CHAPTER 1

Introduction

SAP testing is complex, difficult, and esoteric. The perils and risks to the intended SAP production system are maximized when the project team does not have enough skilled testers and a robust approach for testing the system, tracing the entire system design and architecture to testable requirements, and eliminating defects. A comprehensive plan or approach for testing an SAP system even for a small project includes assembling a test team, acquisition of test tools, establishment of a test lab, construction of a test calendar, monitoring of testing activities, training for testing participants, and completion of a test cycle based on predefined criteria.

A test plan for SAP includes the approach; description; roles and responsibility for conducting usability testing; white box and black box testing for interfaces; negative testing; security testing for roles, integration, scenario, unit, and performance; user acceptance testing; and regression testing. Few companies implementing SAP are structured or organized to address all these various types of testing.

This book offers guidance and assistance to project managers, test managers, and configuration leaders who want to establish a testing framework from the ground up based on industry-established principles for evaluating requirements, providing coverage for requirements, diagramming processes, test planning, estimating testing budgets and resource allocation, establishing an automation framework, and reviewing case studies from large SAP implementations.

WHY THIS BOOK?

SAP is by far the world's largest enterprise resource planning (ERP) application, and this position is not likely to be relinquished anytime soon. SAP traces its origins to its mainframe-based R/2 version from

the 1970s. Even though SAP and other vendors have developed implementation methodologies for SAP that provide emphasis for activities needed to design the system and for going live, many of these methodologies fall short of addressing robust testing practices for SAP in particular in light of the prominence of automated test tools and outsourcing testing activities.

The roles and activities for SAP configuration, for SAP advanced business application programming (ABAP) development, and SAP Basis are for the most part well understood and clearly defined. In contrast, when it comes to SAP testing, the roles and activities are a mystery and subject to interpretation. It is possible for a person who is entertaining becoming an SAP consultant to take courses on how to configure a particular SAP module, how to build security roles, or how to develop ABAP programming code but not how to test SAP R/3. Functional testing of SAP is often left to individuals without a testing background whose main tasks are to configure the system. Other types of SAP testing such as technical and system testing for system performance and backup and recovery are also left to individuals without a testing background whose main responsibilities are to design and maintain the technical architecture for SAP.

This book was written to help SAP projects address weaknesses in the SAP testing life cycle, define testing and quality assurance activities, and overcome misconceptions about SAP testing. The book contains contributions from industry leaders in the fields of SAP testing, test tool automation, and templates to help project managers and test managers establish immediate testing best practices. It covers all aspects of SAP testing from preparation to resolution of defects.

WHAT DOES THIS BOOK OFFER ME?

This book is written from the point of view of the company or entity requesting SAP services from a systems integrator. The book emphasizes testing practices predicated on the following principles:

- Building a system with quality as opposed to merely testing for quality.
- Adhering to testing practices from SAP's ASAP Roadmap Methodology, IBM's Ascendant™ methodology and guide for imple-

menting SAP, Institute of Electrical and Electronics (IEEE) standards, and the Capability Maturity Model (CMM) from the Software Engineering Institute (SEI).
- Drafting of clearly stated requirements that can be validated with test cases.
- Construction of a requirements traceability matrix (RTM) to verify all in-scope requirements.
- Supporting each testing effort with an exit, entrance, or suspension criterion.
- Validation of production support changes through thorough regression testing.
- Subjecting all test results to third-party verification and approval (sign-offs) from appropriate project stakeholders.
- Diagramming processes and requirements with Unified Modeling Language (UML) notation.
- Documentation and adherence to test plans and test strategies that are subjected to version control.
- Early formation of a test team that participates in design workshops during the blueprint phase, change control boards meetings, and the "go/no go" decision.
- Inclusion and enforcement of quality assurance (QA) standards.
- Peer reviews and inspections for testable requirements.
- Independent verification and validation of system design.
- Independent "hands-on" testing with participation from end users who execute test cases.
- Functional testing with manual testing and automated test tools.
- Maintaining testing deliverables such as test cases, test results, and testing defects in a test management tool that includes security features and audit trails.
- Compliance with company and industry audits.

The book is a primer for testing SAP from unit testing through regression testing for production support. The intended audience for the book includes test managers, project managers, integration leaders, test team members, QA personnel, and auditing groups. The book provides templates, case studies, and criteria to establish a framework for SAP testing, establish a test case automation strategy, mentor junior testers, and identify tasks and activities that the SAP system integrator is expected to accomplish for the client requesting

SAP services. Specifically, this book will address the following activities that are prevalent yet poorly conducted at most SAP projects:

- Identifying the testable requirements.
- How to ensure that requirements are testable, unambiguous, clearly defined, necessary, prioritized, and consistent with corporate policies or industry standards.
- How to retain testing documentation to support audits (i.e., Section 404 from Sarbanes-Oxley).
- Defining the scope of testing.
- Creating a test plan and test strategy.
- Developing a strategy for acquiring automated test tools and for automating processes that includes verification at the graphical user interface (GUI) and back-end layers.
- How to create a library of automated test cases for production regression testing that can be executed unattended.
- How to define and verify service-level agreements (SLAs).
- Boundary testing for negative and positive testing.
- How to apply quality standards from the SEI and Rational Unifying Process (RUP).
- Creating robust flow process diagrams that include narratives.
- Techniques for estimating the testing schedule, duration for testing activities, and number of testers needed.
- How to compress or reduce the necessary number of test cases with the technique of orthogonal arrays (OATS) for projects implementing SAP variant configuration or projects that have multiple variations for end-to-end processes and are time constrained.
- Defining criteria for deciding which processes or scenarios are suitable for testing.
- How to estimate testing costs and budget.
- Creating an RTM to ensure that coverage has been provided for all types of testable requirements (i.e., functional, security, performance, archiving, technical, and development requirements).
- Creating a test schedule and a test calendar.
- Defining objective criteria to commence, finish, and stop testing.
- Managing, categorizing, and resolving test results.

CHALLENGES IN SAP TESTING

Established commercial implementation methodologies for SAP typically fail to address how requirements will be met, the criteria for testing, the framework for utilizing test tools, necessary resources for testing, estimating testing budgets, specific testing roles and responsibilities, and how test defects will be managed and resolved. Furthermore, many factors hamper successful testing at most SAP projects such as unclear requirements, inability to trace the system design to requirements, missing dedicated test teams, waiving defects without appropriate workarounds, and inadequate involvement of needed participants for testing such as subject matter experts (SMEs) for capturing requirements and end users for user acceptance testing. Despite these testing challenges, many SAP project managers perceive that their SAP implementation is successful or "fine" even when the production help desk team is flooded with complaints that the system does not perform necessary functionality, the production system does not meet intended performance SLAs, security roles are not defined and implemented correctly, the system produces short dumps because it cannot perform exception handling or not enough negative testing was conducted, data is not converted properly from legacy systems, and end users cannot find even the most basic data or necessary reports.

The SAP arena is replete with functional, development, and technical consultants that moonlight and parade as SAP testers for various testing efforts but often lack sufficient knowledge to establish a successful testing strategy and framework. What is more puzzling and baffling at SAP projects is that it is the individuals with the least amount of knowledge and skills in the area of testing who are the ones in charge of leading and managing the testing effort since many SAP projects do not have dedicated test managers or centralized test teams. Admittedly, testing at any SAP project is an integrated effort that requires the expertise and skills of several resources such as SMEs, functional configuration resources, ABAP developers, and business analysts. Yet executing testing activities without the guidance and help of testing professionals is analogous to taking a trip without knowing what the final destination will be.

Frequently, an individual moonlighting as an SAP tester will state that "testing is breaking or exploring the system" or that he "knows how to test," which undoubtedly leads to a misconception about what SAP testing is really all about. The truth is that many companies fail to adequately test an SAP system and rather deploy the system into production because testing is taking too long, which consequently forces the production support team to fix the system for the first six to eight months after the system is deployed because it was never properly tested prior to its release into the production environment. Whether by accident or on purpose, often the modus operandus for many corporations is to deploy an unstable and/or poorly tested SAP system into production because defects and system problems can be dealt with at a later date in the production environment, even while there is substantial and empirical data that demonstrates that removing system defects is least expensive when done in the early stages of testing.

Industry data shows that removing system defects in a live production environment is at least 20 to 40 times more expensive than doing so in the unit-testing phase or during the requirements-gathering phase. Many defects can be eliminated or prevented altogether with thorough evaluation and peer review of requirements. Many corporations pay expensive consulting fees to fix production problems arriving at the production help desk rather than address these problems or defects during the applicable testing phase. The main reason that this occurs is that SAP projects often do not spend the time or have the appropriate resources to ensure that the captured requirements are peer reviewed and evaluated with objective criteria, or to construct an RTM to provide coverage for all requirements and establish objective testing criteria for each testing phase. Another critical or overlooked reason that causes defects that should have been resolved during testing to slip into the production environment is that individuals acting as SAP testers cannot reach consensus on testing nomenclature or the test approach.

The mere term *testing* in the SAP world is in and of itself broad enough to create ambiguity, since different individuals will have different perceptions and experiences about what testing means. Testing encompasses many activities such as requirements gathering and traceability, test planning, test design, test execution, test reporting,

test results, and resolution of defects to cover a wide range of testing efforts such as unit, boundary, scenario, development, white/black box, security, smoke, integration, performance, user acceptance, and regression testing. Rarely, if ever, do two or more individuals from the configuration, development, or technical teams have the same nomenclature or understanding for a particular type of test. Chaos and inconsistency are the ensuing results from misunderstanding about what the term *testing* entails or what activities are associated with testing. Dedicated test teams can establish consistency for all testing terms for all project members based on established guidelines and nomenclature from credible sources such as the Software Engineering Institute (SEI), IEEE standards, and the SAP ASAP Roadmap methodology.

A common theme repeated at many SAP projects is that conclusive evidence is missing to show that requirements have been met before releasing the system into the production environment. Most project managers or functional managers cannot answer with any degree of confidence or objectively whether the in-scope requirements captured during the requirements phase have been met before releasing or deploying a system. This occurs because the concept of an RTM is not embedded within most, if not all, of the mainstream or conventional methodologies for implementing SAP for either initial SAP implementations or SAP upgrades.

Test tools pose a challenge for many SAP projects. SAP tools hold the promise of unattended test case playback at any time, increased testing coverage, testing of processes with multiple data and process variations, verification of objects and calculations, and generation of automated test logs with time stamps for audit purposes and compliance. Many SAP system integrators and test tool vendors are adept at convincing companies to spend hundreds of thousands of dollars in acquiring automated test tools and test tool training only to have the test tools gather dust. Test tools can sit idle because the company acquiring the test tools is missing an automation framework and thus cannot successfully engage the appropriate resources to maintain, install, and utilize the test tools. The payback period or return on investment (ROI) for test tools is not maximized or even reached until a series or library of automated test cases can be constructed and reused frequently for future system releases or to support production

regression testing to the point where automated execution is cheaper than doing the same tasks with manual labor hours. Constructing a library of automated test cases is rarely achieved even by companies that have had SAP in the production environment for years because they do not allocate the necessary skilled resources to maintaining and utilizing the test tools. Companies that commit hundreds of thousands of dollars to the acquisition of test tools that sit idle compromise their testing budgets. Consequently, these companies resort to testing SAP exclusively with manual testers.

Another common challenge for testing SAP is inadequate training at all levels for either cross-matrixed testing resources or dedicated testing resources. Training is needed for testers who are participating in one-time testing efforts such as user acceptance testing, or participating in all testing efforts for execution of test cases and resolution of defects. The test manager needs to develop the procedures for mentoring and educating all project resources who are expected to participate in testing activities. Training consists of the following activities:

- Training dedicated testers on how to maintain and install automated test tools, test management tools, and develop automated test cases.
- Training testing participants on test procedures for logging defects and reporting test results.
- Training on how to evaluate and peer review requirements.
- Training on testing nomenclature to standardize testing terms for all project members.
- Training for roles and responsibilities for resolving defects.

The challenges mentioned above are some of the most prevalent problems and issues that permeate most SAP projects. By no means are these the only challenges present at SAP projects. Many SAP projects suffer from poor documentation and configuration management for testing deliverables or work products, and inability to successfully meet audits or design a solution that is in compliance with industry regulations and requirements. The aforementioned challenges are used as illustrations that highlight the need to establish robust testing techniques, methodologies, strategies, and frameworks.

EARLY TESTING MATTERS

It is never too early to implement and establish the testing program. In fact, for readers familiar with the SAP ASAP Roadmap methodology, Exhibit 1.1 shows that testing strategies are defined as early as the project preparation phase. For readers familiar with the different software development life cycles including the waterfall model, the software industry has developed a similar model known as the V-shaped model that emphasizes testing as a consideration throughout the development life cycle. Furthermore, industry standards suggest and manifest that testing early helps to decrease costs since identifying and resolving defects early on during the initial software development life cycle is much more economical than troubleshooting and resolving defects once the system has been deployed into the production environment.

Testing early and often is instrumental to reducing development costs, ensuring fulfillment of in-scope requirements, and aligning with the project's scope statement. The most effective testing programs start at the beginning of a project, long before any program code has been written. The requirements documentation is verified first; then, in the later stages of the project, testing can concentrate on ensuring the quality of the application code. Expensive reworking is minimized by eliminating requirements-related defects early in the project's life, prior to detailed design or coding work.

The requirements specifications for a software application or system must ultimately describe its functionality in great detail. Typically in SAP, requirements for initial implementations are captured during the blueprint phase with workshops where various stakeholders state what they expect SAP to accomplish for them or for existing SAP implementations that are undergoing major upgrades or implementing previously deferred requirements. One of the most challenging aspects of requirements development is communicating with the people who are supplying the requirements. Each requirement should be stated precisely and clearly, so it can be understood in the same way by everyone who reads it.

If there is a consistent way of documenting requirements, it is possible for the stakeholders responsible for requirements gathering to effectively participate in the requirements process. As soon as a requirement is made visible, it can be *tested* and clarified by asking the

EXHIBIT 1.1 Testing and Quality Assurance Activities for an Initial SAP Implementation

Phase	Project Preparation	Blueprint	Realization	Final Preparation	Go-Live Support
Activities	• Define testing strategies	• *Present QA standards and processes* • *Review requirements* • *Attend workshops for gathering requirements* • *Conduct requirements peer reviews* • *Enforce QA processes* • *Assemble test team* • *Set up test lab* • *Procure test tools* • *Test tool training for functional users* • *Customize test tools*	• Define Baseline Test Cases • Create test plan for baseline • *Kick-off presentation* • Test baseline • Define final scope test cases • Create test plan for final scope • Test final scope • Conduct development testing • Conduct integration testing • Prepare for system testing • *UAT preparation and execution*	• Conduct System Testing	• Continuous Improvement • *Define regression testing strategy* • *Define change control processes* • *Automate test scripts* • *Modify existing test scripts* • *Participate in CCB meetings* • *Execute test cases* • *Document test findings* • *Support test tools* • *Support SOX*
Deliverables	• Testing strategy paper • Automation standards • *Quality Assurance Plan* • ***Established Project Methodologies, Tools and Governance Standards***	• *Create Checklists for evaluating specifications and diagrams* • *Establish Test lab with installed test tools* • *Construct RTM (requirements traceability matrix)* • Test Plan and Criteria	• Baseline test case • *Test Readiness Review* • *Test Cases* • *Test Results* • *Test report* • *Developed automated scripts* • *Execution calendar* • *Lessons Learned*	• *Execute stress/load/volume/performance testing* • *Generate and produce system testing scripts* • *Gather and interpret system testing results* • *Test report*	• *Automation framework* • *Regression testing paper* • *Refined library of Automated test scripts*
Tools	• Testing Strategy White papers	• *Functional Design Specs* • *Technical Design Specs* • *Flow process Diagrams*	• BPPs with test conditions • Test scenario template • BPML • Stress testing strategy • Stress/Volume testing sample plans	• *Loadrunner*	• *Quicktest Pro* • *Kintana*

Note: During realization phase, 50% or more of all project costs are dedicated to testing activities.

stakeholders detailed questions. Whether your team develops requirements using UML and some form of use case or writing "the system shall" statements, a variety of *requirement tests* can be applied to ensure that each requirement is relevant, and that everyone has the same understanding of its meaning. UML is a widely accepted technique for requirements gathering or reverse engineering an existing system, and its notation is composed of multiple diagramming techniques such as use-case notation, activity, class and sequence diagrams, and so on.

In order to introduce the concept of "test early and test often," it is important to recognize the following two items: (1) involve testers from the beginning and (2) verify the requirements.

Involve Testers from the Beginning[1]

Testers need to be involved from the beginning of a project's life cycle so they can understand exactly what they are testing and can work with other stakeholders to create testable requirements.

Not only can testers verify testability of the requirement, but they will also learn the thought process that went into the requirement as it applies to the application under test (AUT), making the tester more knowledgeable about the AUT.

A defect occurs when an executed test case produces test results that do not match the expected test results. *Defect prevention* is the use of techniques and processes that can help detect and avoid errors before they propagate to later development phases. Defect prevention is most effective during the requirements phase, when the impact of a change required to fix a defect is low. The only modifications will be to requirements documentation and possibly to the testing plan, also being developed during this phase. If testers (along with other stakeholders) are involved from the beginning of the development life cycle, they can help recognize omissions, discrepancies, ambiguities, and other problems that may affect the project requirements' testability, correctness, and other qualities.

[1]Adapted from Elfriede Dustin, *Effective Software Testing*, Reading, MA: Addison Wesley, 2002.

A requirement can be considered *testable* if it is possible to design a procedure in which the functionality being tested can be executed, the expected output is known, and the output can be programmatically or visually verified.

Testers need a solid understanding of the product so they can devise better and more complete test plans, designs, procedures, and cases. Early test-team involvement can eliminate confusion about functional behavior later in the project life cycle. In addition, early involvement allows the test team to learn over time which aspects of the application are the most critical to the end user and which are the highest-risk elements. This knowledge enables testers to focus on the most important parts of the application first, avoiding overtesting rarely used areas and undertesting the more important ones.

Some organizations regard testers strictly as consumers of the requirements and other software development work products, requiring them to learn the application and domain as software builds are delivered to the testers, instead of involving them during the earlier phases. This may be acceptable in smaller projects, but in complex environments it is not realistic to expect testers to find all significant defects if their first exposure to the application is after it has already been through requirements, analysis, design, and some software implementation. More than just understanding the "inputs and outputs" of the software, testers need deeper knowledge that can come only from understanding the thought process used during the specification of product functionality. Such understanding not only increases the quality and depth of the test procedures developed, but also allows testers to provide feedback regarding the requirements.

Verify the Requirements

In his work on specifying the requirements for buildings, Christopher Alexander describes setting up a *quality measure* for each requirement: "The idea is for each requirement to have a quality measure that makes it possible to divide all solutions to the requirement into two classes: those for which we agree that they fit the requirement and those for which we agree that they do not fit the requirement." In other words, if a quality measure is specified for a requirement, any

solution that meets this measure will be acceptable, and any solution that does not meet the measure will not be acceptable. Quality measures are used to test the new system against the requirements.

Attempting to define the quality measure for a requirement helps to eliminate requirements not suitable for implementation and thus testing. For example, everyone would agree with a statement like "the system must provide good value," but each person may have a different interpretation of "good value." In devising the scale that must be used to measure "good value," it will become necessary to identify what that term means. Sometimes requiring the stakeholders to think about a requirement in this way will lead to defining an agreed-upon quality measure. In other cases, there may be no agreement on a quality measure. One solution would be to replace one vague requirement with several unambiguous requirements, each with its own quality measure.

It is important that guidelines for requirement development and documentation be defined at the outset of the project. In all but the smallest programs, careful analysis is required to ensure that the system is developed properly. Use cases from UML notation are one way to document functional requirements, and can lead to more thorough system designs and test procedures. (Here, the broad term *requirement* will be used to denote any type of specification, whether a use case or another type of description of functional aspects of the system.)

In addition to functional requirements, it is also important to consider nonfunctional requirements, such as performance and security, early in the process. They can determine the technology choices and areas of risk. Nonfunctional requirements do not endow the system with any specific functions, but rather constrain or further define how the system will perform any given function. Functional requirements should be specified along with their associated nonfunctional requirements.

Chapter 3 offers a checklist that can be used by testers during the requirements phase to verify the quality of the requirements. Using this checklist is a first step toward trapping requirements-related defects as early as possible, so they don't propagate to subsequent phases, where they would be more difficult and expensive to find and correct.

TYPES OF SAP TESTS

Traditionally, the main types of SAP tests include unit, development, scenario, integration, performance, and regression testing. These tests are further described below to provide greater granularity into what each type of test entails.

Unit Testing

This is the lowest level of testing at the SAP transaction level. Unit testing includes boundary testing for positive and negative testing. Negative testing should be performed for custom fields and transactions to ensure that the system only allows valid input and can adequately perform exception handling. An example of a negative test for a process would be attempting to process an order with the wrong status.

Unit testing includes testing security roles. The configuration team owns the unit-testing effort and is responsible for planning and execution of unit testing. The main focuses for unit testing are:

- Master data
- Negative-positive testing
- Transaction functionality
- Security roles and profiles

Negative testing is performed on security roles and profiles, custom fields, objects, and processes. Each test in negative testing needs two elements:

1. Intentionally specify conditions that will cause the software to generate an error.
2. Ensure that the generated error is handled in a specified manner.

An example of a negative test condition would be "Attempting to post a material to an invalid profit center should produce an error message." Another negative testing example for security roles and segregation of duties would be "An inventory clerk attempts to ap-

prove a million-dollar purchase order when he is only permitted to approve purchase orders for a maximum of $500,000."

Negative testing will be designed to address the following situations:

- Check exception handling and error message.
- Prove that the system will deal with program exceptions and erroneous data.
- Limit or prevent an end user from trying to do something he should not.
- Demonstrate that the system does not do anything that it is not supposed to do.
- Users are permitted to perform only actions based on their authorizations, position roles, and permissions.

Development Testing

This is the testing for reports, interfaces, conversions, enhancements, work flows, and forms (RICEWF) development objects developed primarily with ABAP code. Testing of development objects includes testing for security authorizations, performance, extracts, data transfer rules, reconciliations, and batch scheduling jobs. In many SAP projects, third-party tools such as Control-M and AutoSys are acquired to schedule reports and interfaces with dependencies, and these scheduled jobs need to be tested prior to releasing the system into the production environment. Development testing should also ensure that data can be tested through the intended target system. The owner(s) of the target system can specify the applicable or representative sets of data needed to test interfaces and conversions, which allows the development or ABAP team to conduct white box and black box testing on ABAP programs.

The development or ABAP team is responsible for planning and executing the development tests, but the configuration team is responsible for approving the results for the development tests.

Development testing ensures that the interfaced data originating from legacy systems can be effectively transferred into SAP or sent from SAP into a legacy system. In order to design test cases for

RICEWF objects, technical specifications that can contain pseudo-code will need to be developed. The development test cases need to reflect the testable conditions from the technical specifications.

Business Warehouse (BW) testing is also part of the development tests. BW testing includes testing the infocubes, queries, reports, and multicubes. The main types of tests for BW testing are:

- **Reconciliations.** Are financial calculations rolling up correctly?
- **Extracts.** Is there a match between the number of extracted records and the number of received records?
- **Performance.** How fast can a query be performed, and does it conform to established performance SLAs?
- **Security.** Who is permitted to slice and dice the data in the Bex Analyzer? What are the established roles for generating queries?
- **Data transfer rules.** Is data transformed correctly for all fields from the source system to the target system?

Scenario Testing

The equivalent of a string test, scenario testing is the testing of chains of SAP transactions that make up a process within a single area or module. Scenario testing includes testing of a process with data from external systems and applicable SAP roles/profiles.

Scenario testing is primarily a manual effort but can include some partial automation with test tools for processes that are stable, frozen, and proven to have worked manually. The scenario testing is owned by the configuration teams but includes participation from SMEs and members of the test team and development team.

Integration Testing

Integration testing is the testing of chains of SAP transactions that make up an end-to-end process that cuts across multiple modules, for instance, order-to-cash, purchase-to-pay, and hire-to-retire with external data and converted data. Integration testing includes testing through the external systems and SAP bolt-ons with security roles and workflow. Integration testing consists of multiple iterations. The ded-

icated test team is the owner of the integration test. Integration testing requires participation from members of the configuration and development teams for defect resolution. Additionally, SMEs and end users participate in the integration test as reviewers and for approval of test results.

Integration testing is mostly a manual effort but can include some partial automation with test tools.

Performance Testing

Performance testing encompasses load, volume, and stress testing to determine system bottlenecks and degradation points. A performance test helps to determine the optimal system settings to meet and fulfill the established SLAs.

The dedicated test team is the owner of the performance test. Performance tests are conducted primarily with automated test tools. In theory it is possible to conduct performance testing with manual test cases, but this proves highly impractical since it is not easily repeatable and requires both human and hardware resources that are often not available. A performance test, even if automated, can still include manual execution of interfaces, batch jobs, and external processes that send data into SAP.

The basis, database, and infrastructure teams help monitor the performance test, whereas the configuration team helps to identify test data and document test cases that are suitable for the performance test.

User Acceptance Testing

User acceptance testing allows the system's end users to independently execute test cases from the perspective of how the end users plan to perform tasks in the production environment. The owners of the user acceptance testing are the end users, and the configuration and test team members resolve defects identified during the user acceptance test. The test team and change management team members help train end users and prepare them for the user acceptance test.

Regression Testing

Regression testing ensures that previously working system functionality is not adversely affected by the introduction of new system changes. System changes targeted for the production environment need to be analyzed for impact and cascading effects on other processes. Since SAP R/3 is an integrated system, a single system change—whether it is a hotpack, an OSS note, or a transport to resolve a defect—can have far-reaching consequences for other processes, and thus regression testing is needed to ensure that "nothing is broken" as a result of a new system change. Regression testing is primarily an automated testing effort. For regression testing, a library of automated test cases is constructed and played back to ensure that system transports do not break or alter system functionality.

The test team owns the execution of the regression test. Determining the impact of a system change is primarily the responsibility of the integration team and change control board (CCB).

Other types of SAP tests include usability, archiving, data migration testing, and technical tests. Usability testing is discussed in Appendix A. Technical tests such as backup and recovery, printing, faxing, electronic data interchange (EDI), availability, and so on are also needed in particular for initial SAP implementations and/or global SAP rollouts. The concept of technical testing is beyond the scope of this book.

Data migration testing for established SAP implementation refers to SAP projects that have global SAP rollouts or multiple business units and want to introduce SAP to other company divisions or business segments. For example, a company may have designed the order-to-cash business process within SAP for one division and may have plans to extend the same or slightly modified version of the order-to-cash business process to a different division that has different data values, and thus the new data values need to be tested.

Depending on contractual, scope of the project, project's oversight, or industry regulations, the SAP tests described above may need to be either very formal and structured or casual. The tests described will at a minimum require identification of valid test data, rewriting of test cases, or creation of new test cases; manual testing; peer reviews; and approvals at the end of each testing cycle.