Chris DeNardis

Chris DeNardis is a Supervisor of Software Engineering at Rockwell Automation, where he directs the test group for the Electronic Operator Interface, the System Quality Lab and the Software Tools Group. He has tried to automate much of the reporting features of testing, while keeping test process simple and easy to follow. He fosters team building and accountability.

Chris has reviewed books such as "Lessons Learned in Software Testing" by Kaner, Bach, and Pettichord; and the upcoming book from Rex Black, "Critical Testing Processes." Chris is listed on the practicality gauntlet of *STQE* magazine as well as for StickyMinds. He monitors the StickyMinds.com Message Boards as an Industry Specialist. Chris' philosophy is that regardless of one's position, the most important title is "Student."

Chris can be reached at cpdenardis@ra.rockwell.com or at cdenardis@wi.rr.com

BIO
Chris DeNardis is a Supervisor of Software Engineering at Rockwell Automation, where he directs the test group for the Electronic Operator Interface, the System Quality Lab and the Software Tools Group. He has tried to automate much of the reporting features of testing, while keeping test process simple and easy to follow. He fosters team building and accountability.

Chris has reviewed books such as "Lessons Learned in Software Testing" by Kaner, Bach, and Pettichord; and the upcoming book from Rex Black, "Critical Testing Processes." Chris is listed on the practicality gauntlet of *STQE* magazine as well as for StickyMinds. He monitors the StickyMinds.com Message Boards as an Industry Specialist. Chris' philosophy is that regardless of one's position, the most important title is "Student."

Chris can be reached at cpdenardis@ra.rockwell.com or at cdenardis@wi.rr.com

---

Six years ago, I embarked on the career as a software tester with a company involved in control engineering. After just one year of testing, I was promoted to leader of the test group. As I read Pam Hardy's article "Perspectives from a new software tester" in STQE Vol. 2, Number 2, I thought of myself a few years back when not only was I new to testing, but now responsible for managing the test group.

Prior to testing, I worked for a company where I had developed, tested and installed control systems. I also provided technical support to these systems I installed. I have seen how customers used the software. Therefore as a tester, I was thorough and sadistic, at times, testing the software by applying much of what I saw customers doing. It was my job to find those bugs, so the customer would not. After about a year of testing, I was put in charge of the Test Group.

Once I became test leader I was fortunate that much of the testing process was already in place. However, since testing was done manually, I had to spend much of my time making certain that testing was completed. I also had a group of testers that I worked with side by side. However each tester had their own definitions of what severity to apply to a defect. Or how to verify a defect. Project leaders always asked how much time will you need to test out this feature so that they could calculate a budget for test time. There was always a question as to why a certain area is tested or what to test. Then as testing commenced, project leaders would ask how far testing was, or when do I expect it to be completed. Finally, I felt as if there was a "we versus they" scenario between the developers and testers. There seemed to be a disjoint between development and test groups.

There were four things that became very obvious to me, that were necessary to get better organized:

1. To have a common set of ground rules on the test progress, defect reporting, and verification.
2. Be able to convey how is testing going on a frequent basis.
3. Be able to determine what do I need to test and stand behind the reasons why.
4. Maintain good communication with the technical leaders to help move the product through the development phases by being proactive rather than reactive.

Common Set of Ground Rules

There were processes, which I had implemented on my own as a tester to help me better organizes my testing. I realized that a majority of my testers all had different mechanisms. My test group seemed to be open to anything as long as it did not take too much time to do and did not impede the testing activity. Some testers would copy the test cases directly to the test logs, while others would summarize them. Therefore, it was hard to tell if this was the exact test case tested, or if it summarized, indicating the testing was also summarized. When defects were found, they would not always get put into the testlogs. Therefore, it would appear testlogs were passing, but defects were still found with that area. Therefore, I had to come up with a technique, but – "keep it simple stupid", or the KISS principle. We did not need another process to slow us down, as there was a belief that testing was the bottleneck of product deliverables. There was usually so much to test, and usually not enough testers nor time.

I wanted total buy in from my group of testers otherwise, the new ideas I implemented would be doomed to failure. So I empowered my test group to come up with ideas. Most of the testers in my group had experience with the software we produced -while others learned how to use the software through testing and training. All

the ideas were consolidated into a “The Testers User’s Guide”, and shared with the project leaders. (Email me if you’d like to see a copy of our Testers User’s Guide – or check up on Stickyminds web page for the template.)

This guide clarifies the process done of test so that when we tested, though it maybe in different ways, we are able to have a common set of ground rules to follow that facilitated the documentation of our testing. This guide had things such as:

- What testing is all about.
- What is expected from the testers.
- Where to find all the programs on our internal network in order to perform the tests.
- What is a “typical day” of testing, and the exercises that are involved on a daily routine of testing.

This guide also serves as a training guide to new testers that come on board. The guide contains answers to the more typical questions asked, to help decrease ambiguity involved in the process of testing. It instructed a tester how the process works in the test group. First, getting assigned the task and entering this into their time sheet. Second, locating the testlog on the network and begin testing it. If they found a defect, what to do. And if they needed to verify a defect what to do.

We use testlogs to document what tests have been completed. Testing is done formally through the completion of predefined testlogs with test cases. If the test case passed, it was marked as such. If a defect was found, it was recorded in a defect tracking system. The process we use for defects entered is a closed loop / corrective action operation. Which basically means, the defect entered can only be closed by the tester who submitted it - after it has been verified. A tester would submit a defect and rate the severity of the defect as either a “Show Stopper”, “No Workaround”, “Work Around”, “Spelling / Nitch”, or a suggestion. All these severities are defined in “The Testers User’s Guide” . The defect coordinator would prioritize the defect based on the description as either 1-HIGH – must be fixed before shipping, 2-MED - fix if there is time. Or 3-LOW – most likely gets moved to the next project. Then the defect would be assigned to a developer to fix. The developer would fix the defect, and it would be returned to the tester to verify. The tester was responsible for verifying the defect.

The next problem the guide addresses is how the testers deal with the different defect resolutions. There are various resolutions such as “Fixed”, “Not a Defect”, “Not Repeatable”, “Not Implemented”, “Do Not fix”, and “Transferred”. Steps are outlined for what the testers should do in the event of each of these resolutions. Therefore, all the testers follow the same steps for all the defects resolved.

Other thing this guide suggests is the proper way to describe the defect and to include any associated files, pictures, or steps to help the developer reproduce the problem and fix it. We follow the same scenario as Rex Black’s article “Grounding Test Status in a well written bug report” from STQE Volume 2, Number 2, with the exception of the review of defect by others before it is submitted.

Recently there was a survey done from the Quality Assurance institute and published that talked about testing standards and procedures, process improvement plans, testing activities, tester training, and statistical testing asking those participants various questions. The survey can be found at Reference: http://www.qaiusa.com/bill-perry-download/word download files/Test Survey.doc. Interesting things found in that survey which support what I was doing was that:

- Test reports not filled out consistently or correctly – 38% do, while 53% sometimes, and 9% not at all.
- Everyone had their own process to defect verification process – 71% use standards, while 29% don’t.
- 59% of the test organizations re-use their test cases.

How is Testing going?

This was a typical question asked of the test leader. Another question asked “What percent of testing is completed, and how much is failing”? Finally the more important question – “When do you expect to be completed with testing”? In order to do this, one had to quantify the amount of testing that was tested, and what was remaining. In addition, how many of these tests actually passed, and which ones are failing.

In order to accomplish this, I quantify the testing by the number of given test cases provided prior to start testing. Granted I expect that testers will be adding test cases as they go along, which will be also tracked. Secondly, I measure this over time, or on a daily schedule. Testers are instructed to update their testlogs daily. Therefore each day, the number test cases passed, failed, and what remains are tallied. This is all summarized in what I

call a Consolidated log of the entire project. The consolidated log contains information from each testlog by the tester, as well as a summary of the progress reported by each testlog.

From the consolidation process of these testlogs on a daily process, a forecast is generated based on trends of execution. In figure 1, and example test suite was setup and trended over several days. From the information shown figure 1, it appears, based on the example, that we are executing our test plan ahead of schedule, but our total passing seems to be falling behind such that it will be completed nearly a week from the execution end date. Thus, I can give an answer on "How is testing going".

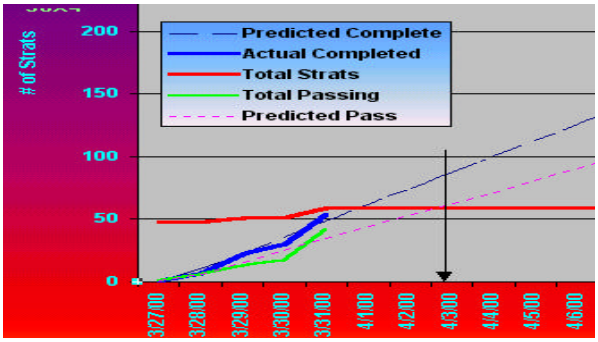

Figure 1 Project Completion chart.

In order to achieve these results of tracking, modifications had to be done in the implementation of both the testlogs, and nature of how testing was executed. It relies upon the testers updated their testlogs on a frequent basis. In the past, our testing documentation process used simple excel worksheets to record the test case pass or failure (See figure 2). The test cases were copied from the test plan to the excel sheet of that test area under the Test Objective column. If the test case passed, the tester put "pass" in the test result column. If the test case failed, a defect was entered into the defect tracking system, and recorded in the testlog as well under the Defect # column. In addition, the severity of the defect was recorded here in the severity column. Once the defect was resolved, and verified by the tester, a Y, for YES, was put into the Verified column.

TestLog

Functionality:	Firmware Build #:				Page 1 of 1
Tester:	Software Build #:				
STS Doc:					STS Rev:
	Start Date:			Completion Date:	
TEST OBJECTIVE / DESCRIPTION	S#	Defect#	Severity	TEST RESULT	Verified(Y/N)
Test Case 1	s1	478	2	failed	
Test Case 2	s2	479	3	failed	
Test Case 3	s3	480	4	failed	
Test Case 4	s4				
Test Case 5	s5			pass	
Test Case 5	s6	491	2	Failed - now passing	y

Figure 2 First Testlog

In the past, I had to manually go through all these testlogs, and verify that all the test cases were executed. In addition I would have to verify that all defects found were verified in the defect-tracking database, and that they were verified in the testlog.

The intention of these logs was good. However these logs were hard to maintain because they were not used consistently. A typical testing phase would contain at least 20 – 25 of these testlogs, with as many as 100 test cases inside each of them. What made it difficult, was verifying that the test cases were tested and that all the defects found were resolved and verified.

In addition to the test cases provided, testers are instructed to test above and beyond these given test cases, as this is where the defects are usually found. The Excel worksheet did not lend itself well in documenting the above and beyond test cases. Or it was hard to tell what was done. Each tester had their own way of documenting this.

To improve the manageability of the testlogs, the testlogs were programmed using VBA (Visual basic for applications) to verify each entry, color the row based on severity of the defect, and validate the entries for fixed defects (See Figure 3). Therefore when the defect had a severity of 1-ShowStopper, or 2-No Workaround, the color of that row would be Red. Likewise, if the defect submitted was considered to be a 3-Workaround, the color of the row would be yellow. Followed by 4-spelling / nitch in a cyan color. Finally 5-Suggestions, would be entered in green. This way when reviewing the testlogs, a tester or test coordinator, could easily distinguish the severity of the defects by the color.

Once the defect was fixed, the tester is instructed to verify the defect, and place a Y, for yes, in the verified column. Once this is completed the row color would return back to the default color. In addition testers would enter the software release build number that the defect verified in.

Some of the defects found would not always get fixed. Some of the other resolutions that are applied to the defects are NAD – Not a defect, NI – Not implemented, DNF – Do Not fix, NR – Not repeatable, MP – Moved to the next project, or LP – Lowered priority. All of these "Verifications" would also get tracked in this log followed by, if necessary, a comment about what project it was moved to, or what priority it was lowered to. See Figure 4 where MP, and LP as well DNF resolutions were used. In these cases the row color would revert back to normal state leaving the rows – something about the purposes

TestLog

Functionality:Area2	Firmware Build:						
Tester: John Doe	Software Build #:	5					
STS Doc: Test Documet for STQE						STS REV 1.0	
	Start Date: 3/28/2000		Completion Date:				
Show Stopper - 1 / No Work Around - 2 / Work Around - 3 / Spell / Nit - 4 / Suggestion - 5 / Tech Suggestion - 6	s*,z*	>1	[0-6]	Insert Row (Pass = Passed Test; N/A = Not applicable; else enter the failed results)		Current Build	
Do Not Put Strategies Above This Line	S#	Defect#	Severity	TEST RESULT	Verified	Build #	Pass
Test Case 1	s1	478	2	failed			F
Test Case 2	s2	479	3	failed			F
Test Case 3	s3	480	4	failed			F
Test Case 4	s4	490	5	failed			F
Test Case 5	s5			pass			P
Test Case 6	s6	491	2	failed	Y	6	P

Figure 3 New Testlog

Each tester is responsible for testing "above and beyond" from those test cases presented. Those test cases written by the tester are be put in the testlog under the section of "Other things tried" (See figure 4). As you can see the color is back to normal even though a defect was logged, but is now moved to another project (MP), or lowered in priority (LP), or do not fix (DNF).

Other things tried							
Tester Input Case 1	z1	464	1	FAILED	MP	Project 3.50	P
Tester Input Case 2	z2	511	4	FAILED	LP	2-MED	P
Tester Input Case 3	z3	465	3	FAILED	DNF		P
Do Not Put Strategies Below This Line							

Figure 4 Other Things Tried

When the log is saved, and closed, the testlog validates its entries, and updates statistics about the testlog such as the number of test cases that passed, failed, verified, and were added to the testlog. When all the testing is complete, and all the defects fixed, and verified the history of the defects would remain – see figure 4.

These logs are stored on the project directory located on our network. Therefore, when the testing is nearly completed, all of these testlogs are opened, and verified that all test cases were executed, and that there were no defects left unverified.

The next improvement was to consolidate these logs and summarize the results on a single sheet. The contents of each log are pulled in as tab worksheet inside this consolidated log (See the tabs located in the figure 4). The example provided has four test areas. A copy of these test logs can be viewed by selecting these tabs. In one quick view, one can see what testlogs were updated. How many test cases passed, failed, were added, and what are the severities the defects are at. Therefore, if there are any test cases that failed, those are indicated by a red color (see test logs in areas 2 and 3 of figure 4). Likewise, a testlog that is completely tested, and passed would show up as green. (See "Install" testlog in figure 5). Those testlogs which did not have severe defects are indicated by the Pass/Conditional condition or yellow color (See "Volume" testlog of figure 5.).

Below the testlog listings in the consolidated sheet, is a summary of all the testing, showing percent complete, percent passed, and percent failed. Special attention is applied to the number of test cases, which are added during a test phase. This is indicated by the Added Strategies column – or the above and beyond testing. So for our test example (in figure 5) there were 48 test cases. Currently, with addition 11 test cases, at the time of this consolidation there are 59.

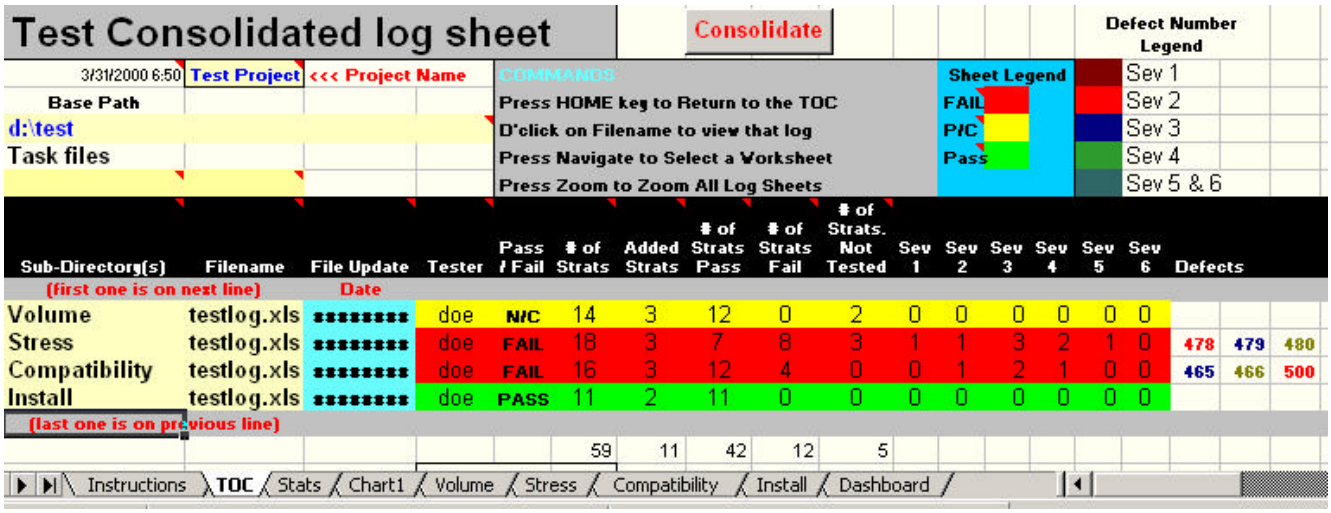

Figure 5 Consolidated Log

This method makes managing the testing activities with the use of testlogs easier. When the testing is completed, all the rows should show green – indicating that all of the test cases inside the testlog are executed, and passing.

The consolidation happens on a daily basis. With the information collected daily, a forecasted completion date can be predicted by using the trend function in Microsoft Excel. The chart summarizes the progress, and is published on our intranet.

Additionally to communicate the areas we were testing, as far as passing/failing and if failing what defect nubmber, I included a redition of James Bach's Dashboard (see figure 6) (http://www.satisfice.com/presentations/dashboard.pdf) which was posted on our intranet site. This way a software manager could see what defects were holding up test, or failing.

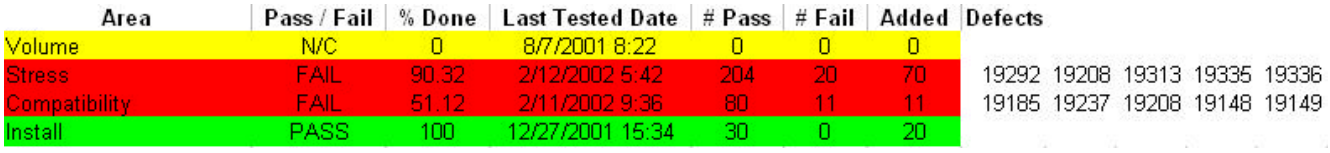

Figure 6 Dashboard

The final step of the consolidation process is to query the defects found in the testlogs, against the defect tracking system. The defect tracking system returns information about the defect, such as fixed, lowered priority, or if it is moved to another project.

What do I need to test?

Software products continue to evolve with new features, new functionality, and new operating systems to run on. The biggest thing to wrestle with is the new features and requests from customers. What areas do I need to test out? How much time do I have to properly test out the product to assure quality? Finally, will development give it to me in time so that I have ample time to test it? I have found this to be the most difficult part of being a test leader. You would like to test it all, and make sure there are no bugs. But there is a timeline to follow and products that must be shipped. Risk analysis is done on the all the areas tested. From this analysis the determination is made on what areas can one safely say is OK not to test.

The more typical problem most test leaders run into is to have enough time to properly test all the new features. I ran into a situation where just a few months after I took over as a leader I was faced with the task of getting a product tested and ready for the customer use for a customer commitment. If I had applied past techniques of testing it all, I never would have made it through testing. The product was getting more mature, and was only adding a few new features to it. I had a customer commitment to meet. Therefore, I did a risk analysis on all the areas, and only applied those that were at most risk.

In order to begin to figure out what testing needs to be done for the next test phase, I have to consult with the development team to determine what areas were going to be modified or enhanced. If the area is new, a functional requirement specification (FRS) is composed by the developer. This FRS is inspected by marketing, development, technical leads, and the test leader. A formal inspection log summarizes all the problems or suggestions with the FRS. And once everyone agrees to all the changes to the FRS, it is signed off, and development begins. At this point, test cases are derived from these requirements for testing. I have found that without these set of requirements, development slips because it continues to enhance the feature without a requirement. Testing is limited because we have nothing to test against other than our understanding of how it works.

In addition to writing the test cases for the next phase of testing, there is an evaluation of all the customer problems that came in since the last release. Test cases are either rewritten or enhanced to include those problems found by the customer. This would be a part of the regression testing effort necessary to verify that the problems were fixed.

Finally, with all the test cases added, I compose a test plan. This plan is communicated to the project leaders, and technical developers informing them of the areas that the test group will test. A question is asked if there were any areas, which did not need to be tested, or areas, which need more embellishing. This way the project leaders and developers know what is going to be tested and can comment before we actually begin testing.

Maintain good communication

With all the automation plugged into test process, as well as reporting, this information has to be properly conveyed so as to not to be the “doom and gloom” of the product. As Rex Black says in his article “Effective Test Status Reporting” the test coordinator has to convey this information properly so that it is understood even if it is not good news.

It is good to be proactive, and work with the technical developers side by side, to avoid “we versus they” scenario. To get over the “we versus they” scenario I and other technical leaders get together as a team to determine what features we are going to put in. We keep in mind the amount of time that will be necessary to develop the feature as well as test the feature to make this a success.

I found that after we were successful in getting the product out the door on time, it was due to the fact that we all worked together. Development kept me aware of things that were changing, as well as stressed what areas should be tested. The test team concentrated on those particular areas versus those that did not have changes to. The new features went through a “integration phase” where a developer and tester were paired up to go through, test and repair the feature prior to formal testing. Before formal testing would start, we would conduct a team meeting to demonstrate all the new features, the problems still getting worked on, and the functionality of the feature. This meeting would include the entire test team, marketing, and development. This way the problems could be ironed out before formal testing took place. In the end, we beat our overall average of execution, and got the product out the door on time.

As a test leader, I feel that I have to be well versed both in the product, and in testing techniques to properly point out possible problem areas. As James Bach points out, I give my testers the areas where to flash their flash lights toward. The defects submitted are emailed to the leaders, including myself. This way I can forward the email back to the tester, and ask for clarification, or explain to the tester why it is not a defect or what to check into further. Sometimes I feel for my testers when they find a real bug, but after consulting with our technical support and customer base, I have to defer it until the next project. Therefore, it is a good idea to do a risk analysis on the possible defect, while cross-referencing with tech support.

When testing begins and defects roll in, we can quickly evaluate what areas are failing. If testing is off to a rough start, we can be proactive, and stop formal testing, and use my influence as a test leader to go into integration testing. The way we do integration testing is more one on one, the tester and developer working together as a type of “unit testing” until that area is more stable.

Because the defects are emailed to us from the defect-tracking program, we can quickly ascertain the severity of the defect, and I can either comment back to the tester, or inform the defect coordinator that it needs to be fixed since it’s halting testing in that area. In some cases I have to defend the testers point of view. Sometimes the defects can be deferred until the next project since our customers haven’t found that particular bug. As a test leader I have to stand in front of defects that are real problems, but also be able to facilitate a discussion about the trade off and make a decision to let the defect go for next release.

As we near test completion, and shippable date, we hold a defect review meeting to determine what defects are left to get fixed, and what defects we can safely push forward with. Additional meetings may be called to summarize what defects have been fixed, and those to be deferred. Sometimes we hold a risk assessment on those remaining defects that will hold up the release of the product.

Summary

As a test leader, get a handle on the problems that are in your test process. Determine what goals you need to work on to get your group up and running. This is not an easy task, as there are deadlines to be met. Empower your testers to come up with the changes that would make the testing more efficient. Testers know their job, and usually have good ideas to help improve. Then come to an agreement on the ideas, and develop a guide line for the group, and make sure it is followed.

Locate or develop the tools necessary to assist you in your planning, estimating, and forecasting of completion. This way efficiencies can be found, as well as documenting the entire test results. After all you cannot manage what you cannot measure.

To determine what to test, start out with requirements of what needs to be tested as well as the enhancements added to the product. Once the plan is developed, pass it out to the developers so they are aware of what is getting tested. Encourage your testers to assist in the test plan efforts. Collect data on past defects to support your testing efforts. And use customer input to increase the usability of the product.

Communication is an essential part of this entire process. It helps to not only communicate the progress of testing, but it also aids in the team work between the developers and testers as they work toward a common goal – to deliver quality software product.

Finally, the process doesn’t stop here. It continues to evolve, with new ideas that have come along. Use your influence as a test leader to bring forth the most efficient way of testing, while not sacrificing the quality of the product.

Final Notes

This paper was derived from the article “Perspectives from a Test Manager” published in STQE magazine Sept 2000 and modified with updated information for this conference. The original article can be found at:

- Stickyminds web page.
- International Software Testing Institute (http://www.softtest.org/sigs/manage.htm)
- In Macromedia flash presentation at Data Kinetics website http://conference.dkl.com/stp/seminars/test_manager.html.