# Building an Independent Test Group

**3/8/2004**

**Scott Eder**
**Executive Director SQA**
**Catalina Marketing**

## Table of Contents

## Introduction

Software quality. What is it? Companies have been struggling to answer that question for years. Instead of coming up with a single answer, companies embark on an evolutionary process based on their current situation and the maturity of their software development efforts. This journey is based on success. As the company is more successful and their user base grows, so does the need to release continually higher-quality products. One example of a company's software quality evolution is outlined below:

- *Production Testing* – developers perform their unit testing, but the majority of the testing is performed in Production. The company's goal at this point in their history is to get their product into as many hands as possible.

- *Developer Testing* – developers spend more time on unit testing and perform some level of integration/system testing prior to Production release to prevent some the defects from reaching Production in front of growing, less tolerant user-base. The goal of the company at this point is to continue to expand the user base, but also to keep the current customers satisfied by delivering a more stable product.

- *Power User/Beta Testing* – developers are spending more and more time on testing, which draws them away from expanding the functionality. The company needs to expand the product offerings in order to continue growing the user base and to address the requests coming in from their current users. As the product matures, so do the users. The "power user" emerges and is asked to exercise the new version of software as they would normally use the product prior to Production release and report any issues they discover. By this time, the user base has grown large and critical or major defects found in Production have direct financial impact to both the company and their customers.

- *Independent Test Group (ITG)* – developers are developing and unit testing new products and functionality and even though power users perform their Beta testing, critical and major defects still make it to Production. It's at this point, when a company continuously feels financial pain from poor quality, that they commit to a focused software quality effort; hence, the birth of the Independent Test Group.

This illustrates but one example. Different companies progress at a faster pace or even start at some of the latter stages, while others may never evolve at all. It depends on the needs of the business.

This paper focuses on the latter stage of the software quality evolutionary process above namely, introducing and establishing an Independent Test Group (ITG). It covers some high-level conceptual areas like establishing the overall guidelines and objectives for the group, communicating those objectives to other areas of the business, and providing a sense of community to the tests. It then details some practical processes to provide structure to the new testing initiatives. Regardless of the evolutionary stage in which a company finds itself, the basic test processes in this document provide a structured approach that can be followed by anyone interested in improving software quality.

## Establishing the ITG Direction

The Company has made the decision to invest in an ITG. Now what? As illustrated in the section above, the decision to form an ITG was made because of the financial impact of Production defects; therefore, the primary objective of the ITG is to prevent Production defects. To do this effectively requires resources -- time and people. The trick is identifying just how much of each is needed to provide the appropriate level of quality.

### *Appropriate Levels of Software Quality*

The test management of an ITG is a juggling act. The Test Manager must take into account the number of available resources, the size of the application, the current scheduled release dates and business drivers in order to effectively manage and release the right product. The right product for the business may not necessarily be the one with the fewest defects. It may be the one that delivers the most value in the shortest amount of time; therefore, the Test Manager must understand what areas of the application need the most testing focus and what areas can be delivered with a minimum amount of testing. The Test Manager should research and develop the appropriate level of quality.

#### Understanding the Business Drivers

Determining the appropriate level of software quality starts with understanding the business drivers. What is important to the business?

- Is it looking to deliver new functionality as quickly as possible? If so, the Test Manager needs to fit into this paradigm and align the test resources to not significantly impede this effort, while providing measurably better software releases. Focusing the testing efforts on specific critical areas of the software is more important than testing every single condition for every single field throughout the application.

- Is the nature of the business such that Production defects cannot be tolerated as may be the case with certain regulated industries? If so, the testing must be as comprehensive as possible and take as long as necessary. Releasing the product on a certain date is not as important as ensuring that defects are discovered and fixed prior to release.

Aligning the test objectives with the business drivers will enable the business to be more successful and help the ITG to succeed.

#### Understanding the Customer/User Usage and Needs

Once the business drivers are understood, the next step is to focus on the user community. What are their expectations from the software?

- Does the user require all aspects of the software to execute flawlessly? When asked as a group, the general consensus will be a resounding, “Yes!” However, when asked individually, the user’s viewpoint is specific to the functionality they use.

- What would the user be willing to accept? The answer to this question depends on the application and the user. It's important to understand the trade-offs the user would be willing to accept. Some would sacrifice quality over getting new functionality quicker. Others would be willing to wait until the defects were worked out before accepting the new software.

The bottom line is that understanding what drives the user community for a given application helps establish the appropriate level of quality

### Understanding the Application

The third facet of the appropriate level of quality equation is in understanding the application itself. What are the critical functions performed by the application?

The following activities will help determine the appropriate level of quality for a given application.

- *Separate application into component parts* – identify the major sections of functionality within a given application.
- *Develop information process flow diagrams* – identify and document the flow of information from one application component to the next.
- *Analyze reported Production defects for impact and common trends*– analyze a set of Production defects from the previous release to determine the overall impact of a given defect. When determining impact, assess the customer liability, damage to reputation, time and resources needed to develop and test the fix, and potential delays to future schedules. Defect trends help identify areas of the software that are heavily used and need more testing focus.

Understanding the components, assessing potential defect impact and analyzing the Production defect history will help determine what areas of an application must be tested more thoroughly than others.

Once these three items – business drivers, customer usage and needs, and the application— are understood, the test manager can better direct the test efforts to establish a win/win/win scenario for the business, the users and the test team. Expectations need to be set and buy-in achieved as to what the ITG can do, will do and why.

## Generating Support Throughout the Organization

The act of researching and developing the appropriate level of quality measurement sets the tone for generating support throughout the organization. Actually sitting down with the stakeholders to understand their business and their needs could very well be charting new territory.

### *Stakeholder Interviews*

During these stakeholder interviews, the Test Manager has a golden opportunity to espouse the need for quality processes and the direction in which the ITG is heading in addition to understanding the needs of that particular stakeholder.

### *Follow-up Communication and Updates*

Once the appropriate quality level has been ascertained and expectations set, the real work begins. Keep the stakeholders in the loop. Open and honest communication about the ITG's quality efforts go along way in generating support. Acknowledge areas that need more work, tout areas that are working well, and let everyone as major milestones are achieved.

## Creating a Sense of Identity

Up to this point in the document, the focus has been on establishing the direction of the ITG by working externally with management and the user community. It is important not to neglect those who make up the actual ITG. These testers will be working toward meeting the goals established in the previous section and dealing with the typical test-related issues in the trenches from the potential animosity that arises when developers first start working with testers to establishing themselves as valued members of the software development team. There are a couple of techniques to help ease the load on the team members and provide support when needed.

### *Establish Name Recognition*

It's important for most people to feel like they belong to a community of their peers. This is no different for the members of the ITG. Since this is an independent test group, most likely it has been established as its own department either within IT or on the Operations side of the business. The department can be that supportive community provided the concept of teamwork and the open sharing of ideas and knowledge are fostered and openly encouraged. A little advertising helps to build name recognition throughout the organization. While engaged in conversation with other employees outside the ITG, bring up the department and what it's trying to do. Word will spread.

### *Celebrate Success*

When a project is completed and the release successfully deployed to Production, celebrate! Working through all the planning, test execution and defect resolution means a lot of hard work. The team needs to realize that they have accomplished something and take credit.

### *Produce Quality Deliverables*

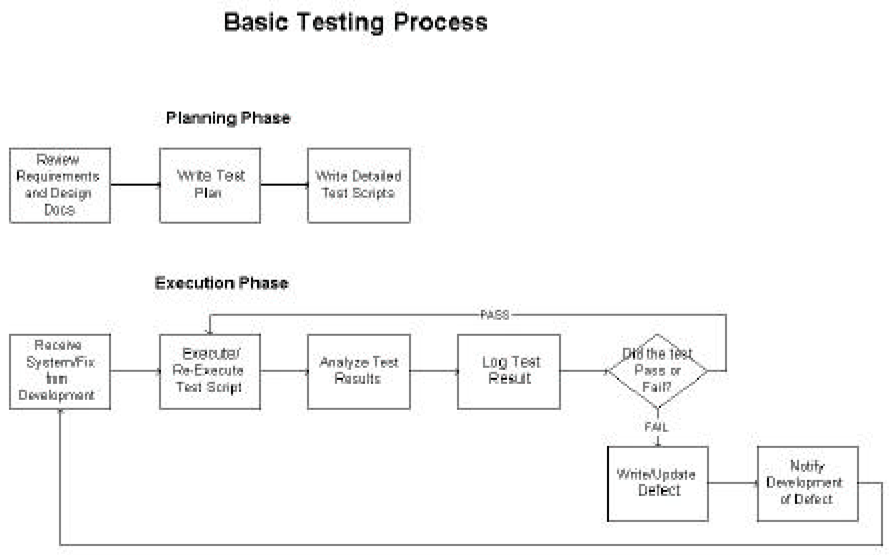
A tester represents the quality of his or her application and whether he or she is executing a test, writing a test case, or especially documenting a defect, he or she must be cognizant of the quality of the work. Perception is reality. In some cases, the only interface between a tester and developer is through a defect. If the tester consistently does a poor job of documenting defects, the developer will notice and then question the integrity of the testers work. The same holds true for other deliverables and can also be said of other project roles. Deliver quality products…period.

## Implementing Basic Testing Processes

The processes explained in the following sections are designed to provide a starting point for an organization looking to build structure into their testing approach. Each process has been pared down to its basic functional elements and data flow. They should be reviewed and modified to work for a particular organization.

### *Basic Testing Process*

The testing process boils down into two primary phases—Test Planning and Test Execution. The following process diagram illustrates a basic testing process.


Basic Testing Process
Planning Phase
Review Requirements and Design Docs
Write Test Plan
Write Detailed Test Scripts
Execution Phase
PASS
Receive System/Fix from Development
Execute/ Re-Execute Test Script
Analyze Test Results
Log Test Result
Did the test Pass or Fail?
FAIL
Write/Update Defect
Notify Development of Defect

During the Test Planning phase, the Test Plan and detailed test cases are written against the known functionality, requirements and design documentation. During the Test Execution phase, the test cases are executed against the delivered system according to the documented test plan. As test cases fail and defects are uncovered, they're reported to development to be addressed according to the defined Defect Management process. This iterative process continues until the software is released to Production.

### Test Planning and Scripting

The primary focus of the test planning phase is to determine what has to be done in order to deliver the best application to Production. There are two basic deliverables during this phase – the Test Plan and detailed test scripts.

**Test Plan**
The Test Plan is ultimately a high-level document based on the requirements and design that explains the overall approach to testing the application. It doesn't go into the gory details of

each test case. It's written to convey to the test team and the other project stakeholders how the application will be validated. Some of the key information contained in the Test Plan includes:

- *Test Entrance Criteria* – these are the milestones, or tasks, that must be completed before the application can enter the Test Execution phase of the project. Examples of typical Entrance criteria are as follows:
  - Code is written and unit tested.
  - All defects discovered during unit testing have been addressed.
  - Detailed test cases have been written.

- *Test Exit/Success Criteria* – these are the milestones, or tasks, that must be completed before the application is released to Production. Examples of typical Exit/Success criteria are as follows:
  - All test cases have been successfully executed.
  - All critical and major defects have been fixed.
  - All other defects found during testing have been addressed. By addressed, it means that the defects were either fixed or deferred to another release based on the decision by the Test and Development managers. Upon exiting the testing phase of the project, there should be no defects in an ambiguous state.
  - System performance meets or exceeds the established baseline.

- *Test Risks* - lists the potential issues that could impact the project specifically related to the testing phase. Impact to the project could be a schedule delay, change in the project scope or a myriad of other things that could change the current course of the project.

- *Test Assumptions* – lists some basic decisions made by the test team in advance of the testing effort.

- *Testing Scope* – based on the appropriate level of quality, list the items that will be tested and also those items that will not be tested.

- *Test Approach* – describes how the testing will be performed at a high level. One technique is to break the application down into functional components and address each one separately. As an example, for a GUI-based data entry application, the functional components may contain items such as Log on, Data Entry, Data Storage, and Reporting. For each component describe at a high level the types of tests that will be executed, how the results will be validated and the data that will be needed for the testing.

- *Test Estimates/Schedule* – quantifies the effort needed to accomplish the testing documented in the test approach.

- *Test Resources* – identifies the people who will be performing the tests and what areas they will be responsible for testing.

- *Test Environment* – details the actual environment to be used in testing. This could include needed hardware, database regions, software tools, testing facilities, etc…

**Detailed Test Scripts**
The detailed test scripts specifically document the steps that will be executed to validate the application functionality. There are numerous ways to document test scripts. Regardless of whether the scripts are being entered into a Test Management tool or they are being written in Word or Excel, there are still two essential ingredients—Test Steps and Expected Results. The basic concept of cause and effect applies. When a certain command button is clicked on the screen, for instance, a specific function is supposed to happen. The Test Steps are the actions that a tester should perform. The Expected Results are critical since they explain how the system should react to the action taken by the tester.

The following table depicts a basic test script template.

| | | | | | |
|---|---|---|---|---|---|
| **Test Name:** | | | | | |
| **Purpose:** | | | | | |
| **Test Setup Parameters:** • | | | | | |
| | | | | | |
| **#** | **Test Step** | **Expected Results** | **Actual Results** | **Pass/ Fail** | **Comments** |
| 1. | | | | | |

A few new fields, in addition to Test Steps and Expected Results, have been added to this template to make script management easier and to prepare for the Test Execution phase of the project.

- *Test Name* – a unique identifier for the test. A naming convention should be developed to help identify specific test scripts in the future.
- *Purpose* – defines what the test script is validating.
- *Test Setup Parameters* – describes any special setup procedures that need to be executed prior to running this test. It also establishes the starting point in the application for the test.
- *#* - represents the number of steps in the script.
- *Actual Results* – field established to track the actual testing results when this script is executed during the Test Execution phase.
- *Pass/Fail* – designates whether the test passed or failed validation against the Expected Results. This will be filled in during the execution of the test script.
- *Comments* – field established to allow for the entry of comments during the execution of the test script. Comments are free form and could contain specific information about the test results or information about a defect discovered during the execution of that specific step. It could also be left blank if there are no comments to add.

Another way to document test cases is through the use of a test matrix. A test matrix provides a quick way to generate a large number of cases in a "checklist" type fashion. It does not have the same level of detail as shown in the above step-by-step example, but can be used effectively when the above level of detail is not deemed necessary or unachievable due to time contraints. The following example shows how a simple test matrix may look if a tester wanted to cover some basic data entry tests for a given screen:

| **Field Name** | **Under Min Value** | **Minimum Value** | **Mid Range Value** | **Maximum Value** | **Over Maximum Value** |
|---|---|---|---|---|---|
| First Name | | | | | |
| Last Name | | | | | |
| Middle Name | | | | | |
| Age | | | | | |
| Address 1 | | | | | |
| Address 2 | | | | | |
| Address 3 | | | | | |
| City | | | | | |
| State | | | | | |
| Zip | | | | | |

Each cell in this matrix represents a test case. The first column in the table shows the names of the fields on the screen to be tested. The remaining columns list the different tests to run in each field.

- *Under Min Value* – enter data in the field less than the low end value needed for this field. This is a negative test designed to cause an error. It tests the entry of data out of tolerance. If the field is supposed to accept from 4-10 characters, enter three. The system should respond appropriately.
- *Minimum Value* – enter data in the field that satisfies the minimum data requirement. If the field is supposed to accept from 4-10 characters, enter four. This is a positive test for valid data.
- *Mid Range Value* – enter data that falls within the acceptable range of data. If the field is supposed to accept from 4-10 characters, enter 5 or 8. This is a positive test for valid data.
- *Maximum Value* – enter data that satisfies the maximum data requirement. If the field is supposed to accept from 4-10 characters, enter 10. This is a positive test for valid data.
- *Over Maximum Value* – enter data in the field greater than the maximum requirement. If the field is supposed to accept from 4-10 characters, try to enter 11 or higher. This is a negative test designed to cause an error.

Again, this matrix is only testing some basic data entry functions. However, in order to use this matrix effectively, a tester must already have a good understanding of the system and its requirements. It requires more discipline on the testers' part to maintain and use test matrices, but can be a powerful tool.

### Test Execution

This is where the proverbial rubber meets the road.  The planning and scripting have been completed, or close to it, and the software has been delivered into the test team's eager hands. It's time to execute the test scripts according to the Test Plan.  Resources have been assigned to their respective areas and start executing their respective scripts.

The Execution phase is a very iterative process comprised of executing a series of tests, validating the test results against expected results, documenting any uncovered defects, and testing the fixes when delivered.  This process continues until the Test Exit/Success criteria are met as documented in the Test Plan.

The Test Execution phase of the project can be the easiest phase for the test professional provided enough time is spent in the planning phase and a sound defect management process is in place.

## *Defect Management*

During the test execution phase, a means for documenting and tracking defects must be established.  Clear, concise and accurate defect documentation is essential to the success of any software development process.  It is through the iterative process of finding and fixing defects that a developmental project matures into a production-ready system.  An effective and efficient method of dealing with defects can ultimately save time in the development schedule and lead to higher quality.  In order for this process to work, though, all parties involved need to use the same terms and have a common understanding of what needs to be done at what stage.

### Common Defect Data Elements

First of all, what is a defect?  For the sake of this document, a defect is defined simply as something in the application that is not correct.  A defect can occur in virtually any part of the application under test including the graphical user interface, reports/output, calculations, menu/screen navigation, hardware interfaces, software interfaces, etc…

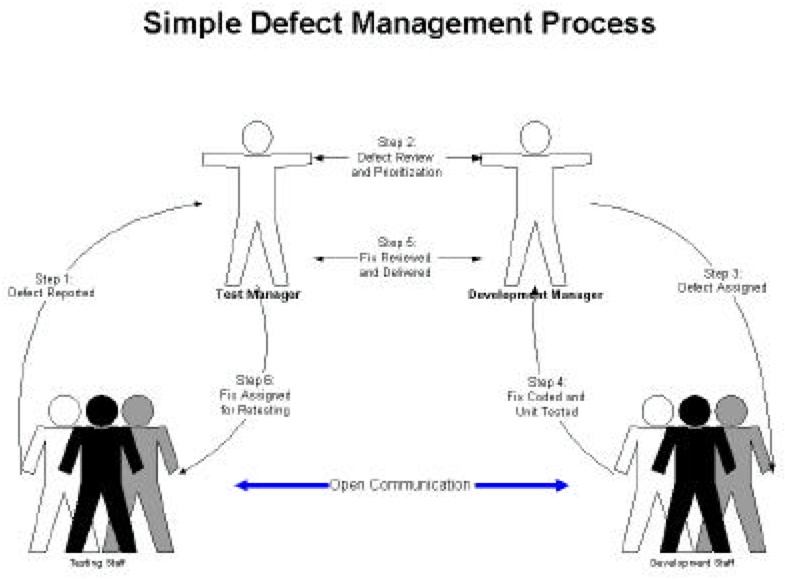
When documenting a defect, there are certain key data elements that need to be addressed.

- *The Problem* – a description of what the tester deems to be incorrect
- *Who* – the tester who found the problem
- *Why* – a brief explanation of why this is incorrect.  This could be as simple as a reference to a specific requirement or a calculation in a Design document.
- *What* – a brief explanation of what the correct behavior should be.
- *Where* – the portion of the application in which the problem occurred.  This could be a screen, a process, a report, etc…
- *When* – the date on which this issue was found.  This will be used later to determine defect find rates over time.
- *How* – the steps the tester was performing to cause the problem.
- *Severity* – gauges how soon a fix needs to be made
  - Critical – Testing cannot continue until this defect is corrected
  - Major –  A large  issue exists, but testing can continue (must be fixed prior to release)

  - Minor – A small issue exists, but testing can continue (should be fixed prior to release)
  - Cosmetic – An aesthetic issue (fix if there's time)
- *Priority* – determines the order in which defects should be addressed by development
- *Supporting Documentation* – include any documentation or screen shots that show the defect
- *Reproducib*le? – indicate if the defect can be reproduced every time the documented steps are executed or if the error happens sporadically.

## Simple Defect Management Process

Now that the common defect elements have been identified, we need to place them in a context to be used; hence, the following simple defect management process.


Simple Defect Management Process
Step 2: Defect Review and Prioritization
Step 1: Defect Reported
Test Manager
Step 5: Fix Reviewed and Delivered
Development Manager
Step 3: Defect Assigned
Step 6: Fix Assigned for Retesting
Step 4: Fix Coded and Unit Tested
Open Communication
Testing Staff
Development Staff

This process traces a defect's lifecycle from detection and write-up through to the fix and retest. The individual steps in the process are self-explanatory, but it's important to emphasize the open communication between the testing and development teams. Sometimes a quick conversation can lead to a better understanding than a whole string of emails or discussion threads.