

QA Best Practices Migration Model

The core of any strategy for managing the development, verification and release of software is a defined process for the migration of software from inception to production.

All IT operations require at least two environments – an environment in which to build new or changed code (referred to in the diagram at the end of this white paper as Development), and an environment for Production use (referred to as the Formal Test Phase). For very small companies – such as start up operations – this may be adequate. For the vast majority of IT operations, however, at least two separate and distinct environments are required for Quality Assurance testing within the Formal Test Phase: System Test and Model Office. Separating the Development environment from the System Test and Model Office environments produces significant advantages:

1. **IT can update software quicker to respond to business needs.** Having separate QA test environments allows QA to complete verification of an assembled version (Version 1) while IT is simultaneously working on developing the next subsequent version (Version 2). Without the QA environments, verification of the next release and the development of the subsequent software release could not occur at the same time.
2. **Development is more efficient.** If Development were the only step before Production, Development would have to be a mirror image of Production, making Development a more expensive environment and allowing less flexibility and efficiency to developers.

Roles and Responsibilities of Development

A Best Practices Testing Model starts by putting the initial onus of the project on Development. Development is charged with several things:

1. **Unit Testing:** Development is responsible for properly unit testing all code created prior to sending it on to be Integration tested. Code should not have any problems running or operating on its own.
2. **Integration Testing:** Development is also charged with bringing all the individual pieces of code together and testing them as a standalone version. The code should be able to work independent of all other applications and not have any major bugs or issues.
3. **Packaging:** After Development unit and integration tests all code, it packages the code together and documents the release notes. The release notes should contain directions on how to install and implement the code into production and list any

known bugs or issues that were found in unit and integration testing and currently exist with the package.

Migration Process – Development to System Test

QA preparatory work (e.g., setup, getting up to speed on the new version, developing test scripts, developing a test schedule) should start during the Development phase so that the QA organization can hit the ground running when development is complete.

As Development nears completion, a Turnover meeting is held with QA, business area stakeholders and the Development team. During this meeting, the turnover status, date and process are defined. Plans are developed for migration of the new version into the System Test environment. The dates for when Development will be done with Version 2 and when QA will complete testing of Version 1 need to be evaluated to determine when the application will be migrated.

Pragmatically, in most cases, the turnover will not be totally clean. There will still be open bugs and there may be application components that have not been developed by the start of testing. The QA organization needs to assess the impact of those open items and develop strategies to work around them by first focusing on testing components that are ready.

System Test Environment

The System Test environment is an ever-changing environment because fixes are constantly being delivered to the environment. It is designed for the QA Organization to test the application in a stand alone environment. The application and its critical interfaces are tested during System Test. The test environment does not yet contain each and every corporate application replicating its production portfolio. Model Office is the environment that will replicate all these applications.

When the code initially arrives in the System Test environment, a shakedown test is run on the migrated code to ensure that the software migration was done properly. Following the shakedown test and resolution of problems found, full scale system testing begins.

At least two cycles of testing are completed in the System Test environment, and for large scale testing operations, three testing cycles are the norm. The number of test cycles executed is contingent upon when the code is stabilized. The rule is that there must be at least one complete regression test cycle subsequent to code stabilization. During system testing, several types of testing are completed, including: functional, usability, regression and negative testing. These testing formats involve the customers, business analysts and business testers. Throughout the System Test phase, code changes and fixes are made as needed, with the goal of stabilizing and 'freezing' the code for one final regression test prior to migration to the Model Office test environment.

As code is migrated to the System Test Packaging Area, the libraries are managed and concatenated so that the latest fixes are picked up as the final deliverables. Once all of the system test cycles have been executed and expected results compared to actual, the application is a completed version, ready for the Model Office environment.

Migration System Test to Model Office

The Model Office is sometimes referred to as the “QA environment” because it is the environment where production is mirrored, except where there are intentional omissions for reasons such as sizing or financial constraints.

The final requirement for migration to Model Office is that the QA organization presents to all affected areas the current status of the application. The ultimate decision to allow the application to migrate to Model Office resides with the Business Area. These people consider the facts presented by QA to determine whether or not the outstanding issues and bugs are too critical to proceed. Once the Business Area approves, the version is then migrated to the Model Office environment staging area.

Model Office

Model Office is the opposite extreme of System Test. Where the System Test environment is constantly changing, Model Office is very stable. There are few, if any, fixes allowed in the Model Office environment. Approval to make a code change in the Model Office environment usually requires senior management approval. The environment mirrors Production, except for those intentional discrepancies such as size. The QA organizational mission is to test the migrated application in a production-like environment and assess its associated impact on all other corporate portfolio applications.

Model Office is used to evaluate three potential areas of risk:

1. Does the application work as designed?

The assumption is that this is the smallest risk because applications coming into Model Office should already be production ready to satisfy customer, stability, security and data integrity requirements.

2. Does the application behave properly with other distributed applications?

Model Office tests how well applications behave together. Model Office treats the applications being tested at any one time as a version to be tested and released together. The assumption is that the testing during development or system testing is unlikely to have included testing with the full suite of other distributed applications.

3. Is the packaging process working properly?

Model Office tests the code packaging procedures as they occur in production. Applications are required to demonstrate a successful packaging process in Model Office before release to production.

The Model Office facility can be used for other purposes when not testing distributed application versions, such as for training or user evaluation sessions. These are not considered the primary objective of the facility and are given second priority for scheduling.

Since the purpose of Model Office is to provide production emulation, testing should be conducted in the manner in which users will actually work with the application. For that reason, the following should be included in test plans:

1. Before beginning the test, set up the desktop as a user would likely have it. If a user would have two or three other common tools open at the same time, do the same in Model Office. For Windows-based products, having a single application open at a time is almost always the wrong test strategy.
2. Don't consistently complete transactions in a single pass. In real life, work is interrupted by breaks, waiting for answers to questions and switching to immediate priority tasks. Exit the application at a variety of points with varying degrees of completeness.

Testing in Model Office does not occur in cycles. The test cases used are specifically selected from those in the System Test environment so to cover the general functionality of the application in production.

Since the Model Office environment is stable, performance testing begins. Following performance testing, the QA organization establishes a code freeze to prohibit any more fixes. Future fixes are made available in the next release or worked with in the emergency environments, depending on their criticality.

Once the code freeze is implemented, final tests are run and the final package is assembled. Release notes are updated with any new information, and again the QA organization takes responsibility for the version by gathering the current facts about the version and the testing results. The Business Area comes in to make the final call on whether the version is ready for production, or if the issues and bugs are "show stoppers."

Migration Model Office to Production

The final – and most important – migration is the release to production. For a simple release, the process may involve just changing some data tables (e.g., pricing changes) and a brief notification to users that the new data is available. At the other extreme, when significant new applications are introduced, the transition to production may need to be coordinated with training schedules and require a complex software installation or conversion process. Regardless of the complexity of the migration process, the following steps should always be included:

1. Users should be advised in advance of the changes and when they will be made.

2. Help Desk personnel should be advised of the changes, as well as restrictions and key contacts, in the event that significant production problems occur.
3. A “post mortem” meeting should be held – usually around two weeks after the application goes into production – to review how the production migration went, as well as to discuss any defects that were allowed into production and were missed during the testing process. That information should be used to make process improvements.

Versioning – Overview

Versioning is a strategy for managing software changes. When done properly, it reduces the cost of managing change while at the same time improving quality. Versioning is an approach that bundles software changes and releases them on a schedule, rather than releasing changes as each one is developed. Bundling software changes into versions or releases makes the software release process more efficient than releasing software on a 1 by 1 change because it allows for the version to be tested more efficiently.

For example, the regression testing process can be executed once for the version rather than repeatedly running the regression testing for each change. In many cases, it is not practical to run a regression test bed for each individual release, which adds serious quality risks. Consider the situation in which two changes need to be introduced that impact the billing system and therefore require billing regression testing. Releasing those changes individually would double the cost of regression testing.

The timetable for versioning varies based on the customer’s needs. A common schedule is to release routine non-integrated changes on a monthly basis while bundling end-to-end (integrated) or other significant changes on a quarterly basis.

Version Size

The size of a version can vary significantly depending upon the changes made. For most IT operations, it makes sense to differentiate at least two types of versions:

1. Major versions, in which significant new or revised applications need to be tested and will likely require multiple testing cycles before demonstrating production quality. The time allotted for these versions is usually longer than for maintenance versions.
2. Minor versions, in which the changes introduced are fixes, routine updates to applications or updates to data tables.

Schedules vary depending on the needs of the business, but a common version schedule is to implement changes monthly, with 2 to 4 major versions during the year for the introduction of significant changes. Integrated testing changes are usually restricted to the major versions. If this is not done, significant additional testing time is required to retest

every application in every version for potential integrated impacts. It is far more efficient to focus testing on the applications impacted in each version test and restrict the more comprehensive testing to a small number of integrated versions.

Versioning – Schedule Management

The version calendar will vary based on customer needs, but for illustrative purposes, here is an outline of a typical versioning strategy:

1. Software changes for each month are aggregated and released on the third Saturday of each month for non-integrated versions and on the fourth Saturday for integrated versions.
2. The testing window for a non-integrated version is three weeks.
3. A QA Manager or a designated subordinate manages the change calendar.
 - a. For example, assume that the next version (Version 3.0) is significantly behind and holding up some significant changes in the following version (3.1). The Test Manager would develop strategies to deal with that situation. Could the important changes in 3.1 be moved ahead of 3.0 in the test schedule? Or are they dependent on changes in 3.0?
 - b. For large or complex applications, a Change Coordinator position should be considered to manage the scheduling and coordination of the version content.

Versioning – Making Exceptions

Versioning makes sense as a standard practice, but there are exceptions:

1. If the change is purely data – not changes to systems logic or changes in systems settings, then versioning is not required. For example, changes in pricing or an update to a list of employee names can be tested off schedule.
2. Emergency fixes are almost always tested off schedule by definition. If they can wait – then they're probably not an emergency. However, given the abbreviated testing, emergency fixes need to be regarded as a high-risk alternative and used sparingly.

Emergency Testing & Environment

Emergency Testing is the “safety valve” for a versioning approach to production updates. It permits changes that are absolutely critical to get into production quickly when the changes are so urgent that they can't wait for the next release. Typically this can be one of three types:

1. A defect not found during testing that is causing significant production problems.

2. A last minute change imposed by external regulators.
3. A last minute change mandated by overwhelming business needs, i.e. a new requirement from a customer that must be met now to avoid significant revenue loss.

While the latter two situations do arise, the bulk of emergency testing focuses on defects not found during pre-production testing.

The primary objective of Emergency Testing is to evaluate the change requested and insure that it resolves the problem. Within time allowed, regression cases should also be rerun to reduce the regression risk. If time does not allow that, then part of the Emergency Test process is to get written agreement with user management that the change must be released immediately even if it creates a risk of even greater production disruption.

When Completed

Emergency Testing is completed after either:

1. the fix and regression cases have been successfully completed, or
2. the fix has been successfully tested and user management accepts the risk of no regression testing or abbreviated regression testing.

Test Environment Strategy

The Emergency Test Environment should be as close to a mirror image of production as possible. It can be a separate environment maintained just for emergency testing. However, in most cases, if the Model Office environment is not required for testing the next version for at least two weeks after a version, then the Model Office environment can also be used as the Emergency Test platform. This is not a perfect solution; if a problem occurs after two weeks, there is no place to test it, but that is unlikely.

Once an emergency fix is released, a determination needs to be made as to whether the same fix also needs to be made to the next version if it is already in testing. Generally the answer is yes, unless the next version has a more updated copy of that system's component than the one which just went to production.

Turnover Process

The turnover process is essentially the same as for Model Office, with the recognition that in this case "user notification" may just amount to a quick broadcast email that the system is up again.

In the turnover process, everybody involved must understand that the process is inherently risky. While the accelerated fix process increases the risk that the fix has defects, the accelerated testing process simultaneously decreases the odds that the defects

will be found. For that reason, this path is intended only for emergencies and should not be used as a “fast path” into production. Most emergency fixes result from a failure somewhere in the testing process. After the fix is released, the cause of the failure should be identified and corrective action taken.

Regression Testing

Regression testing should consist of a strategy (see the *Triage Regression Testing Guide* white paper) that focuses on the most important exposed pieces of the application and tests them as hard as is possible. The regression test bed will have test cases that cover system components that fall into at least one of these four categories:

1. **High Use**
The most frequently used features, screens and options where defects would disrupt the largest amount of production activity.
2. **High Defect**
Testing of the systems components which have over the last few versions demonstrated the highest defect rates – typically the most complex code or new code which is still experiencing instability.
3. **High Recovery**
Systems components that when broken require a difficult or complex recovery process or that are highly visible to the customer.
4. **New**
The newest parts of the system (particularly changes in this version) where we may not yet have enough experience to determine if they are high use, high defect or high pain. These newest components would get extra testing until credible production metrics are available.

The proposed methodology also calls for the bulk of the regression test bed to be automated. This creates a test bed that is repeatable. Regression tests, even ones that accept some risks such as described above, are large undertakings. Given that a regression test bed is essentially the same cases run with each version, it makes sense to automate the process. Without that automation, the discipline of running the test bed tends to erode when work pressures increase or staff turnover occurs. Automating the scripts makes repeatability practical.

Regression test beds also need to be managed. Over time, most regression test beds end up with core cases run with every version and with a library of other test cases for additional testing of areas requiring extra attention for a particular version. The total number of cases can be substantial. The cases need to be organized and the scope of each case documented. Test management software should be used to manage and execute the test cases.

Change Tracking

A central repository is one of the most important assets required to manage a quality control process. It identifies the content of each version and the status of each item in that version. The repository allows developers to know if other work that is underway may impact their project, and permits testing personnel to monitor post-production release defects found and use that data to improve the regression test bed. It provides the basis for developing production metrics to assess version quality. For all these reasons, a central change tracking system is recommended as a fundamental quality management tool.

Performance Testing

Performance Testing seeks to determine if an application can handle anticipated volumes of concurrent users and still provide acceptable processing and response times. Performance Testing is sometimes distinguished from Stress Testing in that Performance Testing measures response time, while Stress Testing determines the volume levels that cause a system to crash or cease to function. In this document Performance Testing is used to describe both of these tests.

Objective

The objective is to confirm that response times are acceptable for at least two levels of concurrency:

1. **Average concurrency**

The average number of concurrent users that would be on the system at any one time.

2. **Maximum concurrency**

The highest number of concurrent users that would be on the system under any reasonably foreseeable set of circumstances (e.g., peak periods). Generally, longer response times are tolerated at the maximum concurrency level. If problems are identified, additional concurrency levels are typically tested to identify the concurrency levels where performance begins to degrade.

Prerequisites

Performance Testing requires a stable application. Therefore, it typically starts after the application is shown to be stable in Systems Testing. It should be completed before Model Office ends, to permit the application to be frozen. Performance Testing can occur anytime in the window between frozen System Test code and final Model Office testing. If a serious performance problem is identified, the application will not move into production.

When Completed

Performance Testing is completed after testing confirms acceptable performance levels for average and maximum concurrency requirements.

Roles & Responsibilities

Performance Testing can only be done on a practical basis by the use of automated test tools that can emulate potentially hundreds or thousands of users. Typically, this testing is performed by a specialized unit familiar with these tools. However, that unit will need support, which will require involvement of testers and Business Analysts. For example:

- The concurrency requirements need to be defined.
- The acceptable response times need to be defined. If not defined in advance, then someone must at least review the response times identified in production to determine if they are acceptable.
- Test scripts must be provided to the Performance Test Unit. If the Performance Test Unit uses a performance testing tool from the same vendor as the capture/replay tool, then no re-scripting is usually required.
- Developers are involved in the evaluation of results and the resolution of problems identified.
- Developers may also involve network management personnel if problems are network related.

Types of Testing Conducted

Testing is typically conducted by creating two types of virtual users: DB users and GUI users. DB users are created by running a transaction and recording the information and responses sent to the application. At replay, the system is not run again; instead just the data is sent. Using this method, hundreds of virtual users can be run from the same computer. These create the “background noise.”

On top of that are layered the GUI users. These virtual users are actually running the scripts and going through the application screen by screen. The GUI users show what the performance level would look like from a user’s perspective and identify the specific points in the application where performance degrades.

Test Case Strategy

Performance Testing is entirely automated. Cases should be a representative sample of the types of activities that take place in production so that the test is realistic.

Change Tracking

Typically problems found by the Performance Testing team are referred back to a contact in the application area who documents them. Performance problems should be logged just like any other defects.

Test Management

This would likely not be used except to identify scripts for use by the Performance Testing team.

Capture/Replay

As noted, all virtual users are created via capture/replay.

Turnover Process

Turnover normally occurs at a meeting or teleconference with Performance Specialists, Testers, Business Representatives and Developers.

Ideally, performance results are available for review before the meeting, but often the reviews take place at the turnover meeting.

If results are acceptable, then this information is incorporated into the normal Model Office to Production turnover process.

BT3 Best Practices Test Migration Model

