Successful Testing Guidelines - 33 Software Testing Tenets

Please see "The Test Wheel – A Visual Test Management Tool and Guide" for more about the levels and types of testing, and supporting processes and tools needed to deliver suitable, maintainable, and successful systems.

It's 2016 – and time to revisit the critical importance of promoting systems through the recommended test cycles. With testing and related QA taking up as much as 50% of the time and resources on large IT projects, testing cannot be an afterthought, shortcut or compressed into the last weeks, days and hours before Go-Live.

In large-scale systems failure disputes worldwide, poor or incomplete testing is a major factor in challenged and failed projects at both customers' sites and in the courtroom. Shortcuts taken in testing fail to reveal easily and early discoverable bugs, errors and flaws that will cost orders of magnitude more to find, isolate, and fix later. Project management most often does not know how to estimate the testing and quality assurance work needed to produce a successful, reliable system. The wholesale elimination of readily available, effective automated regression testing has resulted in the introduction of bugs after-the-fact. And poorly educated project steering committees and management teams have signed off and accepted poorly constructed, executed, and incomplete testing – oftentimes ignoring "rainy day" testing of errors that will occur, but occur less frequently.

Before the list of 33 tenets that every project manager and software systems customer should know about testing, here is a crash course in Testing Basics.

1. **Q: Why is testing so critical?**
   A: Testing reveals the actual scope and quality of the system being delivered. It allows the stakeholders (i.e., customers, users, business partners, business analysts, designers, data base architects, programmers, QA, department heads, PMO, project managers, and more) to measure how well and complete the system is in terms of contracted-for functionality – and conversely, determine what functionality is not being delivered, was misunderstood, was omitted, or has been deferred – and examine the impact and effect of the working, wrong and missing functionality on the planned system use, benefits, and new personnel/business processes.

2. **Q: What are the *typical* test levels and test level sequence and what do they test for?**
   A: Unit testing looks at particular functions/code in a module at the most 'micro' scale. This is the only testing performed by programmers (not testers), requiring detailed knowledge of program design and code. **All other tests are performed by trained testers and or specialists in the area(s) to be tested – not the programmers.**

   Integration testing tests the combined parts/components/modules of a larger application to ensure they function together correctly. It includes code, tables, user procedures, hardware, telecom, APIs, servers, etc.

   Interface testing asks whether the system interfaces properly with 3rd party systems to collect, format, and properly use data transmitted or polled from 3rd party partners. Is the right data collected completely at the right time? Are transmission errors handled? Does the new system send processed data back to other 3rd parties? These interface tests are critical to answer "Does the application perform as advertised? As contracted for?" Compares planned results to actual test results. Test specialists required.

System testing with converted/audited data is based on overall requirements specs and covers **all** combined/integrated parts of a system and all interfaces. Led by the developer and developers' specialized testers and designers, with programmers and customer subject matter experts (SMEs) available as needed. There is a strong focus on data interaction, data reconciliation, network communication, interacting with other hardware, applications, or systems. System testing likely covers the complete app environment mimicking real-world use.

End-to-end testing is similar to system testing in that it is on the 'macro' end of the test scale, but the test/use cases are organized to show/prove that the important ***business functions*** work properly and as expected (e.g., from Sales Order to manufacturing or stock picking, to proper shipment by promised date, through delivery, billing, accounting, and customer payment based on customers contract pricing. So, can a salesman promise a realistic delivery date to a large customer based on queries, screens, reports and information the new system provides? Are such data and related recommendations easily available at the right time, in the right format, and with the necessary, correct information? Does a production schedule have the information to make better decisions and forecasts? Trained customer SME's and super-users will be heavily involved in these tests.

Performance testing includes **load and stress testing** to test the software under heavy loads to determine at what point response time degrades or fails. It also includes **functional testing** under unusually heavy loads, large complex queries to a database system, etc.; **recovery testing** to examine how well the system in development recovers from crashes, hardware failures, and catastrophes (*also known as failover testing*); **security testing** of the system's ability to protect against unauthorized internal or external access; and more.

User Acceptance Testing (UAT) is the final testing based on specifications of the end-user or customer over some limited period of time which looks for issues with the software at the end-user/customer/business partner (e.g., supply chain) level. The UAT typically is the contractual test which, if passed, signifies acceptance of the system by the customer and starts the warranty period. If it doesn't pass, the project team continues work on the system until it passes.

3. **Q. How much testing is enough?**
A. The short answer is, "It depends!" (William Hetzel, *The Complete Guide to Software Testing*).

It is impossible to test everything. Such 100% testing can take years to perform for large, complex systems or poorly delivered ones. By then the system may be obsolete. If requirements change at only 2% per month – that is an average requirements' change of 24% per year. That's ~50% in two years. Let risk drive the decisions on what to test.

Different customers have different opinions of what requirements are critical or important for the same system solution as well as differing levels of willingness to take on risks – and thus cut certain tests.

Also, waiting for customer approval/signoff regarding test plan completion is problematic – and does not mean the system is properly tested and has uncovered the critical and high priority errors. Customers are not trained in testing large systems (e.g. ERP Systems) and cannot reliably sign off on the number of tests NOT included in a test plan.

Some customers need assurances that all their industry regulatory requirements are met. Still others will accept incomplete or minimal testing and the potential disastrous side effects to meet certain business goals, mandated deadlines, and promises. Such shortcuts may result in missing defects until production, a possibility of losing customers (Apple's Map app for iPhone), meeting a politically assigned go-live date with a non-functional system (the healthcare.gov website), incurring high failure costs, missing a market opportunity/window, or losing market standing.

A developer and customer must use a consistent understanding of the level of risk aversion or acceptance to determine how to rank the priority of the test cases. Different criteria can prioritize testing including: most used functionality; importance of security; must-work functionality (i.e., industry or government regulations); available safety stock; on-time deliveries; user friendliness and intuitive use of GUI for users, customers and business partners; customer returns; impact of certain functions not working, etc.

4. **Q. How predictable is the Total Cost of Ownership (TCO) of the system from ideation through sunsetting?**
   A. Good TCO calculation should rate the level of system: its maintainability; operability; interoperability; extensibility; expandability; upgradability; portability; standardization; reusability; required documentation; value of the test documentation, tools and regression test suite turned-over to Maintenance, etc. These things can be tested for and evaluated. Systems without such abilities will incur excessive maintenance costs and delays, operations costs, ongoing testing, QA, and security costs, and will need to be replaced earlier (i.e., the system will have a shorter useful life than planned.)

The above basics should be reviewed in every important software development project early on so testing isn't first looked at when the project is running late. At crunch time, an "easy" way to deliver the system on-time, on-scope, on-the money is to shortcut testing. Remember, Microsoft puts on several testers on each of its products from the 1st day of development to look at the cohesiveness, completeness, clarity and testability of the requirements, architecture, design, programming standards, reused code, etc. as the project goes through each development phase.

The Industry Is Still Making the Same Testing Mistakes 50 Years Later

Today, testing is still not viewed as a **critical business process** – it's instead seen as just another technical step on a checklist. The following are 5 Testing Truths everyone should keep in mind.

1. All non-trivial software has bugs. The goal is to keep out critical- and high-impact bugs.
2. All tests are not created equal. Each one focuses on different facets of the system being built.
3. Programmers are incentivized to deliver as much good, well-documented, materially error free code following given standards in a given time frame. (i.e. They "make" code). Testers are incentivized to find as many "different" errors as possible – especially the critical and high errors (i.e. They "break" code). These fundamentally different incentives must be managed and embraced so that the two groups will work and communicate well with each other – which is needed for delivering a successful test and successful system.
4. 100 tests is ≠ 100 tests. Testers must have enough programming and testing background to know when apparently different test cases really are testing for the same programmatic or algorithmic mistakes.
5. Patterns in the errors uncovered reveal where to find more. What do they point to?

   - Lack of good requirements (i.e. incomplete, inconsistent, unclear, not concise, untestable)

- Poor forward and reverse traceability from requirements ↔ design ↔ coding ↔ testing. Are there requirements without the cross reference number to the tests planned for them? Does the design cover all the requirements when compared to the Requirements Traceability Matrix (RTM)?
- Does a large percentage of errors come from one or two programming teams? Do they need more training? Was their code/documentation peer reviewed? Did it receive static analysis? Look for more errors coming from this group until they have been properly educated.

In addition, the following **33 Tenets Learned Regarding Software Testing** are broken into three categories: management lessons; recommendations for testers; and facts that have been proven over and over again and accepted as best practices in the IT industry.

Type	#	33 Recommendations/Lessons Learned regarding Software Testing
Management	1	Testing is a Business Process – NOT a series of Technology tasks. It must be managed.
Management	2	Software development & testing today, should include customers, users, business partners, suppliers, business analysts, programmers, data base leads, conversion managers, project managers, quality assurance staff, executives, Project Steering Committees, PMO, and more. Testing is finally rising to the critical process it really is.
Management	3	Testing can take ≥ 50% of the total estimated project effort and schedule (when you include all levels of pre-test and test planning, management, and execution, quality assurance and quality control, and risk mitigation throughout testing).
Management	4	Testing produces the most efficient results when performed in a proper sequential manner. Set proper pass-fail criteria for each test and stick to it. Identify workarounds so that no single test failure brings all testing efforts to a halt.
Management	5	A robust, current, automated RTM is necessary for good testing and good maintenance.
Management	6	Negative testing (aka "rainy day" testing) is critical for risk minimization. I guarantee unusual situations will occur post Go-Live, especially if "rainy day" testing isn't completed.
Management	7	Thoroughly testing with converted data before Go-Live is critical.
Management	8	Carefully estimate testing cost, resources, schedule, quality, risks, and shareholder expectations. Apply actual hours to estimated hours and determine variances, earned value, and testing productivity for each team. Use metrics to re-estimate the future testing effort periodically.
Management	9	Industry specific ERP systems – even those allegedly requiring only simple configuration and no customization – must be fully tested to ensure the configuration meets system, functional, and business requirements. Configuring is a type of programming. Users of ERP systems do not know what assumptions were made by the ERP providers, or what design avenues were taken and rejected in the system being tested. That's why testing is critical here.
Management	10	Ensure you have tested, controlled, secure, and separate environments for software development, testing, quality assurance, and production. Use a software configuration management tool to record and control the movement of source code from one environment to the next.
Management	11	Properly use testing tools: regression testing suite, automated testing as necessary, security software, configuration management software, library control/maintenance tools, etc. Also use tools that analyze/track code: level of complexity, coverage during testing, defect entry and resolution, and meeting design and coding standards.
Management	12	Properly determine the number and types of test scenarios, test cases and use cases needed to prepare and test at each test level. Have this reviewed or performed by experienced test planning experts.
Management	13	Use & save the data, test cases, execution & acceptance of tests in an automated testing tool. It will have value later.
Management	14	Descriptions of testing process, standards and reviews must be distributed, trained and enforced. This includes locations and naming standards for files and work products, and descriptions of the tests to be performed.
Management	15	An upper management executive should/must be assigned as the "Testing Phase Champion" to assure the importance of testing is upheld and testing shortcuts are not taken to meet fixed delivery dates.
Testers	16	Testers should be part of the initial requirements elicitation team and creation and maintenance of the RTM. This will help ensure clear, concise, consistent, and complete requirements.
Testers	17	Testers are breakers; programmers are makers. Testers have a different mindset to "make" the system reliable. Testing requires special skills and tools. Hire, train and advance appropriate certified software testers within your organization.
Testers	18	Final user testing must allow users to interface with the system in their new organizational roles and positions.

Type	#	33 Recommendations/Lessons Learned regarding Software Testing
Testers	19	Test for quality and non-functional "abilities" (reliability, testability, maintainability, auditability, usability, securability, recoverability, stability, scalability, portability, survivability, and performance).
Testers	20	DevOps approach to systems development calls for early and continuous testing. Test early and often.
Testers	21	Software-Defined environments –1) Virtualization and Simulated artifacts and infrastructures, and 2) Deployment Automation and Release Management tools are now being effectively used to permit early and ongoing testing, deployment, and change. This requires a culture change. "Integrated automation tools increase efficiency, enhance scalability & test reliability, aid collaboration & traceability, and improve quality…" IBM, DevOps, 2015
Testers	22	Each testing role must be planned for, making specific persons responsible, including: data preparers, database admins (DBA), expected result preparers, testers, QA/Independent IV&V auditors, business analysts, programmers, SMEs, super-users, fixers, etc. Independent V&V or QA testing should be performed during each testing level; such professionals should vote on readiness for promotion to next test level.
Fact	23	Testing is the most underestimated and least understood of the SDLC phases.
Fact	24	All tests are not created equal. Each focuses on different aspects of the system being built and build on one another.
Fact	25	All non-trivial software has bugs. The goal is to keep out critical- and high-impact bugs.
Fact	26	Regression testing must be performed, period!
Fact	27	A regression testing suite must be updated/current, documented and turned over to maintenance for their use.
Fact	28	It costs exponentially more to fix errors not found in the earlier SDLC phase when they were first injected.
Fact	29	"Users do the craziest things." Things developers, programmers, and professional testers would never think of. So DON'T shortcut UAT.
Fact	30	There are many causes of testing errors. Find and fix the "root causes": - User error - Training error - Unclear Requirements -Design errors - Conversion defects - Integration error - Poor GUI - Test database bug - Interface error - Bad error messages - Out of sequence test steps
Fact	31	Politically mandated Go-Live dates most often require testing be severely shortcut - which causes system failure.
Fact	32	Testing the same thing 100 times only counts as one test!
Fact	33	Dynamic testing only finds 85% of the defects. To address and uncover the remaining 15% of "latent" defects, the testing team must perform Static Analyses (e.g., reviews, inspections, audits and walk-throughs).