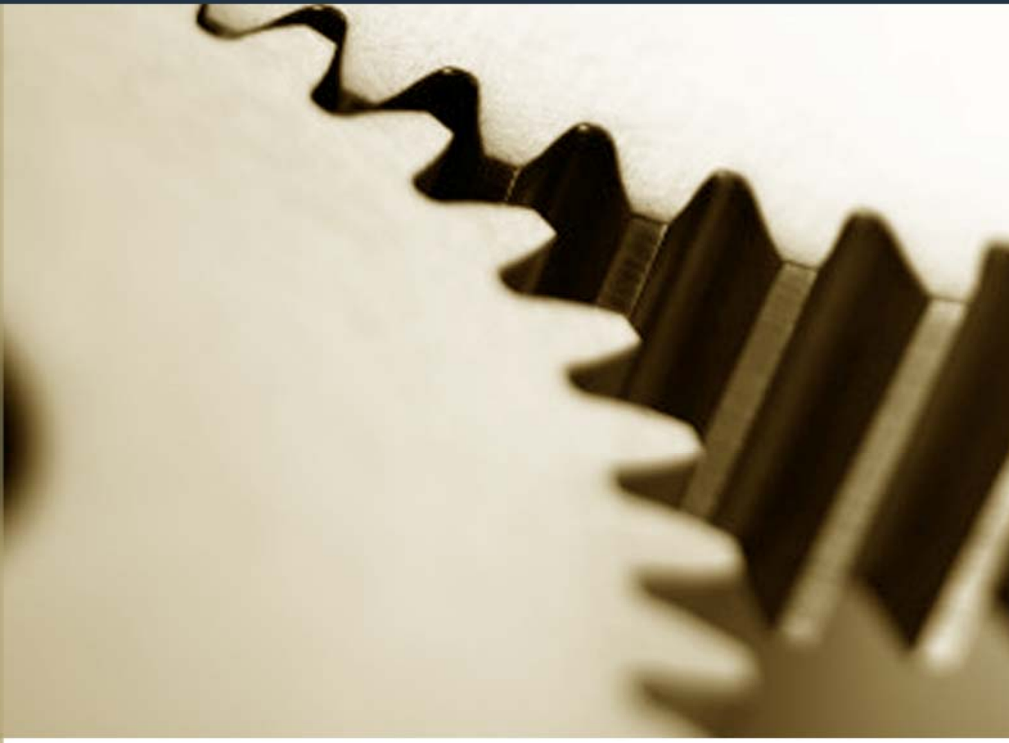
TECHNOLOGY INSIGHT
Info-Tech Advisor Premium - Understand



About this research note:

Technology Insight notes describe emerging technologies, tools, or processes as well as analyze the tactical and strategic impact they will have on the enterprise.

Set Your Sights on the Six Software Testing Targets

Publish Date: October 16, 2007

IT managers with a desire to understand software testing can quickly become overwhelmed by the various types of testing, all seeming equally important. However, when attempting to improve the functional quality of software, only six primary goals need to be met. To meet these goals, six basic types of testing can cover the gamut and effectively contribute to application quality.

Executive Summary

Software testing, the determination of how well an application works or whether it works at all, is a key factor in software quality assurance. However, functional testing can quickly become a time and cost consuming process if clear testing goals are not established.

This note discusses:

» The six primary goals of functional testing and how they can be achieved.

» How to incorporate each type of testing into the software development cycle.

» Test automation tools.

» Important factors to consider when developing a testing strategy.

To truly benefit from software testing, it is important to understand the types of testing that can be leveraged to meet the organization's goals. IT managers must consider costs, time, and resources when evaluating current and future testing initiatives.

Technology Point

With the pressures of deadlines and budget constraints, many IT managers find it hard to accomplish much more than developing applications and pushing them through to production as fast as possible. However, sacrificing critical testing steps means IT managers are also forfeiting their confidence in the software's quality.

The reality is that no software can be developed perfectly. Nevertheless, various types of testing can be performed to help build confidence in the quality of the final product. This raises the question of what types of testing should be used since even a cursory Internet search can reveal as many as 25 testing types, ranging from usability to system to compliance testing. The answer lies in establishing and focusing on the *goals* of testing. All software has the same set of primary goals that should be attained for the product to be deemed functionally correct. Planning to achieve these goals aids in the selection of the types of testing that will be executed.

All the types of testing can be categorized as helping to achieve one or more of the primary goals. Real cost savings can be attached to each primary goal in terms of the cost of re-work, quality assurance time, lost sales, etc. IT managers who fail to understand the goals of testing may end up implementing testing strategies that do not achieve their objectives, thereby releasing sub-standard code into production and driving up development costs.

For more information on application quality costs, refer to the ITA Premium research note, "Understanding the Cost of Software Quality."

What It Is & How It Works

The Six Goals of Testing

Testing should be goal oriented – otherwise, it's pointless. Testing without a set of objectives leads to never-ending, costly testing cycles that result in software with little or no quality improvements. The goals of any test initiative needs to include the six primary goals listed in Table 1.

Table 1. Six Primary Goals of Testing

Source: Info-Tech Research Group

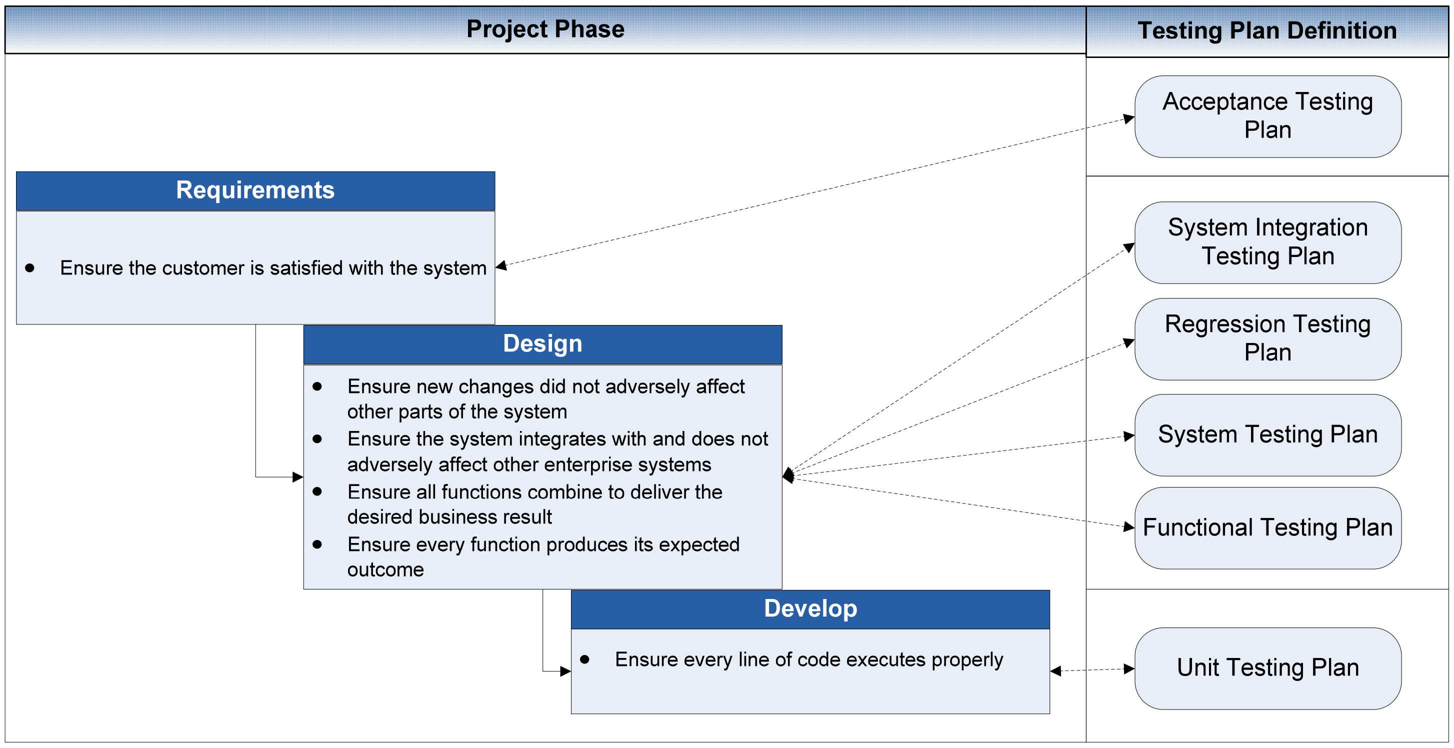
Primary Goal	Reasoning
1. Ensure that every line of code executes properly.	Developers generally write code in isolation. In the absence of complete code reviews, pair programming, or flawless developers, defects will likely evolve within the code. These defects are extremely difficult and costly to uncover later in the development cycle.
2. Ensure that every function produces its expected outcome.	In general, lines of code combine to perform functions (often referred to as “function points”). These functions have expected outcomes which are (hopefully!) described in a functional specification. Expected outcomes must be confirmed as part of any testing strategy.
3. Ensure that all functions combine to deliver the desired business result.	Functions combine into systems which deliver business results. These business results are (hopefully!) described in a business requirements document. Testing has to ensure the system meets the business objectives laid out in the business requirements document.
4. Ensure that new changes do not adversely affect other parts of the system.	Every new change to an existing system has a high probability of adversely affecting other functions in that system, causing a ripple effect of defects. Defects can be introduced by making changes both after the system has been tested and after it has been deployed. A primary goal of testing must be to ensure existing business objectives continue to be met.
5. Ensure that the system integrates with and does not adversely affect other enterprise systems.	Once the system has been shown to work properly in isolation, it must be shown to work in coordination with other systems. Systems must integrate and interoperate with each other so each can deliver their business objectives.
6. Ensure that the customer is satisfied with the system.	This goal is arguably the most important of the testing goals. IT departments exist to provide solutions that allow businesses to meet their objectives. This testing goal also helps to ensure that the requirements adequately meet the user’s expectations and avoid a “right solution, wrong problem” scenario.

Get Real: Set the Testing Goals

To be practical, the six primary goals of functional testing must be assessed on a project-by-project basis. Info-Tech recommends discussing and defining the desired goals during the design and development of the software. This ensures that test plans are thorough enough, yet focused. Figure 1 presents a model of a testing strategy that conveniently integrates testing into the Software Development Life Cycle (SDLC).

Figure 1. Goal Focused Test Planning

Source: Info-Tech Research Group



Project Phase
Testing Plan Definition
Requirements
Ensure the customer is satisfied with the system
Design
Ensure new changes did not adversely affect other parts of the system
Ensure the system integrates with and does not adversely affect other enterprise systems
Ensure all functions combine to deliver the desired business result
Ensure every function produces its expected outcome
Develop
Ensure every line of code executes properly
Acceptance Testing Plan
System Integration Testing Plan
Regression Testing Plan
System Testing Plan
Functional Testing Plan
Unit Testing Plan

For each development iteration, the phases down the left side of the model are executed. For clarity, only the Requirements, Design, and Develop phases are shown. During each phase, the diagram illustrates which type of testing plan must be defined in order to ensure each of the six test goals will be achieved.

The defined testing plans can be compiled into a single testing strategy document. The resulting testing document must include:

1. **The primary testing goals that must be attained.** Remember that each project is unique and the types and scope of testing required to ensure success may vary. Define the types of testing that will be executed to achieve the prioritized testing goals.
2. **The goals of each testing activity.** Each testing activity must have definitions that will signify testing is complete. For example, a functional test plan may specify only five critical functions that must be tested extensively, or it may specify that all functions exposed to end users must be tested with failure and success scripts.
3. **Risks that are known and whether they will be acceptable.** For example, if the software is for internal use, security risks may be low priority.

Achieve the Goals with Six Types of Testing

Given the six primary goals of testing, six types of functional testing can be used to ensure the quality of the end product. Understand these testing types and scale the execution to match the risk to the project.

1. **Ensure every line of code executes properly with… *Unit Testing*.**

   Unit testing is the process of testing each unit of code in a single component. This form of testing is carried out by the developer as the component is being developed. The developer is responsible for ensuring that each detail of the implementation is logically correct. Unit tests are normally discussed in terms of the type of coverage they provide:

   » **Function coverage:** Each function/method executed by at least one test case.

   » **Statement coverage:** Each line of code covered by at least one test case (need more test cases than above).

   » **Path coverage:** Every possible path through code covered by at least one test case (need plenty of test cases).

   Unit tests allow developers to continually ensure that a unit of code does what is intended even as associated units change. As the software evolves, unit tests are modified, serving as an up-to-date form of documentation.

2. **Ensure every function produces its expected outcome with… *Functional Testing*.**

   Functional testing addresses concerns surrounding the correct implementation of functional requirements. Commonly referred to as black box testing, this type of testing requires no knowledge of the underlying implementation.

Functional test suites are created from requirement use cases, with each scenario becoming a functional test. As a component is implemented, the respective functional test is applied to it after it has been unit tested.

For many projects, it is unreasonable to test every functional aspect of the software. Instead, define functional testing goals that are appropriate for the project. Prioritize critical and widely used functions and include other functions as time and resources permit.

For detailed information on how to correctly develop use cases to support functional testing, refer to the Info-Tech Advisor research note, "Use Cases: Steer Clear of the Pitfalls."

3. **Ensure all functions combine to deliver the desired business result with… *System Testing.***

   System testing executes end-to-end functional tests that cross software units, helping to realize the goal of ensuring that components combine to deliver the desired business result. In defining the project's system testing goals, focus on those scenarios that require critical units to integrate.

   Also, consider whether all subsystems should be tested first or if all layers of a single subsystem should be tested before being combined with another subsystem.

   Combining the various components together in one swift move should be avoided. The issue with this approach is the difficulty in localizing error. Components should be integrated incrementally after each has been tested in isolation.

4. **Ensure new changes did not adversely affect other parts of the system with… *Regression Testing*.**

   Regression Testing ensures code modifications have not inadvertently introduced bugs into the system or changed existing functionality. Goals for regression testing should include plans from the original unit, and functional and system tests phases to demonstrate that existing functionality behaves as intended.

   Determining when regression testing is sufficient can be difficult. Although it is not desirable to test the entire system again, critical functionality should be tested regardless of where the modification occurred. Regression testing should be done frequently to ensure a baseline software quality is maintained.

Don't Stop Now!

After system and regression testing, IT managers may be confident that the developed software functions as it should and become eager to release it. However, testing should not stop and the software should not be delivered; there are still two testing goals that have not been met. Testing focus now shifts to the bigger picture: how the software affects the organization.

5. **Ensure the system integrates with and does not adversely affect other enterprise systems with… *System Integration Testing*.**

   System Integration Testing is a process that assesses the software's interoperability and cooperation with other applications. Define testing goals that will exercise required communication (it is fruitless to test interaction between systems that will not collaborate once the developed system is installed). This is done using process flows that encapsulate the entire system.

   The need for a developed system to coexist with existing enterprise applications necessitates developing testing goals that can uncover faults in their integration. In the case that the new system is standalone software and there is no requirement for compatibility with any other enterprise system, system integration testing can be ignored.

6. **Ensure the customer is satisfied with the system with… *Acceptance Testing*.**

   Acceptance testing aims to test how well users interact with the system, that it does what they expect and is easy to use. Although it is the final phase of testing before software deployment, the tests themselves should be defined as early as possible in the SDLC. Early definition ensures customer expectations are set appropriately and confirms for designers that what they are building will satisfy the end user's requirements. To that end, acceptance test cases are developed from user requirements and are validated in conjunction with actual end users of the system. The process results in acceptance or rejection of the final product.

Test Automation

Software testing can easily become complicated, time-consuming, and hard to manage. Overall testing quality can suffer due to the sheer volume required for sign-off. Automated testing suites accelerate the process by generating test scripts and accurately tracking results during the testing process. They can provide an audit of issues and resolutions required to fix them. Consider automated testing tools to speed development lifecycle and functional testing.

Automation tools do not eliminate the need for a testing team or testing plans, but they do remove the requirement for continuous manual input throughout the execution of test cases. Table 2 presents the pros and cons of software test automation that should be considered before including tools in the testing strategy.

Table 2. Advantages and Disadvantages of Test Automation

Source: Info-Tech Research Group

Advantages	Disadvantages
» Executes test cases fast. » Creates reusable test suites. » Records test results and eases report generation. » Reduces costs (personnel and rework).	» Cost of tool and staff training. » Requires planning, preparation, and dedicated resources. » May not eliminate manual testing for complex cases.

There are a variety of test tools on the market, each with specific strengths, weaknesses, and goals. A good place to start is to use testing tools to help automate unit testing and regression testing. For more information on automated testing tools, refer to the Info-Tech Advisor research note, "Get to Know Automated Testing Tools."

Key Considerations

Testing is not a process that should be implemented in an ad-hoc manner. A clear understanding of the six primary testing goals and how they can be achieved provides great insight into how to begin thinking about testing software. However, that knowledge is not sufficient to begin effectively testing. The following factors must be considered when determining a software project's testing strategy and plan.

1. **Project Size and Criticality.** Large and critical projects should consume the majority of testing resources and should aim to achieve the six testing goals. These types of systems require thorough and well planned testing phases. On the other hand, when projects are relatively small or non-mission critical, using all testing types could be a waste of time and effort.
2. **Budget.** Include resources for testing when determining an accurate budget for a software project. If a software project goes over budget in its final phases, the finger of blame often points directly at a poor testing strategy and a lack of understanding of the goals of testing. Allotting sufficient resources to testing can significantly impact the budget by decreasing the time spent on fixing bugs, or it can cripple a project by overwhelming a small effort with redundant testing processes. Find the appropriate balance for each project.

Although the overall objective of testing software is to uncover *every* problem in logic, design, and implementation, it is not a very realistic objective. A more practical approach involves planning how much testing a project can realistically sustain. It is not sensible to develop blanket testing targets for all projects and budgeting for them. Consider the following factors when determining the testing effort:

- Level of criticality to organization.
- End users – internal or external.
- Conformance to specific standards.
- Software complexity.
- Time to market.

3. **Number of Testers and Tester Experience.** Testing can be a time consuming and laborious process. Ramp up as many employees for each testing phase as possible. The more eyes reviewing software, the more likely they are to find bugs, and therefore improve quality.

Make sure the lead tester is experienced enough to understand and apply the goals of testing. In most cases, having dedicated testers for each type of test is likely not feasible, so be sure to select testers who have experience with more than one form of testing type. Allowing for economies of scale when hiring experienced testing staff can decrease costs over the total project.

Key Takeaways

1. **Keep the goals of testing as the highest priority.** There are many types of testing that can help to achieve these goals, but losing sight of the goals is a roadmap for failure. Plan the test, and test according to the plan.
2. **Complete testing is impossible.** The reality is there is no such thing as complete testing; it is impossible and impractical to test every aspect of the software. Rather, IT managers and project sponsors must define what "complete testing" means for the software.
3. **Testing should be done by a dedicated testing team.** Testing systems should never be completed only by a developer. Having individuals on staff whose sole responsibility is to ensure management of test scripts and recording results provides far greater quality to a system. This does not let the developer off the hook! Unit and functional testing still need to be performed by each developer.
4. **Develop testing goals that can evolve with the software.** Start small and test what you can within budget and time constraints. It is easier to add test cases than remove them. For longer lived projects and products, develop the testing strategy and refine the goals over time.

Bottom Line

IT managers with a desire to understand software testing can quickly become overwhelmed by the various types of testing, all seeming equally important. However, when attempting to improve the functional quality of software, only six primary goals need to be met. To meet these goals, six basic types of testing can cover the gamut and effectively contribute to application quality.