UNIX Code Migration Guide

UNIX Application Migration Guide

Chapter 12: Testing and Quality Assurance

Larry Twork, Larry Mead, Bill Howison, JD Hicks, Lew Brodnax, Jim McMicking, Raju Sakthivel, David Holder, Jon Collins, Bill Loeffler
Microsoft Corporation

October 2002

Applies to:
  Microsoft® Windows®
  UNIX applications

Summary: Chapter 12: Testing and Quality Assurance deals with the creation of a test environment, developing a test plan, and evaluating the results. It then covers the release process and release criteria that verify the application's quality and suitability for a production environment. (21 printed pages)

Contents

Introduction
Overview of the Test Life Cycle of the Migration Project
Stage 1: Plan the Migration Test
Stage 2: Define the Lab Strategy and Build the Test Bed
Stage 3: Design the Test Plan and Test Cases
Stage 4: Execute the Test
Stage 5: Evaluate and Analyze Results
Types of Testing
How to Create a DTP and a DTC
Reporting and Release Processes
Appendixes
References

Introduction

This chapter covers the suggested strategy for testing an application that was migrated from UNIX to the Microsoft® Windows® operating system or to Microsoft Interix. Testing is the process of determining the difference between the expected results and the observed results on the migrated application—in particular, the output of the parent application versus the output of the migrated application. Only after thoroughly testing and measuring the test results, as defined by test plans, can you determine whether the migrated application satisfies all of its design goals.

This chapter describes the test objectives and the proposed release criteria. It also introduces the generic testing processes and the methodology that you can employ to test your application.

The appendixes at the end of this chapter provide various templates and examples used in testing. To ensure that proper test coverage is applied to the migrated application, these templates and examples cover the following areas:

- Infrastructure
- Security
- Functionality
- Management
- Performance

Overview of the Test Life Cycle of the Migration Project

The test life cycle of a migration project is synchronized with the development stages of the project. In the software life cycle, the test life cycle follows the waterfall model, the spiral model, or a combination of both models.

Test Life Cycle Stages

In a migration project, the test life cycle consists of the following stages:

- **Stage 1: Plan the migration test**. This stage is essentially part of the plan for the entire migration project and is covered in detail in Chapter 4. In this stage, you establish the main objectives, scope of testing, test criteria, scheduling, infrastructure, resources, success criteria and requirements for finalizing the deliverable.
- **Stage 2: Define the lab strategy and build the test bed**. In this stage, you define a strategy for planning the lab environment, network configuration, and hardware and software requirements. You also create a process for installing the test bed for both the native UNIX application and the migrated application.
- **Stage 3: Design the test plan and test cases**. In this stage, you create a test plan and test cases as defined in the scope of the migration project. You use the input available in the test plans of the native UNIX application. However, if new features are added to the migrated application, some rework may be necessary. In this stage, you also establish the people who will execute the test cases, their responsibilities, and the actual schedule of the migration.
- **Stage 4: Execute the test**. In this stage, you execute the test and manage the test results.
- **Stage 5: Evaluate and analyze results**. In this stage, you evaluate the test results and use the final evaluation of the project to determine how to approach the next phase of the migration—the deployment phase.

Figure 1 represents the flow of the different stages involved in testing the migrated application. The details of each stage are explained later in this chapter.

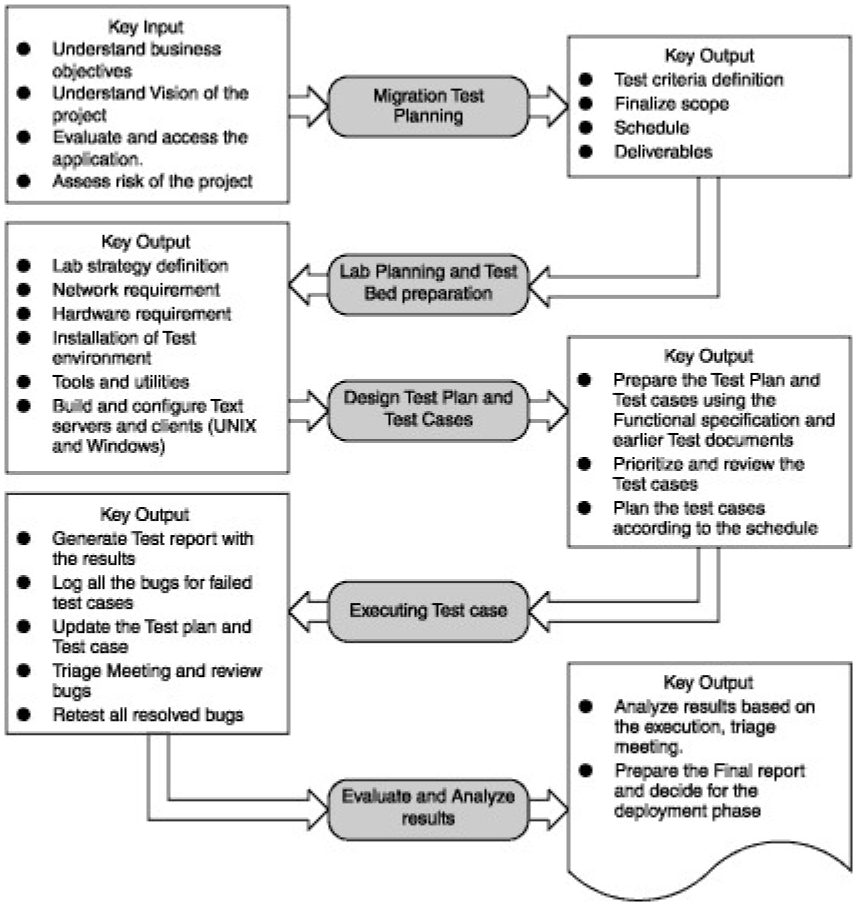
Key Input
Understand business objectives
Understand Vision of the project
Evaluate and access the application.
Assess risk of the project
Migration Test Planning
Key Output
Test criteria definition
Finalize scope
Schedule
Deliverables
Lab Planning and Test Bed preparation
Key Output
Lab strategy definition
Network requirement
Hardware requirement
Installation of Test environment
Tools and utilities
Build and configure Text servers and clients (UNIX and Windows)
Design Test Plan and Test Cases
Key Output
Prepare the Test Plan and Test cases using the Functional specification and earlier Test documents
Prioritize and review the Test cases
Plan the test cases according to the schedule
Executing Test case
Key Output
Generate Test report with the results
Log all the bugs for failed test cases
Update the Test plan and Test case
Triage Meeting and review bugs
Retest all resolved bugs
Evaluate and Analyze results
Key Output
Analyze results based on the execution, triage meeting.
Prepare the Final report and decide for the deployment phase

Figure 1. Overview of the test life cycle

Best Practices

The following best practices are recommended to ensure that the test phase of your migration project is successful:

- Create a master test plan (MTP) that defines the scope of the test phase. You should do this when you are scoping the requirements, before you begin the design phase of your migration. During the migration process, it is important to document and finalize the scope of testing and the test coverage. By maintaining an MTP, you can track the scope of the migration project from the start of the project.
- Create one detailed test plan (DTP) that captures all the scenarios related to testing the migrated application. For the detailed test cases (DTCs), you can have multiple documents based on functionality, modules, or group.
- Create a numbering convention for the test scenarios (for example, 1, 2, 3 and so on) and for the test cases (for example, 1.1, 1.2 . . ., 2.1, 2.2, 2.3 . . ., 3.1 . . .) in the DTP and the DTCs, respectively. This is very important and helps in identifying the test cases effectively.
- Review the DTP with the development team, which can provide valuable input and ensure adequate prioritization of test cases.
- Maintain a test plan matrix for functionality and for the test cases. The test plan matrix is

especially useful during integration testing. For example, if you must derive the use-case scenario for some functionality, you can use your numbering convention to map the functionality F1 to test cases 1.2, 3.2, 1.4, 4.5, and so on. This ensures that if there is any specific testing to be done, the test engineer can add or modify the matrix. Also, if any functionality is changed or modified or any bug is fixed, you can use the matrix to conduct strategic testing.

Another use of the test plan matrix is to trace the requirement changes and identify the impact of testing. If any new feature is added to the migrated application, the scope of testing that feature will have an impact on overall testing; you can use the matrix to track the scope effectively.

- Perform a root cause analysis on all bugs in an early stage of the migration project. The test team can classify the bugs based on the cause (for example, content, code, UI, design, wrong requirement). Giving feedback about these causes to the development team can ensure that future modules do not contain these defects.
- Collect metrics from the test results to determine the quality and productivity of the project and the team members.
- Track daily status. To maintain control during the migration process, it is strongly recommended that the test lead prepare a daily status report that highlights areas of concern, accomplishments, test metrics, quality, bug statistics, and action items.

Stage 1: Plan the Migration Test

Based on the size and scope of the project, the program manager can determine the types of testing that are applicable and how heavily to test the different areas of the application. The output of this stage is the MTP, which contains the test goals, the success criteria, the scope of the project and other items discussed in this section.

The MTP is a living document throughout the test life cycle. The content for this document is generated from the functional specification document and the high-level release schedule. As such, the MTP should be subject to the same change control procedures as other documents involved in the migration project.

Appendix C, "Master Test Plan Template," is a sample MTP template that includes descriptions of all the necessary areas that need to be documented in an MTP.

Test Goals and Success Criteria

The MTP defines a set of goals for the test process; these goals are essential for keeping the migration project on track. The types of tests to run on the application depend on the goals of the migration project.

The rule of thumb for defining test goals essentially depends on the application under consideration. The application should demonstrate the success criteria in three parameters:

- **Features and functionality**. The migrated application on the Microsoft Windows platform should have the same features and functionality that it has on the UNIX platform. This is the basic rule of thumb for any migration project unless there is a change in requirements. For example, if the migrated application must support 200 clients, as compared to 20 clients on the UNIX application, this is a new scope. The test plan and scope are then changed accordingly.
- **Interoperability**. The test case in this parameter is for demonstrating the coexistence of the

Windows and UNIX environments. The application may need to exchange files or access data. Ensuring interoperability decreases the complication of file and data conversion across the platform. For more information about interoperability, see Chapter 5.
- **Performance**. Compared with the performance on the UNIX-based system (for example, Sun Ultra Sparc 5 or HP-UX), the application must run faster on the Windows platform. Slower overall performance on the Windows platform is not acceptable. However, what constitutes acceptable performance depends on the goals of the migration project. If a goal is to have a higher-performance application on Windows, the testing phase must thoroughly cover performance testing. You can measure performance by using existing benchmarks whenever possible—or, when no benchmarks are available, you can measure performance based on the subjective analysis of testers and developers.

Similar parameters can be added in this definition of test criteria, depending on the type and nature of the application considered for the migration.

Based on the preceding three parameters, you can define success criteria as shown in Table 1.

Table 1. Test pass and fail criteria

Success criteria	Pass definition	Fail definition
Features and functionality	The migrated application has all the features and functionality that it has in the UNIX environment.	The migrated application fails to provide similar features and functionality.
Interoperability	The migrated application can access existing data or project files with minimum or no conversion, and can save those files back to the UNIX environment with minimum or no conversion.	The migrated application fails to gain access to existing data or project files, and the conversion process is highly complicated.
Performance	The migrated application meets the performance numbers defined in the scope of the project for the Windows platform.	The migrated application does not meet the performance goal.

Defining test goals and success criteria can add a framework to the test plan and encourage the personnel involved to agree on the time frame of delivering the project. You can derive the test schedule from the number of test cases in each of the preceding test parameters.

Assumptions related to the platform, infrastructure, and business goals of the migration are also covered in this stage of the test life cycle. For example, if the migration is from Solaris to Windows Advanced Server, the MTP must reference only these two platforms.

Scope of the Test

The scope of the test phase depends on resources, time, and business objectives. Accordingly, there are different methods that you can use to envision the scope of testing for the migration project and to create a plan for lab strategy and a definition of lab requirements. The MTP should document the scope of the test, based on answers to the following questions:

- Do you want the migrated application to be installed and started properly?
- Do you want to verify the functionality and performance of the migrated application?
- If there are issues or bugs, what is the priority? Do you want to fix bugs during the test phase or

just note them and handle them in the next phase?
- Is it necessary to test every component and module of the migrated application, or do you have high-priority modules and low-priority modules?
- Does the migrated application have all the features that the application has on the UNIX platform?
- Has any new feature been added to the migrated application? What is the scope?
- Does the application need an interoperability environment?
- Does the application need any automation testing?
- Does the native UNIX application have existing test scripts, a test suite, a test case, and a test plan? These items are critical because they can save a lot of time in testing the migrated application.

Answers to these questions help determine the scope of the project and keep the project on track as you move forward with testing.

Test Schedule

The amount of time available for testing the migrated application is critical to the success of the project. From the scope of the project and the test criteria, you can define the tasks and the time needed for completing the test phase. If the UNIX application already has a test plan and test cases, you can save a lot of time on the test cases by covering functionality testing and feature testing. Similarly, you should consider the time needed for creating the test environment and the test bed for the migrated application. In some cases, it is necessary to create both UNIX and Windows environments for comparison purposes.

The test schedule details the sequence of tasks that the test team will follow to accomplish the testing. It is usually part of the MTP; in addition, large projects may associate the test schedule with the DTP. The test schedule also clearly defines tasks and allocates them to relevant personnel. Project team members identify the different test stages applicable for the migration and select appropriate team members for the tests. Each level of actual testing (unit testing, integration testing, functional testing and system testing) will require a separate work plan, scope, and entry and exit criteria, which the test team will determine. For more information about the levels of testing, see "Stage 3: Design the Test Plan and Test Cases" later in this chapter.

The test schedule is reviewed during weekly test meetings and the milestones are tracked. Any deviation in the schedule requires immediate intervention, during which the test lead identifies alternatives to resolve the concerns.

A sample template of the test schedule is included as Appendix A, "Test Schedule Template," to help a test team plan and track the key tasks and milestones.

Roles and Responsibilities

The typical roles that may be required in a migration test team, along with the responsibilities of each role, are described in Table 2.

Table 2. Test roles and responsibilities

Role	Responsibilities
Test lead	Define test goals and generate MTP. Generate build and triage plan. Generate DTP.

	Review DTP and DTC. Track test schedule. Review bugs entered in bug-tracking tool and monitor their status during triage meeting. Generate weekly status reports. Escalate issues that are blocking, or review impact analysis and generate change management document. Ensure that appropriate level of testing is achieved for a particular release. Lead the actual Build Acceptance Test (BAT) execution. Execute test cases and generate test report.
Test engineer	Generate DTC. Review DTC and DTP. Document problems found during deployment. Conduct BAT. Execute test cases. Report bugs in bug-tracking tool. Retest bugs that are fixed. Set up lab.

Dependencies

The MTP must also document dependencies. It is essential that all dependencies are identified and listed to ensure proper and timely testing of the application and the architecture. Some of the factors that may affect the testing effort include:

- Functional specifications
- Architecture diagrams
- Design documents
- Build dates
- Material and personnel resources
- Changes in functional scope

Risks and Mitigation Plan

A risk is the probability of an event occurring, which could jeopardize the system. To evaluate risks, the test team can prepare a matrix that identifies risks and assigns one of the following exposure factors to each risk:

- **Probability of loss**. This is the probability that the risk will occur. It is usually adequate to create three levels of probability: for example, "Not Likely" (less than 50 percent), "Possible" (50 percent) and "Very Likely" (greater than 50 percent).
- **Size of loss**. This is the impact on the project timeline when the event associated with a risk occurs. Again, three levels are usually adequate: "Negligible," "Jeopardizes Finish Date," and "Significant Effect on Finish Date."
- **Contingency plan**. This is a plan for handling the circumstances of the risk. The contingency plan could entail building into the schedule an extra number of days to meet these circumstances, adding staff and other resources or changing the delivery scope.

Table 3 lists some of the common risks encountered in testing projects and possible contingency plans.

Table 3. Risks and possible contingencies

Risk	Contingency plan
Development falls behind schedule.	Determine your ability to begin initial testing in parallel with the last stages of development, and add test and development resources.
Testers are unfamiliar with the application.	Factor in additional days to train the testers on the application.
Applications are based on emerging technologies, which may result in unexpected delays.	Ensure that the schedule remains flexible.
Scope of new requirements is evolving and may increase, which may result in an unexpected increase in overall project scope.	Ensure that the schedule remains flexible.

For an example of the risks identified when a migrated application is tested, see Appendix B, "Risks Identified for Testing."

Stage 2: Define the Lab Strategy and Build the Test Bed

Although not much documentation is needed for this stage, it is critical to define a lab strategy and to create a single or multiple test bed. It is advantageous to have a controlled and dedicated lab for both the migrated application and the native UNIX application. The number of servers and clients that must be added to the network depends on the application.

Defining a Lab Strategy

The lab strategy is an approach to setting up and configuring the lab with the available resources, and accounting for what additional resources must be added. The following questions cover some of the issues involved in defining a lab strategy:

- Does the test environment already exist?
- Does a new lab need to be built?
- What are the hardware configuration requirements for the migrated application and for the UNIX application?
- Does the application need interoperability?
- What is the functional nature of the application considered for migration?
- Does the migrated application require performance testing? Does the lab have resources for performing high-end performance testing?

Determining network requirements

To define a lab strategy, you must determine the network requirements for the lab. Most of the requirements should be derived from the assessment process (for details about assessment, see Chapter 3). However, the following questions can also help you determine network requirements:

- Do the resources in the lab require domain configuration?
- What network configuration is required?
- Does the lab require any network support?
- Does the application need any backup plans?

- What kind of support do you need for the test computers?
- Is there any database configuration? Do the databases need any specific network protocol?
- What is the operating system of the migrated application and the UNIX application?
- Does the application require Microsoft Windows 2000 Services for UNIX (SFU) and interoperability options?

Determining hardware requirements

The majority of the resources allocated for the lab fall under hardware requirements. If the migrated application is intended for a single user workstation, hardware requirements may not be a significant issue; however, if the application is intended for a client/server system, planning for hardware requirements must occur during the initial stages.

You must determine what server configuration the application requires, and whether the test lab must include any special requirements. Many times, applications that are installed on a server may require special configurations, like a separate Internet Protocol (IP) address or service package. Often, vendors themselves recommend the minimum platform requirements for their applications; some vendors even provide application engineers to help configure and optimize the server platform for the applications.

You must follow a similar approach to determine the client configuration that the application requires. A low-end system can be used as the clients during testing to reduce the cost of production. Sometimes, even workstations can be used as clients.

Interoperability requirements are another important consideration. If the application needs password synchronization, resource sharing and data sharing, you must plan separate interoperability requirements based on the testing requirements. For example, to set up password synchronization, you might require SFU user name mapping in conjunction with single sign-on daemon (SSOD) and password authentication module (PAM) modules on UNIX. Similarly, for data sharing, you might use a Samba client for Windows or a network file system (NFS) server for SFU.

Determining software requirements

The software requirements for the test phase are normally included in the initial plan for the migration project. The software list includes the following:

- Operating system, utilities, and scripting language (such as Windows 2000 Professional or Advanced Server, Microsoft Windows XP, SFU 3.0 and Active Perl)
- Any additional software packages and libraries, such as Interix and Trolltech Qt or RogueWave Software
- Test management and automation tools (such as Rational Rose, SilkTest or Winrunner) and Assessment tools (such as MigraTEC 32Direct or PorteNT)

Building the Test Bed

Using the help of system support engineers, you can plan and implement the test bed. During this stage, the operating system is installed and servers are configured, as indicated in the lab strategy. There can be multiple test beds, and each test bed can have multiple hardware configurations and systems. For example, if the application is a UNIX application running on Linux 7.1 and Solaris 8.0, and if the migrated application must be tested on both Windows 2000 Advanced Server and Windows XP, you may need to prepare four systems that have these operating systems installed.

In general, take the following steps to build a test bed:

1. Build, configure and test the physical network.
2. Build, configure and test the server or servers.
3. Build, configure and test any data sources.
4. Build, configure and test the clients.
5. Perform a final check of all lab components and overall system readiness.

Note For more information about creating a test lab, see "Chapter 4: Building a Windows 2000 Test Lab," in the Windows 2000 Server Resource Kit Deployment Planning Guide.

Stage 3: Design the Test Plan and Test Cases

During the migration process, tests must be conducted on various levels to detect bugs as soon as possible, and to limit the complexity of the tests. The following levels of tests are generally accepted:

- **Developer testing**

  Developer testing is mentioned here for completeness only. It refers to the tests conducted by developers during the development process or prior to submitting work to the tester. These tests are informal, often ad hoc, and rarely ever documented. They are useful for eliminating most of the simple bugs, and for understanding and defining the scope derived from the specification.

- **Unit testing**

  The first formal testing activity is unit testing. A unit is the smallest piece of software that can be tested on its own (functions in imperative programming, classes in object-oriented programming). Unit testing ensures correct operation of these small pieces, which form the building blocks for the complete system. In the case of a migration project, unit testing ensures the correct operation of a migrated application running on the Win32 platform as compared with the UNIX platform.

- **Integration testing**

  The next level of testing is integration testing. This is actually not a one-time activity, but an iterative process in which units are combined into components of increasing size until the system is complete. A component is a combination of units that forms a useful abstraction that has a defined functionality (a module, a subsystem, a framework or a cluster of classes). The focus in integration testing is on the interfaces between the units or components that constitute the current level of integration. Integration testing requires a detailed design or (for larger components) a specification that can be validated. Integration testing proves that all areas of the system interface operate with each other correctly and that there are no gaps in the data flow. The final integration test proves that the system works as an integrated unit when all the fixes are complete.

- **Functional testing**

  The next level of testing, functional testing, ensures that each element of the application meets the functional requirements of the business as outlined in the requirements document or functional brief, system design specification and other functional documents produced during the course of the migration process (such as records of change requests, feedback and resolution of issues).

- **System testing**

  The final level of testing is system testing. This refers to both the internal system testing prior to shipment and the final testing conducted by the customer. As the name implies, system testing focuses on the system as a whole and its visible functionality and performance. Functionality and performance are tested based on the requirements and the validation criteria stated therein. System testing also proves that the documented performance standards or requirements are met. Examples of testable standards include response time and compatibility with specified browsers and operating systems in the native UNIX application and in the migrated Windows application. If the system hardware specifications state that the system can handle a specific amount of traffic or data volume, the system is tested for those levels.

Creating the Detailed Test Plans

DTPs are created in parallel with the lab setup and development work. For large or very complex projects, DTPs are generated from the MTP for the various areas that have been identified for testing. For smaller projects, the DTPs may be included within the MTP.

DTPs, like the MTP, are based on the functional specification and other high-level design documents. The test team assigns a priority to each scenario in a DTP based on the probability that the scenario will occur and its impact on business. DTPs are usually grouped according to the organization of the test group, according to the order of the availability of various components in the system, or according to the area of functionality. Essentially, the DTPs have scenarios in the context of high-level considerations, such as functionality, interoperability or performance criteria. The test cases are derived from these scenarios. Table 4 shows the tester's responsibility for each criterion.

Table 4. Testers responsibility for test criteria

Evaluation Criteria	Tester's Responsibility
Features and functionality	Execute tests that demonstrate that the application provides the same functionality as found on the UNIX platform.
Interoperability	Execute tests that demonstrate the ability of the application to share server data with clients that are running the application on the UNIX platform.
Performance	Execute tests that demonstrate the performance of the applications on the Win32 platform. Where data is lacking to make a valid comparison with the performance on the UNIX platform, the tester uses industry-standard benchmarks (when available).

The template of a DTP is provided in Appendix D, "Detailed Test Plan Template."

Creating Detailed Test Cases

Each of the scenarios listed in the DTPs is translated into one or more detailed test cases to create the DTC documents. A DTC document essentially describes input, an action, or an event, and an expected response. The quality assurance (QA) engineer or another tester performs the test case to determine whether the specific functionality of the migrated application is working correctly. Alternatively, the QA engineer or tester can compare the results of the original UNIX application and the ported application simultaneously.

Because test case development requires working on the complete operation of the application, the process of developing test cases can help find issues and defects in the requirements or design of the migrated application. It is therefore recommended that the detailed test cases be prepared early in the development cycle.

A detailed test case contains:

- Detailed test setup instructions
- Detailed test procedures, including step-by-step instructions for executing the test case
- Expected results and success (pass/fail) criteria

Figure 2 represents the steps in test planning and test execution.

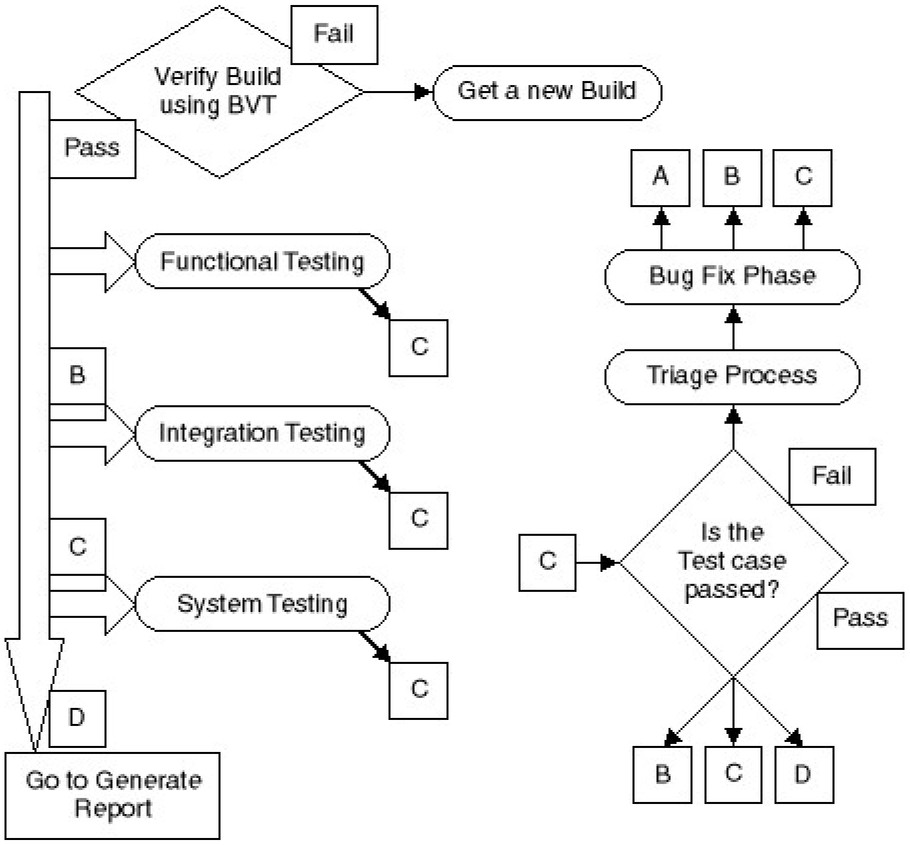

Figure 2. Steps in test planning and test execution

Stage 4: Execute the Test

After the build process is complete, the test team performs a Build Verification Test (BVT) before accepting the build and submitting it for further testing. After a successful BVT, the test team conducts a BAT, or smoke test, to verify that the build is stable enough for further testing. Frequently, these two tests are combined into a single effort.

If the BAT test is successful, the test team accepts the build and begins design testing. The test team begins design testing by performing the various tests documented in the DTCs. These tests are run on each major build, and the results are recorded and verified. Any deviations from the expected results are documented and are tracked through a bug-tracking tool.

Tracking Bugs

Any bug-tracking tool can be used to record bugs during the implementation phase. Each member of the test team needs specific permissions to access the bug-tracking tool. During testing, all members of the team should be able to record newly discovered bugs. To maintain good control over the status of bugs, only the test lead, test staff or project managers should have the proper permissions to close bugs.

Bugs can be found in documentation, procedures, and other areas in addition to the system itself. All issues with the application or its supporting infrastructure should be captured in the bug-tracking system for resolution. The bug-tracking system becomes the team's "to do" list and can be used to quickly gauge the status of the project.

A tester logs the bugs in the tracking system by using a recognizable description and comprehensive detail. All bugs should contain the following information as a minimum:

- Error messages (recorded exactly as they appear on the screen or in log files)
- Version number of the build where the bug was found
- Exact, repeatable steps to reproduce the bug
- A summary of any unique configuration or environmental factors in place during the test

In addition, a tester can include a suggested resolution if the tester knows a likely resolution.

After developers fix a bug, they change its status from Open to Fixed. If they believe that a bug should not be fixed, they change its status to Duplicate, By Design, Not Repro or Won't Fix. Bugs that are no longer listed as Open or Active must be given final resolution by the test team. The following list provides examples of how the test team resolves a bug by using the tracking tool:

- Bugs with the **Resolution** field marked as Fixed are assigned back to the originator of the bug to perform regression testing on the new build.
- Bugs with the **Resolution** field marked as Duplicate, By Design, Not Repro or Won't Fix are closed by the test team if the team agrees with the resolution.
- If the test team strongly disagrees with the resolution, the test team changes the **Assign To** field to Active and discusses in the triage meeting whether the bug should be reactivated. If the bug fails regression, the test team will reactivate it by changing the status to Active. All active bugs will be reviewed again in the triage meeting. For more information about triage meetings, see "Triage Process" later in this chapter.

Assigning Severity to Bugs

Table 5 lists guidelines that can be used to assign severity to bugs.

Table 5. Bug severity guidelines

Severity	Most common types	Conditions required
1	Bug blocks build or further testing of a feature. Bug affects further testing of a different feature that is being tested in parallel.	System does not work. User cannot even begin to use significant parts of the system.
2	Steps defined in the documentation are not viable. Results or behavior of a function or process	If the following conditions are met, severity is 2: User has no simple workaround to mend

	contradicts expected results (as documented in functional specification). Results or behavior of a function or process contradicts logically expected results. Documented functionality is missing (in this case, test is blocked). Documentation is missing or inadequate.	situation. User cannot easily figure out workaround. The system cannot meet primary business requirements. If the preceding conditions are not met, severity is 3.
3	Function or process is broken. Results or behavior of a function or process contradicts expected results (as documented in functional specification). Results or behavior of a function or process contradicts logically expected results. Minor documentation errors and inaccuracies. Text is misspelled.	If the following conditions are met, severity is 3: User has a simple workaround to mend situation. User can easily figure out workaround. Bug does not cause a bad user experience. Primary business requirements are still functional. Bug does not block a significant number of other test cases. If the preceding conditions are not met, severity is 2.
4	Suggestions. Future enhancements.	Clearly not a product bug for this version.

Triage Process

During the test phase of a migration project, triage meetings are held to review the bugs reported by the test team. Triage meetings are scheduled to occur regularly, usually three times a week during the first test pass and then daily as efforts approach the test completion deadline. Typically, the following people attend the triage meeting:

- Project manager
- Test manager
- Development manager

The test lead schedules and facilitates the triage meeting. Other team members may be brought to the meeting to discuss specific technical issues as needed.

The typical activities performed in a triage meeting and afterward are:

1. The test lead describes each newly recorded bug in the tracking system.
2. A common understanding of the problem is reached among members of the group.
3. The severity of each new bug is reviewed .Because severity is tied to the functionality delivered by the application, severity should not be redefined in triage meetings unless a group decision is also made to change the delivered functionality of the system.
4. Each bug is assigned a priority (high, medium or low) based on factors such as severity and developer availability.
5. Each bug and an action plan derived to facilitate resolution of the bug are assigned to the appropriate team member for resolution.
6. The test lead updates the tracking system based on the decisions made during the triage meeting.
7. Team members work on resolving the assigned bugs and update the resolution details in the bug-

tracking system.

8. Testers regress the resolved bugs. If the test was successful, the documents are updated according to the change control process and each bug is closed; otherwise, bugs are reactivated for further investigation.
9. The development team fixes the design bugs that the test team reported. After the development team fixes these bugs, they are assigned back to the test team members who opened them originally. The test team then retests the implementation and closes each bug as it is fixed. After all bugs are fixed, a newer build is created for retesting.

Generating a Test Report

When the test team completes the test cases, it prepares a test report to list the open bugs and to summarize system performance, system management findings and suggested practices. The bugs are listed according to priority. The test report also highlights the impact on business if bugs are not fixed, along with a recommendation on whether to proceed with production.

For more information about test reports, see "Reporting and Release Processes" later in this chapter.

Stage 5: Evaluate and Analyze Results

During release review, the development, test, and program management teams meet to discuss whether to release the implementation. If this review is successful, the product is released to production. Otherwise, the outstanding issues are reviewed and fixed.

In addition to reports that summarize and evaluate the test results, it is recommended that the release review include the following:

- Best practices for executing the test cases
- Root-cause analysis for the defects found in the testing process
- Identification and discussion of any areas in the migrated application that need further improvement or retesting

For more information about the release process, see "Reporting and Release Processes" later in this chapter.

Types of Testing

The test team performs the following types of tests, depending on the type of application to be migrated. It is recommended that the different types of testing be derived from the test plan of the application, if such a plan exists.

- **Security testing**

  Security testing is performed on the application to guarantee that only the users with the appropriate authority are able to use the applicable features of the application. The systems engineer establishes different security settings for each user in the test environment. Network security testing is performed to guarantee that the network is secure from unauthorized users. The test manager must consider the depth of hacking skill of the test staff and whether they would be able to create an adequate set of challenges to network security. Frequently, it is advisable to use outsourced specialists who can provide comprehensive security testing.

> **Note** In most companies, the systems engineer is a member of the development or product support team rather than a member of the test team.

- **Out-of-memory and memory-leaks testing**

  This testing ensures that the application will run in the amount of memory specified in the technical documentation. This testing also detects memory leaks associated with frequent starting and stopping of the application.

- **Performance testing**

  This testing is necessary for any enterprise-scale application, to provide an understanding of system capacity and the system's behavior under overloaded conditions. After the capacity of each server is known, the architect can make final decisions about how many of each type of server are needed to support the load. Understanding the system's behavior under abnormally high loads allows the architect to decide whether it is important to build the system to handle dramatic and temporary increases in traffic or whether administrative measures can be taken instead.

- **Scalability testing**

  Scalability is a measure of how easy it is to modify the application infrastructure and/or architecture to meet variances in utilization. These tests help to ensure that the infrastructure is scalable under varying conditions and is designed to meet growth needs.

- **Stability testing**

  This testing ensures that the migrated application is stable when exposed to prolonged load tests.

- **Management testing**

  This testing is performed if applicable to prove that the migrated application can be managed. The following procedures can be tested:
  - Monitoring of all the computers running Internet Information Services (IIS), Component Object Model (COM), the Microsoft Active Directory™ directory service, and Microsoft SQL Server™
  - Monitoring of the hardware information on various servers in the architecture
  - Monitoring of the CPU usage and memory usage
  - Monitoring of the authentications carried out on the servers running Active Directory
  - Monitoring of alerts when any of the stated thresholds is exceeded

The test team must decide on the level of testing that is required in each case or type of testing. Each type of testing can be classified according to the following levels of importance:

- **High**. A very important area that must be thoroughly tested.
- **Medium**. Standard testing is required.
- **Low**. Test if time allows.

How to Create a DTP and a DTC

This section describes the various plans and test cases that you should create when you test the migrated application.

A sample functional specification excerpt and the corresponding DTP and DTC follow. This information is based on the example provided in "Client-Area Mouse Message" in Chapter 11.

- DTP content:

  Scenario 1: Mouse click in the client area of a window.

    Left mouse button is clicked.

    Right mouse button is clicked.

  Scenario 2: Mouse click outside the client area of a window.

    Minimize button is clicked.

    Maximize button clicked.

    Close button is clicked.

- DTC content:

  Scenario 1: Left mouse button is clicked (Table 6).

Table 6. DTC scenario 1—left mouse button is clicked

Test case	Detailed procedure	Requirements	Expected result	Result
Left mouse button is clicked in the client area	Press the left mouse button. Move the mouse. Release the left mouse button.	Windows 2000, Microsoft Visual Studio®	Two lines are drawn. First line is drawn to the point where the button was pressed. Second line is drawn to the point where the button was released.	

Reporting and Release Processes

This section discusses the last two phases in the overall testing process: reporting and release to production.

Reporting the Results

After the test team completes a successful execution of the required test cases, a test report is prepared to list the remaining open bugs. This final test report is called the "Release to Production" report. This report describes the areas that were tested and summarizes the testing results. It also lists bugs or issues that were postponed to future versions. The report also highlights the impact on business if the remaining bugs are not fixed, along with a recommendation on whether to proceed with production. The completed report is presented to the project management team to help that team evaluate whether the product can be released to production or whether the product must be fixed.

Release to Production

After the project management team receives the Release to Production report, the teams involved in the migration project (development team, test team, and program management team) meet to discuss whether to release the product to production.

If the teams agree that the product meets all the requirements of the project, it can be released to production. If not, the outstanding issues are reviewed and fixed as part of the standard triage and fix process described earlier.

The following sections discuss the different criteria that the teams should take into account when deciding whether the implementation is ready for release.

Test pass/fail criteria

A test case passes if the actual result matches the expected result. If the actual result does not match the expected result, it is treated as a failed test case.

If a test case fails, it is not assumed that the feature is defective. For example, misinterpretation of project documentation, incomplete documentation, or inaccurate documentation can cause failures. Each failure is analyzed to discover its cause, based on actual results and the results described in project documentation.

Pass criteria are as follows:

- All processes run with no unexpected errors.
- All processes finish in an acceptable amount of time, based on benchmarks defined in the functional specification.
- Load tests show that a satisfactory level of capacity is in place and that appropriate steps can be taken to scale out the system when necessary.

Test suspension criteria for failed BAT and test execution

The test team may suspend partial or full testing activities on a given build if any of the following events occurs:

- A systems engineer is unable to install the new build or a component.
- A systems engineer is unable to configure the build or a component.
- A major feature has a fault that prevents a significant area from being tested.
- The test environment is not stable enough to trust the test results.
- The test environment is very different from the expected production environment, so test results cannot be trusted.

Resumption requirements

The test team may resume testing if:

- The problem that suspended the testing is corrected.
- The development and test teams agree that fixing the bug can be postponed until the next iteration.

Release criteria

The release criteria depend on the severity of bugs that are open. As noted earlier, these criteria are measured on a scale of 1 to 4, where level 1 represents the highest severity, and level 4 represents the lowest severity.

The following are some possible criteria for allowing the code to be released to production:

- There are no open bugs with a severity of 1 or 2.
- Test cases scheduled for both integration and system test phases have successfully passed (especially test cases that cover major functionality or that were given a high priority or a medium priority).
- The final regression test has successfully passed.

  Note The criteria for release vary widely, depending on both the architecture and the nature of the migrated application.

Appendixes

This section provides information about the appendix files that are described in this chapter. Please click this link for all the templates

Appendix A: Test Schedule Template

The Test Schedule Template.mpp appendix file provides an overall testing schedule and is created based on the tests planned and the effort required. The file is a Microsoft Project template that you can modify to suit your own project requirements.

Appendix B: Risks Identified for Testing

The Risks Identified for Testing.doc appendix file is a template that includes examples of risks that may be identified when a migrated application is tested.

Appendix C: Master Test Plan Template

The Master Test Plan Template.doc appendix file provides an MTP template that you can customize for your migration project.

Appendix D: Detailed Test Plan Template

The Detailed Test Plan Template.doc appendix file provides a DTP template that you can use to create DTPs that cover the major areas involved in testing your migrated application. It is possible to merge the contents of the DTPs into the MTP template provided in Appendix C to make a single document, if required.

Appendix E: Detailed Test Cases Template

The Detailed Test Cases Template.doc appendix file provides a template that you can customize to document the test cases used in testing your migration project.

References

For further information, see the following:

- Infosys Software Engineering methodologies
- How to Successfully Migrate UNIX Applications to Windows 2000 Professional

patterns & practices
proven practices for predictable results

Send feedback to Microsoft