system independently to ensure it is working properly before adding more complexity.

Integrated Conference Room Pilot

The Integrated Conference Room Pilot is where it all comes together for the first time in the system. The goal is to test all software modules functioning together. Just because individual modules work fine when tested individually during CRP, does not mean they work well when interacting with one another.

During ICRP, the team tests related business processes across multiple modules. Therefore, ICRP requires a higher level of coordination and communication between all application teams when planning and executing the test and when resolving issues.

Back to our simple business process example: ICRP testing of the *Quote-to-Cash* process might begin with a single order transacted through its entire cycle. The processing starts with setting up a customer and items. In general, the next steps are to enter a sales quote, convert it to a sales order, check credit, generate a warehouse request, picked and shipped the product, and then invoice the customer. Along the way, there will be behind the scene system updates to the inventory, accounting, and perhaps other modules. Finally, the payment is collected from the customer, invoice deductions are applied, and credits are issued within the accounts receivable module. This also includes more updates to the accounting module.

As one can see, the complete cycle for a major business process may include up to five or more software modules. If something is not setup correctly in one area of the system, the entire process can come to a halt. This is the reason integration testing is very important.

Test Cases

Within most business processes, there are many scenarios or variations to what can occur. In the sales order example previously discussed, there will be order cancellations, backorders, and pricing issues, to name a few. In fact, a single process may have well over a dozen scenarios that occur on a daily basis, let alone, other things that happen less frequently. Moreover, the business scenarios are not mutually exclusive. Many can occur at the same time, and unique customer requirements always add a different twist.

A test case is the vehicle to define each variation to a process and to ensure it is properly tested. It documents a business procedure (scenarios) to be tested, responsibility for testing it, and provides a place to record test results, a disposition, and corrective actions. The first objective is to define specifically what to test. Randomly meandering through the software is not really testing.

A test case that is exercised in the system should have only one of two possible dispositions: Either it *passed*, or it *failed*. In order to pass, the process should proceed through the system with no glitches. Therefore, any software bug or setup issue results in a failed test case. In this situation, the fix is eventually made to the system and the entire test case should be performed again. If necessary, this cycle should continue until the test case passes. It is important to minimize the number of failed test cases that must be revisited in the next round of testing.

In the situation where there are no software issues, but policy or procedural issues still remain, pass the test case, and add the item to the issue list. The actions necessary to resolve the issue could require additional test cases to ensure the solution is working as desired.

Testing Oversights

Listed below are common pitfalls during CRP and ICRP testing:

- **Failure to Unit Test Custom Programs in Advance.**
  Many of the standard programs within the package have been *unit tested* prior to CRP as part of prototyping and the design phase. However, this is usually not the case for programs containing custom code. Unit testing of custom or modified programs should occur prior to CRP (or ICRP) as part of developing the software. This testing should involve the developer, application consultant, and the functional analyst. There is no point in introducing a custom program into formal testing that does not meet basic user requirements or is full of bugs. This slows down CRP or ICRP because of software rework that could have been easily avoided with proper unit testing.

- **Inadequate Test Coverage.**
  A term often used within software quality assurance is *test coverage*. The proper coverage is when the great majority of business scenarios have been tested in the software. While it is not possible

to identify and test every conceivable activity within the system, aim for 110% coverage with the hope that 95% of the scenarios that could occur are tested.

Without this philosophy, the many one-off software bugs and procedural issues from scenarios not tested can accumulate into one major go-live mess. This gets back to defining the test cases. A failure to test a range of business scenarios or deviations to the normal workflow results in a lack of test coverage.

The final point is that users can (and will) make mistakes when using the system. Many of these mistakes are predictable, so develop test cases to create the data issue, and then try to reverse it in the system during the test. The steps to correct common user mistakes should be part of the work procedures and end-user training. This will reduce the number of questions or crises after go-live since users will have some understanding of how to address the problem themselves.

- **Allowing the Consultants to Do Most of the Testing.**
  Each application team should have a functional analyst and team leader. These employees should be from the user area and be knowledgeable about the processes in question. For this reason and for the purposes of further knowledge transfer, they should develop the great majority of test cases and perform 90% of all testing. By now, they should understand the software well enough to test it.

  Software consultants can help develop the test cases for the first round of CRP only (to help jump start the process) and assist with some early testing. All consultants should have some basic scenarios to test first. This allows for getting past the most obvious software issues so the team can take over from there.

  During the time the functional analyst and team lead are testing, the consultant should provide support when needed and periodically verify what is happening with the data in the system. “Black box” testing by the users does not always uncover potential data anomies lurking in the background that can cause problems later.

- **The Test System Does Not Represent Production.**
  In order to have confidence in the test results, the test environment should reflect what the system will look like when in production. The problem is that many times certain hardware, equipment, security, and menus that will exist in the production system are rolled into testing too late in the game. Therefore, these items receive very limited testing or cause issues that require the team to retest areas of the software that previously functioned correctly. Start testing all hardware and equipment representative of the production environment during CRP. Security and menu setup can wait until the latter stages of ICRP.

- **Using Data Conversion Programs Too Early.**
  It is always recommended to start on the design and development of data conversion programs as early as possible since many are used fairly early in the testing cycle.

  When to introduce data conversion into the CRP is a different topic. Instead of using the data conversion programs for the first round of CRP, load the test data manually for three reasons: First, one does need much data for round one of CRP; in fact, too much data can get in the way.

  Second, if the data is converted using the programs, and then issues arise during early testing, it is more difficult to discern whether the problem is with the software or a data issue caused by the conversion program.

  Third, the first round of CRP testing will probably necessitate changes to the data conversion programs due to new discoveries about the software. Make the changes and then use the conversion programs starting in the second round of CRP.

- **Not Fully Testing Interfaces and Software Modifications.**
  Interfaces and software modifications are usually the most difficult custom programs to write. If not carefully tested, these programs can become the Achilles' Heal of the project. During CRP and ICRP, one cannot test interfaces and modifications enough.

- **Failure to Regression Test.**
  After each round of testing, it will be necessary to make software configuration changes to address problems. Once these changes are incorporated, it is a common mistake to assume other related programs will continue to function correctly. Due to the integration of the system, sometimes a configuration change in one area breaks other areas of the software.

  After verifying an issue is fixed, it is important to retest other programs that could be affected by the configuration change (i.e., regress). Regression testing does not require a new test case and does not have to be extreme; otherwise testing would be a never-ending loop. Nevertheless, some retesting is always recommended.

User Acceptance Test

At some point, the power users and major stakeholders should participate in the CRP and ICRP. But once this testing is complete, they should be responsible for organizing more end-users to conduct a separate user acceptance test.

The concept behind user acceptance testing is that the project team has finished their testing and believes the system is now functioning as required. However, other managers and end-users not involved with the project to this point, may have major concerns.

Acceptance testing represents the last chance for users to identify any major issues prior to system go-live. Though not considered training, acceptance testing is also another opportunity to transfer software knowledge to the user community prior to end-user training.

Acceptance testing is usually less structured than CRP and ICRP. Nevertheless, when this testing is successfully completed, some project managers require a sign-off by managers participating stating the system satisfies user requirements. The nature of the project and company culture determines the necessity for a user sign-off. But when sign-off is required, the users will probably take this testing more seriously.

Limited Parallel Pilot Test

This type of pilot is not performed in a conference room, but within the actual work area. It is limited in the sense that it occurs within a contained area of the business, and it lasts only a few days or perhaps a week. It is parallel since it involves end-users using both the new and current system simultaneously.

No parallel pilot is a substitute for conference room pilot testing, but it provides the important opportunity to use the new software, procedures, interfaces, etc., in the real world.

There comes a point when the team can continue to test in a conference room, but not learn what can be learned in a three-day pilot. This is because it is not possible to anticipate every scenario that can occur in the actual work environment. Not only is it another chance to identify and fix issues that might otherwise fall into the cracks, but it is a great learning experience for the team and users.

It is important to select a pilot location that best represents the scope of the software footprint because pilots take time to plan and execute. An easy or narrowly defined pilot may limit its value, considering the effort required to conduct a pilot.

For example, the product service department is often an excellent pilot location because of the software coverage. The department is somewhat contained but may use most of the software functionality to be used by other departments. These processes might include sales orders, pricing, purchasing, inventory, and work orders.

Volume/Stress Test

When system performance or stability issues arise at any time (even subtle ones), do not ignore them. Beware if the system is slower or crashes occasionally during testing or end-user training. You had better find out why and address the problem, because there will be many more users pounding on the system when it is in production.

To be proactive, some level of volume testing is recommended. This type of test is about throwing as much load on the system as possible to be reasonably sure the system can handle the transaction volume expected in the production environment. It is usually not possible to replicate the load that will occur in production. Nevertheless, in the test system it is wise to run batch programs that consume a large amount of system resources all at the same time, while a fair amount of users are performing interactive transactions. If transaction processing is slow during this test, it will probably be a lot worse in production.

System Cutover Test

When it is time to cutover to the new system, usually the business operations must cease temporarily in order to get a clean transition from the old system to the new (though normally some early conversion steps can occur while the business is still running).

Therefore, in order to perform the conversion, there must be a window of time with no users on the legacy system to successful complete the majority of steps in the cutover plan. This plan could be lengthy and includes some conversion activities that are automated and some that are manual, and must occur in a very precise sequence.

The cutover window is usually an agreed upon weekend or a weekend with a holiday (depending on the time required to convert). When working with management, scheduling this window is not necessarily easy since there is always work to be perform within the organization.

Whatever the conversion window happens to be, you do not want to blow it with a cutover that is aborted halfway through because of a major problem. Finding another acceptable cutover window could cost the project weeks or even months. Second, a flawed system cutover could cause a rollback to the old system after only several days running the new system. This is also the reason you should always have rollback plan.

In order to increase the chances that the conversion will go smoothly, do a dry run test of the conversion plan prior to the real go-live.