Best Practices:

Enterprise Testing Fundamentals

Testing is a critical aspect of making sound enterprise deployment decisions. Given a large field of possible test scenarios, administrators must decide when enough testing has occurred based on indicators of readiness for the production environment. This article, addressing fundamentals of the test process, explores best practices for enterprise testing—including recommendations for a phased approach to component, feature, and system level testing.

BY CYNTHIA LOVIN AND TONY YAPTANGCO

Related Categories:

Application development

Enterprise management

Enterprise testing

Performance

Planning

Project management

Scalable enterprise

System deployment

Testing

Visit www.dell.com/powersolutions for the complete category index.

Testing enterprise solutions can be challenging, particularly when determining the right mix of approaches to apply within tight cost and schedule constraints. Effective testing can enable sound decisions about production readiness—and ineffective testing can be costly, either because of excessive testing or undetected defects that can lead to downtime and support issues. This article focuses on testing fundamentals that are key to effective enterprise testing.

The benefits of testing

To yield high-quality products, enterprises could test every feature of a product on every possible configuration using every approach or sequence that a customer might use. Of course, such an exhaustive process would take so long that the product might be obsolete before it ever reaches customers.

The key is to create a scientific, structured approach to testing—an approach that will find and fix the highest number of defects in the shortest possible time at the most reasonable cost. In short, the goal is to optimize test activities for a fast time to market while delivering a product that meets quality expectations.

An introduction to test processes

In product development, the initial view of a product is a broad overview of what a customer needs, usually in the form of product requirements generated by an enterprise's marketing team. Product developers then begin an iterative process of defining the required design. These preliminary designs are refined into detailed plans. Product modules or components are then defined and coded by the respective hardware or software engineers.

The test process can be viewed as reverse engineering of the product-development process. It is usually a three-step process consisting of component or module verification, feature verification, and usage validation.

Best practices for unit testing

The process of component or module verification is referred to as unit or structural testing. Unit testing requires knowledge of the code or circuitry—this type of testing is known as "glass box" or "white box" testing. Typically, it is performed by members of the development team.

Unit testing software and firmware

Unit testing can occur as soon as a respective component has been developed. Because this is a test of the

component only, it is not necessary for the rest of the product to be available. Therefore, unit tests can occur in parallel with the design and coding of other components. As new, untested routines are added to the component and defects are found, developers can isolate the defects and attribute them to the new routine or to its interactions with what has already been tested.[1] Unit test cases should be written in advance of, or alongside, the code modules as they are developed.

Unit testing may be used to verify that requirements and design have been implemented as well as to test data flow, checking all logic paths and error handling.[2] Unit testing is also used to augment code and design reviews on critical portions of their products.

Unit testing hardware

Unit testing for hardware is designed to find low-level flaws in chips, subassemblies, and interfaces. For example, the chip specifications may be compared to readings taken with oscilloscopes or voltage meters.[3] Hardware unit tests are performed on single components before related system components are ready to be added and verified.

Best practices for product testing

As components complete unit testing, the product reaches readiness for feature verification. This testing phase is referred to as product or integration testing. Product testing is an in-depth verification of the integrated product that includes all options of its respective features. Test cases determine whether the product's designed and implemented features and functionality meet requirements and specifications. Product testing is performed on hardware, firmware, and software and may use "glass box" or "black box" testing or both. Unlike "glass box" testing, "black box" testing does not require knowledge of the code structure or circuitry.

Product testing should be performed by an independent test team (not members of the development team) and can begin when enough interrelated, unit-tested components are available to allow test progress in one or more major areas of functionality. Product testing introduces additional complexity—for example, when customer requirements call for a software application to be supported on various server, storage, and OS platforms. Everything that a customer can do should be tested during this phase. Test engineers also should anticipate how the product might be used improperly to help ensure the product handles these types of error conditions gracefully.

		Type of test and when to use it		
Technique	**Description**	**Unit**	**Product**	**System**
Alpha testing	Allows carefully selected customers to test the partially verified product in real-world environments; provides early design feedback	Yes	Yes, early	No
Boundary testing	Attempts to overflow data structures, field lengths, and other boundary conditions	Yes, internal boundaries	Yes, all boundaries at user interface	Yes, final verification
Negative testing	Tests the product in unsupported circumstances or environments to help ensure that the user is given clear directions and that the product exits gracefully	Yes	Yes	Yes
Automated testing	Does not require human execution—a test harness or script runs unattended (useful for regression and boundary testing)	Yes	Yes	Yes
Stress and load testing	Verifies correct product operation under stress and load conditions; verifies robustness of design	No	Yes, feature level	Yes, system level
Performance testing	Verifies that product performance meets requirements, such as system boot time	No	Yes	Yes
Usability testing	Provides structured testing for an intuitive, clear user interface; may be performed with internal testers or a sample of external users similar to potential customers; should be performed as early as possible because it may drive design changes	No	Yes	Yes
Beta testing	Allows carefully selected customers to test the functionally verified product in real-world environments; validates a successful design in an almost-finished product	No	No	Yes
Reliability testing	Tests acceptable product operation over time without failure (mean time between failures) or to minimize downtime (mean time to repair); can be particularly useful for hardware testing	No	No	Yes
Acceptance testing	Uses testing performed by customers to determine product's readiness to deploy in the production environment	No	No	Yes, before release to customer
Out-of-box audits	Performs sample testing on the boxed product from the factory; may uncover packaging, manufacturing, or transportation handling flaws	No	No	Yes, during or after release to customer

Figure 1. Test types, descriptions, and phases

Ideally, unit testing and reviews in the earlier test phases will have uncovered the majority of component defects, and product testing will find defects primarily in the interfaces among these components. Each component should work well individually as demonstrated by the unit tests, but when put together, they may fail. If unit test resources or automated unit tests are available, they may continue to be used in parallel with product testing.

Best practices for system testing

System testing, the third level of testing, is a validation of the product's required usage after installation. During this phase, the test team attempts to replicate the users' operating environments.

[1] Source: *Code Complete* by Steve McConnell, published by Microsoft Press, 1993.

[2] For a "logic coverage" approach using minimal test cases, see "Structured Basis Testing" in *Code Complete* by Steve McConnell, published by Microsoft Press, 1993.

[3] Source: *Managing the Testing Process* by Rex Black, published by Wiley Publishing, 2002.

System testing occurs when all of the product features and functionality have been fully implemented. System test execution requires all components to be available in relatively stable, customer-ready revisions.

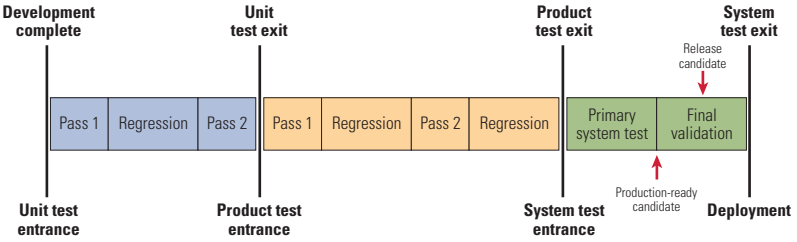

Figure 2. Phases of a typical test model

This phase often includes operating the product in a variety of environments to uncover defects. Variables that may be introduced include different communication networks and other products that will interoperate with the product under test. Products are validated under stress and load conditions. Reliability and other long-duration testing or test cases that have historically uncovered design flaws should be started as soon as possible in the system test phase (see Figure 1). This allows enough time for development teams and suppliers to respond with fixes and for the test team to verify the fixes, while preserving the desired project schedule.

System testing should help determine production readiness. To the extent possible, customer acceptance test techniques must be integrated into system testing. This can facilitate the acceptance process at the customers' sites. If unit and product test resources or automated regression tests are available, they should continue to be run in parallel with system testing.

Repetition and regression testing

A primary purpose of any testing is to uncover defects in the product and to characterize the circumstances that allow these defects to surface. As defects are discovered and reported, development engineers analyze the defects, develop fixes for problems, and provide those fixes back to the test team for verification.

Test engineering strategy must incorporate some repetition of core functionality testing during defect fix verification. These fixes should resolve the reported problems and not cause any other issues in adjacent functionality. This process is referred to as regression testing. Regression testing usually occurs at the end of the execution of a test phase. Depending on the number of fixes that have been incorporated during the test phase and the extent of any design changes required to fix defects, significant portions of the testing may need to be run again in a new test pass.

A typical test model, such as the one shown in Figure 2, incorporates the elements of each test phase and multiple test passes. Many specific test methods or techniques exist within the broader categories of unit, product, and system testing. Figure 1 describes some of these techniques and gives guidelines for applying them.

When developing their test processes, enterprises should consider the following factors:

- Who will perform the testing
- Combinations of possible configurations to be tested or bypassed
- A strategy for developing test cases
- Time allotments for each test phase
- The sequence of test phases (serial or parallel)
- When to start and complete testing
- Appropriate measurements of product quality
- Appropriate measurements of test efficiency and adequacy
- Tools to help manage the test process and improve its efficiency
- A defect management system and strategy

The optimal test process

The test process consists of iterative verification for each level of product integration and complexity. A certain amount of repetitive testing should be built into the test model to help ensure that previously tested features and functionality are not adversely affected by product and code changes during the test cycle. Knowing when to apply various testing techniques and methods in the test process is fundamental to the testing success. In addition, testing should enhance product quality while minimizing resource costs and time to market.

Cynthia Lovin is a senior consultant test engineer in the Dell Product Group Global Test Department, with 16 years of test- and quality-engineering experience in companies ranging from startups to Fortune 500 enterprises. She has Six-Sigma Green Belt certification and a B.A. in Business Administration from The University of Texas at Austin.

Tony Yaptangco is the director of the system test group within the Dell Product Group Global Test Department. He has 25 years of experience in various software and test environments, including 16 years in engineering management positions. Tony has a B.S. in Computer Science from San Diego State University and an M.S. in Engineering Management from National Technological University.

FOR MORE INFORMATION

IEEE Software Engineering Standards Zone:
www.standards.ieee.org/software

VeriTest Testers' Network:
www.veritest.com/tn

Software Quality Engineering:
www.sqe.com

American Society for Quality:
ww.asq.org

