

# TASSQUARTERLY

VOLUME 3, ISSUE 4  
October 2005



## Tester-Developer Relations

**James Bach** provides an approach to the relationship between testers and the developers which can be a challenge to both sides. In order to do their jobs effectively, each team needs specific information and effective communication from the other.

## Test Automation: Think Before You Leap

**Richard Bornet** investigates implementing test automation in a test environment. Unless carefully thought out and planned, such an implementation can add to increased time and cost, the very issues it was supposed to address. There also has a high chance of total failure. To avoid the pitfalls many aspects of such an implementation need to be considered first.

## Testing Without a Map

**Michael Bolton** challenges strict adherence to testing to requirements. In some circumstances, important defects are often discovered when testing outside of set requirements or even without any previously identified requirements.

## Managing Requirements Changes in SDLC

**Murat Guvenc** explores requirements change management. Such changes need to be traced throughout the evolution of the project to support the dynamic nature of software development and testing.

# TASSQ<sub>QUARTERLY</sub>

## FEATURES

- 4 Tester-Developer Relations**  
by James Bach
- 7 Test Automation: Think Before You Leap**  
by Richard Borne
- 13 They Just Don't Get It: Part Three**  
by Bob Sparkes
- 14 Book Review**  
**Beyond Microsoft PowerPoint: What Public Speaking is Really About**  
by Barbara Lake PMP, CSTE
- 17 Managing Requirements Changes In SDLC**  
by Murat Guvenc
- 20 Humour**  
**QA Sound Bites**
- 21 Testing Without A Map**  
by Michael Bolton

## DEPARTMENTS

<b>Editorial</b>	Page 3
<b>Conference Watch</b>	Page 6

## TASSQ

Toronto Association of Systems and Software Quality

An organization of professionals dedicated to promoting Quality Assurance in Information Technology.

Phone: (416) 444-4168

Fax: (416) 444-1274

Email: [tassquarterly@tassq.org](mailto:tassquarterly@tassq.org)

Web Site : [www.tassq.org](http://www.tassq.org)

# TASSQ<sub>U</sub>ARTERLY

Are you new to the testing game or have you recently been asked to manage a testing team or to find new solutions to persistent issues? Where do you begin when people turn to you for answers? Before you attempt to “Google” search the Quality Assurance field, we encourage you to start here. This month’s feature articles are definitely worth reading.

James Bach’s article “Tester-Developer Relations” guides us through the tricky field of team dynamics within IT development shops within the industry. One of our challenges within the QA field regarding the start-up or build of testing teams is how testers interact with the rest of the IT organization. Mutual respect and a clear understanding of the role within the organization will help avoid the QA versus Coders attitudes that plague many teams today. James’ article is to the point and for many of us, a great reminder of a basic golden rule.

You may be faced with the task of creating an automation test team. I can speak personally on this topic and have been asked on numerous occasions to share my experiences. However, Richard Bornet’s article “Think before you leap” is key to helping avoid the major pitfalls of automated testing. The summary check list provided in Richard’s article is an extremely valuable guide to help you determine if your testing team is really ready for automation.

A favourite testing technique that I encourage my team to use is exploratory testing. It has become very apparent to me over the last few years that testers are becoming conditioned to test software using conventional methods or blinders. However, the testing team is losing a great opportunity to provide general feedback to the development team on the use and experience of the software. Michael Bolton’s article on heuristics and exploratory testing challenges our conventional testing techniques.

You will find more thought provoking articles and ideas with this issue to help you improve your organizational quality. Murat Guvenc provides in-depth analysis for managing requirements and explains the necessary steps within the software development life cycle.

We have even included some humour- QA Sound Bites. We hope to have a humour section in each issue of the Quarterly.

We hope you find this issue of the TASSQuarterly inspiring and thought- provoking in your quality journey.

## EDITOR-IN-CHIEF

Joe Larizza

## ASK TASSQ EDITOR

Fiona Charles

## BOOK REVIEW EDITOR

Michael Bolton

## MANAGING EDITOR

Jeanette Mount

## CONTRIBUTING WRITERS

Joe Larizza

Robert Sparkes

Richard Bornet

Michael Bolton

## ART DIRECTOR

Nuree Hwang

## PRODUCTION

Jeanette Mount

**Copyright** © 2005 Toronto Association of Systems & Software Quality. All rights reserved.

*Joe Larizza*

Editor-in-Chief

# TASSQ<sub>QUARTERLY</sub>

*This article is a detailed summary of ways in which developers and testers could interact in order to test more efficiently and effectively. It provides a tester's "wish list" to the developers and outlines commitments testers can offer developers in return.*

## What Testers Would Like from Developers

A wise tester will say "I don't *need* any of these things from you, but to the extent that you can provide them, I will be able to test more efficiently and effectively. I'll make better reports to you and I'll find better problems before it's too late."

### Communication

- Be available to meet with testers.
- Answer tester questions.
- Adopt the attitude that the testers are trying to help you look good out there.

### Quality

- Take responsibility for the quality of the product.
- Work with testers to set quality criteria, so that both testers and developers are pulling for the same thing.

### Scheduling

- Inform testers as to when features will be implemented, so that they can prepare to test them.
- Remember that "testing time" in the schedule is also "fixing time" for developers. Leave room in the schedule for testing and bug fixing to occur.

### Features and Internals

- Inform the testers about the features that will comprise the product, and solicit their concerns or questions.
- For each feature, the more information testers have about how it's supposed to work, the better they will be able to test it.
- Think through your error handling and reporting architecture. Obscure error messages are frustrating for everyone, and slow down testing.

## Tester-Developer Relations

by James Bach

### Requirements

- Help testers understand the purpose of each feature, and how it is expected to be used.

### Bugs

- Come to an agreement with testers on a bug reporting protocol.
- Respond promptly and politely to bug reports.
- Help testers understand how they might write more helpful bug reports.
- Give enhancement requests serious consideration.

### Builds

- Establish a stable, repeatable build process.
- Provide good builds on a regular basis.
- Test the build at least a little before passing it to the testers. Consider building automation unit tests as you go (Extreme Programmers do that).

### Changes

- Late in the project, the less you change the code, the less re-testing will be required. Avoid making unnecessary changes.
- Alert testers to any changes in the design or code, and the nature of those changes. Help them figure out how best to test those changes.

### Risks

- Think about what happens when things go wrong, the unexpected happens, or the user does something that seems stupid.
- Help the testers understand where bugs are more likely to be found, or what kinds of bugs are more likely to be serious.
- Alert the testers if their test strategy doesn't appear to be focused on risks.

### Testability

- Design your product to be easy to test.
- If you develop a tool that helps you diagnose problems or test the product, give it to the testers, too.
- Help the testers produce example data for testing.

- Remember, the more bugs are in the product, the slower the testing will go, because investigating and reporting bugs takes a lot of time.

### Test Planning

- Review the test plan. Help the testers discover problems in it.
- Be sure you understand how your product is being tested.

### Testers' Commitments to Developers

As a tester or test manager, I make the following commitments to developers unilaterally and immediately. It doesn't matter whether they're friendly to me or not. These commitments are an expression of my belief about how a helpful tester behaves. Over time, I think testers who behave this way earn respect and support from the developers.

- We know we aren't the gatekeepers of quality. We don't "own" quality. Shipping a good product is a goal shared by the whole project team.
- We will test your code as soon as we can after you deliver it to us.
- We will test important things first, and try to find important problems.
- We will write clear, concise, thoughtful, and respectful problem reports.
- We will try to test in ways that don't slow down the pace of the project.
- We will let you know how we're testing, and listen to your suggestions.
- We will learn from our mistakes and look for ways to test better and faster.
- We will consider adjusting our processes to work with you better.
- We will not carelessly waste your time.
- Over time, we will improve our technical skills and product knowledge.

### Bio:

*James Bach (<http://www.satisfice.com>) is a pioneer in the discipline of exploratory software testing and a founding member of the Context-Driven School of Testing. He is the author (with Kaner and Pettichord) of "Lessons Learned in Software Testing: A Context-Driven Approach". Starting as a programmer in 1983, James turned to testing in 1987 at Apple Computer, going on to work at several market-driven software companies and testing companies that follow the Silicon Valley tradition of high innovation and agility. James founded Satisfice, Inc. in 1999, a tester training and consulting company based in Front Royal, Virginia.*

### Two Special Events!

**James Bach with Michael Bolton present**

***Rapid Software Testing***  
One-Day Workshop

-- and --

**James Bach presents**

***Skilled Testers and Their Enemies***  
Evening presentation

November 29, 2005  
Toronto

*For details see page 25*

### 4th Annual QAI / QAAM Conference ***Techniques and Tools for Solving IT Challenges***

November 1-3, 2005  
Baltimore / Washington DC Area

407.363.1111 ~ Fax: 407.363.1112  
[web: http://www.qaiworldwide.org/conferences/index.html](http://www.qaiworldwide.org/conferences/index.html)



**Data Kinetics**, the exclusive Canadian representative of the **Quality Assurance Institute (QAI)** offers software testing courses that help prepare **TASSQ** members for certification as either a Certified Software Tester or Certified Software Quality Analyst.

**For Toronto Association of System and Software Quality Members only**  
Save 15% off regular course fees

**2005 / 2006 Public Education Schedule**

<b>Q4/05</b>	<b>Toronto</b>
October 12-13	Essentials of TEST Leadership - \$1,195
October 14-15	Essentials of User Acceptance Testing - \$1,195
October 12-13	Writing & Testing Requirements - \$1,195
October 14-15	Effectively Managing Software Projects - 1,195
October 18-20	Essentials of Software Testing - \$1,695
October 21-22	Essentials of Web Testing - \$1,195

<b>Q4/05</b>	<b>Mississauga</b>
December 5-7	Essentials of Software Testing - \$1,695
December 8-9	Essentials of Web Testing - \$1,195
December 12-13	Essentials of User Acceptance Testing - \$1,195
December 14-15	Essentials of TEST Leadership - \$1,195

For customized on-site training: If your Test Team (5 or more) would like any of the above listed courses (customized) delivered at your facility, please contact [aphomin@dkl.com](mailto:aphomin@dkl.com)

For more details and online registration: [http://www.dkl.com/education/software\\_quality/register.php](http://www.dkl.com/education/software_quality/register.php)  
Call Al Phomin: 1-800-267-0730 ext. 227 or email us: [testing@dkl.com](mailto:testing@dkl.com)

## Conference Watch

### 4th Annual QAI / QAAM Conference

*Baltimore / Washington DC Area November 1-3, 2005*

Page 5

### Quality Assurance Institute Seminars

Page 16

### Project World

*Toronto Conference May 8-12, 2006*

Page 24

### QAI's 26th Annual Software Quality Assurance Conference

*Orlando, FL April 24-28, 2006*

Page 24

# TASSQ<sub>QUARTERLY</sub>

## Test Automation: Think Before You Leap

By Richard Bornet

*This article provides a guide to help effectively implement test automation. At the heart of this discussion are 12 questions you must answer “Yes” to in order to even consider bringing automation to your test environment. Essentially, the message here is to examine automated testing needs, resources and requirements before purchasing the automation suite you are considering. Find the automation tool to fit your needs not fit your testing needs to the automation tool.*

James Bach and I do not always see eye to eye, but we do agree on the present state of what goes on in the name of test automation. Where we differ is in the conclusions we draw from that present state.

Bach writes, “Test automation efforts often end up in a quagmire. Expensive test tools are purchased and handed off to testers who may not have the time or skills to write robust test programs. Or, the tools are delivered to the custody of a dedicated test automation team, which disappear into a dark room trying to design and deploy the ultimate automation framework. Months later, there are a whole lot of scripts--but somehow not much that runs.”

I could not agree more.

James Bach uses this as an argument for showing that, in reality, test automation is of limited value, and energy should be spent elsewhere to get better value.

I could not agree less.

Just because many car mechanics cannot fix a car, this does not mean we stop fixing cars, we just find better car mechanics. Just because medicine could not cure smallpox at one time, this does not mean we should have stopped trying; we just had to find a more effective method and now we have eliminated smallpox from the globe.

The same is true for test automation. If it is not very effective, it does not mean it is not the cure for our problems. It may just mean that we make too many mistakes, do not do it right, and use the wrong approach and people.

For example, I am often approached by organizations asking me to help them with their test automation effort. In many of these cases, they had bought a test automation tool and then discovered that they could not make it work effectively.

The problem is that because of lack of experience, and because of expert salesmanship on the part of the test automation tool company, the organization ploughed ahead with a particular approach (“this tool will solve our automation problem”) without first thinking through what they wanted to achieve and how to get there.

Let us use an analogy to illustrate this situation. Let us pretend that I am a travel consultant and not a test automation specialist. I am approached to help with travel arrangements for a trip to Gnome, Alaska. I am more than happy to help, but then they add this extra bit of information, “Oh, by the way, we bought a BMW for this purpose.”

Now, I have nothing against BMWs, they are excellent cars. But is a BMW the best way to get to Alaska? Wouldn't the fuel costs easily outstrip the airfare? And, it would take such a long time to get there and back! Then I find out that they need to do this trip in the winter and the actual location is outside of Gnome, a bit off the highway. Well I guess you could put some skis on a BMW; anything is possible. Then, I find out that not only do they have to go to Gnome, they also have to go to Austin, Texas and Tampa, Florida. Again BMWs are great, but they may not be the best way to do this.

Another variation of this is “We need to go to Hawaii. Oh, by the way, we are a GM shop so we have to use a GM vehicle.” That's nice, GM is a very reputable company, but it is obvious that getting to Hawaii may be very difficult, if not impossible, using a GM car or truck.

Now, in these two scenarios the problem and solution are obvious. You don't buy the car first and then try to figure out how to use it to get to Gnome and the other locales. First you look at the situation and then decide how to spend the money. Unfortunately, this logic does not seem to get applied to the area of test automation. In test automation,

too often people buy the tool first, then, somehow, hope to adapt it to the situation. “We bought WinRunner, now make it work”. “We’re a Compuware shop”. But, the way to solve this problem is to a) identify all the issues and come up with possible solutions, then b) identify if there any tools that may be helpful, not the other way around.

This fixation with a specific tool or tool producing company causes even more problems. Let us use another analogy. Let us say that we want to build some shelves. We get a hammer and a screwdriver, and we say, “That’s it. We have all the tools we are allowed to use”. Now hammers and screwdrivers are very useful tools, but every carpenter can tell you they are not enough to build shelves. We will also need a sander, a drill, a plane, a tape measure and so on. Too often in test automation, once an organization buys the tool, everything has to work using that tool, even though it may not be appropriate.

**Will you be successful in your test automation effort?**

The tool is only one of the problems one has to overcome. The following questionnaire will help you do the analysis before you embark on your test automation effort.

Answer the following question first before continuing with the rest of the article.

TEST AUTOMATION ANALYSIS	
Which of the following scenarios best describes your goal for test automation?	Select A or B
A: I would be satisfied with limited test automation. I just need a few scripts to do some very basic testing. I need it only for smoke tests. It is a small addition to our testing effort.	
B: I want to fully automate the testing of our application(s). Test automation will be a significant and central part of our testing effort. We need it for full regression. I have several applications that I need to automate.	

The scenario that you selected above determines the approach to take.

**Scenario A: Limited Test Automation**

If the answer to the goal of test automation was “A”, limited test automation, then buy a tool and get someone who is

proficient at the tool to write you a limited number of scripts. This will allow you to execute a limited number of test cases. You just need to make sure that the tool will be able to properly interact with your application(s). But be cognizant that you will only have a limited (read small) part of your application automated.

**Scenario B: Significant Test Automation**

If the answer to the goal of test automation was “B”, significant amount of test automation, then answer the questions below:

TEST AUTOMATION ANALYSIS	
<b>B: Significant automation</b>	Answer Yes or No
1. Do you realize that test automation is a development project?	
2. Do you have people skilled enough to do the coding?	
3. Are the people with the knowledge of the applications and what needs to be tested going to use and control the test automation?	
4. Have you completed a full inventory of all the applications and components that you want to test?	
5. If you have several applications which work in concert with each other, do you have a plan how you will integrate them into one test automation suite?	
6. Do you have a method to organize the test data that can be controlled simply?	
7. Can you afford the tool costs of having more than a few people use test automation?	
8. Do you have an efficient method of passing the results of test automation to developers?	
9. Can you live without knowing exactly which test cases have passed and failed?	
10. Are you willing to make adjustment to the application to accommodate test automation?	

11. Have you thoroughly investigated all the technical aspects you will need to code in the applications? This includes having the correct tools and strategy on how to code all the components and functions that you need.	
12. Are you and management fully committed to test automation?	

If you answered “NO” to ANY of the questions above you have two choices:

1. Solve the problems and make the changes to turn the “No” to a “Yes”.
2. Live with very limited test automation or no automation.

Before we continue, let us go through the above questions (1–12) for Section B. It may give you more insight.

1. Do you realize that test automation is a development project?

No matter what the tools manufacturers tell you, serious test automation is a development project. Someone is going to have to write code. Do not have any illusions about this. I have been on projects where the test automation code is more sophisticated than the code which was used in the application itself. Test automation is a coding project.

2. Do you have people skilled enough to do the coding?

The poorer the coder, the less scope there is. If you take a tester with no coding experience, then you will have only record and play back scenarios. If you have a mediocre coder, many aspects of the application will not be tested because the skill level of coder is not sufficient. If you get a superb coder you may actually get code that works.

Different automation tools use different programming languages. The coder you have may or may not know that language, and may have to learn it. From extensive experience I can tell you that you will get more done in less time by getting a good coder who doesn't know the tool and has to learn it, rather than by starting with a poor coder who already knows the tool.

3. Are the people with the knowledge of the applications and what needs to be tested going to use and control the test automation?

The tester is the person who will be running the tests. Generally, this is also the person who knows the business. The coder, as mentioned above, is the test automation expert who will be writing the test automation code. If you are lucky, these two are one and the same person – the coder is the expert on the application – but this is not likely.

If these two individuals are different, then you need a way to get the information from one to the other. If they sit beside each other and that is their primary responsibility, it may work. But, too often what you get is that the tester passes the information to the test automation expert who works with many other people. The tester loses control of the testing and, therefore is not happy. Then a new build comes in and the tester says, “I need these changes to the test automation”, and the test automation expert says, “I can get it done by the end of next week” (which is a week after the change is scheduled to go into production).

I realized very early on that if test automation is to succeed, the *tester*, not the test automation expert, needs to control test automation. The *tester* needs to create the test scenarios, run them and adjust them as necessary. “Hand-offs”, where the tester gives instructions to the test automation expert who then adjusts and runs tests, do *not* succeed.

This seems like a contradiction; the coder is writing the test automation code, not the tester. But the tester is the one who has to be able to adjust the test scenarios at will. To do this, a method has to be found to allow the tester to do this *without* having to write any of the test automation code. I am currently making a living because I figured out how to make this happen.

4. Have you completed a full inventory of all the applications and components that you want to test?

Every test automation project I have worked on has involved more than was initially envisioned. There were more applications involved, and each of the applications brought something complex to the table which had to be handled.

Here is an example from a recent project. The testers said that we needed to automate the testing of reports. After some investigation, we discovered that even though this application was a Windows application, the reports were produced on the UNIX box. A PostScript file was created and then printed. This was not a question of “Point and Click”. We had to write code to enter the UNIX environment, pull the PostScript file,

dismember it, move it to Windows, recreate it so it looked like the report and then test it. It was perfectly doable, but was far more complex than the original specs had suggested.

Every test automation project has these kinds of issues and some are even more complicated. That is why you need to make an inventory of all the things you want to automate, and then think about who will do it, and whether the project is feasible.

Now, if you answered “Yes” to Question 2 (Do you have people skilled enough to do the coding?), is the answer still “Yes”?

5. If you have several applications which work in concert with each other, do you have a plan of how you will integrate them into one test automation suite?

The majority of the projects I am involved with have multiple applications working together. You may have:

- Mainframe
- Some Client Server applications
- Flat files being passed between the applications
- XML
- Web interfaces
- Web Services
- Java applications.
- One weird application written a long time ago.

Automating just what you can may be better than nothing, but really you need to automate the lot to have the type of full testing that really pays off.

It is quite probable that a standard test automation tool will not be able to interact with all these components. Therefore, are you going to buy several tools and somehow link them together? Are you going to have a different interface for each of these applications? Well that doesn't make much sense. What are you going to do with the applications the tools cannot fully recognize and handle? How are you going to incorporate specialized code? You are in the world of partial automation at best; is this what you wanted?

By now most people are completely out of their depth. Now, go back and answer question 2. This is not an impossible situation, but test automation needs to be organized differently. We have solved these types of problems; how we did it is the subject of a different article.

6. Do you have a method to organize the test data that can be controlled simply?

Before you say “Yes”, let me throw some numbers at you. The test data is the values that will be entered into the various fields in the application and the values you expect to show up in the fields as a result. To make the automation scripts re-usable, the data has to be separate from the scripts, not hard-coded into the scripts.

Now, a mainframe application may have 600 screens, with each screen holding 30 fields, that is, 18000 variables. Most test automation tools cannot handle that many variables without some very complicated coding. Where are you going to store this data? The usual answer is in spreadsheets. But Excel uses 256 columns, not 18000. This means a minimum of 71 spreadsheets and this is only your first application. I guess you could build your own Access database, but what is the user interface going to be, and will this give you the control and simplicity you want?

Not being able to control your test data simply and efficiently highly limits the scope of test automation. We realized this very early on and knew this was one of the first things we had to solve.

7. Can you afford the tool costs of having more than a few people use test automation?

One of the powers of test automation is that multiple people can use it to perform a variety of tests and tasks. Test automation can provide more functionality than just executing tests. Let me give an example. With a project in which I am currently involved, we have to constantly create new clients to be able to execute tests. The new client is not the test, just a pre-requisite to be able to perform the test. It takes about 15 minutes to create a full client manually. We have created automated scripts which do it in 3 minutes. Therefore, we have a list of various types of clients and when we need a specific type, we run that one.

To be able to do this, it becomes very expensive to have everyone equipped with test automation tools, especially when you have more than 10 testers. And, what about the developers? That was one of the reasons we stopped using formal test automation tools and wrote the code to interact with the application directly. We can then compile this code and give it to one other person or to 100 – it is the same price.

8. Do you have an efficient method of passing the results of test automation to developers?

Remember what developers want. They need a screen by screen description of exactly what you did. It would be best if this information was attached to the defect tracking system. Most of the tools do not give you this information, and if they do, they force you to buy extra software for each developer.

9. Can you live without knowing exactly which test cases have passed and failed?

A test case is something very specific; for example, these are 3 different test cases:

- The amount is adjusted by x if the client is male
- The amount is adjusted by y if the client smokes
- The amount is adjusted by z if the client is over 50

Testers often think that the point of testing is checking that every single one of these specific test cases works properly. But in order to test each of these individually, we would have to create individual test scripts for each test case. For example:

- A test with a client that is a male 60-year old smoker
- Another test with a client that is a female 60-year old smoker

Then we can compare the result of these two to check whether the gender factor was properly taken into account.

Any decently sized application has literally thousands of such specific test cases. Are you going to create thousands of test scripts? Of course not. It becomes unmanageable. To be practical, each test script bundles together many, many test cases. For example, we run the test for a client who is a male married 60-year old smoker with no heart condition, and we check that the value is correct.

Now, what if the value is not correct? Then we know that something failed in this test, but we do not know exactly which test case failed. It could be that the gender was not taken correctly into consideration, or it could be that the calculation would have been just fine but we were not able to put in the marital status because the field was disabled.

Not knowing exactly which test case failed does not hamper your development efforts. Remember that the defect is not the failed test case, but is something which

is not working in the application; for example, “the marital status field is disabled when it should be enabled”. The test cases, if executed, will find these defects.

10. Are you willing to make adjustment to the application to accommodate test automation?

Sometimes changes need to be made to an application under test. It could be as simple as being able to navigate properly in a table using the cursor arrows rather than the mouse. It could involve more complexity, for example, having to compile some code into the application to allow for better recognition, or creating an API function to allow the test automation code to interact with the application. In some cases the application needs to be built a bit differently to take test automation into consideration. This does not happen in every case but it can happen.

11. Have you thoroughly investigated all the technical aspects you will need to code in the applications? This includes having the correct tools and strategy on how to code all the components and functions that you need.

This is the part with which everyone has the most difficulty. This is one of the reasons many companies just buy a tool and hope that it will do everything that is necessary. Well, hope springs eternal, but hope, unfortunately, does not write code.

Previously you gathered an inventory of all the application and components that you need to automate. Can the tools interact with all the components in all of your applications? Which tools will do what? How will you integrate various tools to work together in one test automation suite?

As an example, I recently worked on a project where we had XML, a GUI, batches run from UNIX and flat files. All these components had to be integrated to work together. In addition, the GUI was written in an obscure language and could not be recognized by the automation tools. This all had to be designed and then coded.

Since automated testing is a development project it needs proper design. Have you done it?

12. Are you and management fully committed to test automation?

Test automation should be the primary way of testing. But it takes some very hard work to put it into place. When GM builds a new assembly plant, it is fully

automated and they commit the needed time and resources. Management is fully behind this endeavour. They do not put in partial automation and then the rest of the car is assembled manually in the parking lot. The same needs to be true for test automation.

Here are some telltale signs that management either does not get it or is not committed :

- Is the responsibility to test automation given to the most junior testers or to summer students? Imagine Ford saying we will have the automated assembly of our cars be the responsibility of summer students.
- Management will not approve using top notch resources.
- Management is not forcing testers to adapt. Imagine a Ford worker saying you can't build the assembly plant because he has been soldering this piece of the car for 20 years and he doesn't want to change. Ford's answer to this is different then the one often given to testers.

Now that you have read the comments about the questions, you may want to go back and do the questionnaire again. If you answered "No" to any of the questions in scenario B, I would advise you not to proceed until you have handled the issue.

Test automation is an essential and central component of development, but it must be done right with the right people. Anything less will just prove James Bach right.

**Bio:**

*Richard Bornet has been in the software business for over 20 years. The last ten he has spent running various testing departments and creating innovative approaches to improve testing. He specializes in test automation and is the inventor of Scenario Tester and co-inventor of Ambiguity Checker software.*

*He can be reached at [rbornet@eol.ca](mailto:rbornet@eol.ca), or by phone at 416-895-7176*

## ADVERTISING

### Advertising rates

Full page:  
 Single issue = \$350  
 Per issue for 4 issues = \$300  
 Per issue for 8 issues = \$225

Half page:  
 Single issue = \$175  
 Per issue for 4 issues = \$150  
 Per issue for 8 issues = \$115

Quarter page:  
 Single issue = \$88  
 Per issue for 4 issues = \$75  
 Per issue for 8 issues = \$57

Eighth page:  
 Single issue = \$44  
 Per issue for 4 issues = \$37.50  
 Per issue for 8 issues = \$32.50

Business card:  
 Per issue = \$25

To advertise please contact [Tassquarterly@tassq.org](mailto:Tassquarterly@tassq.org)

**Rejection of Advertising**  
 TASSQ reserves the right to reject any advertising.

### How To Submit Articles to TASSQuarterly

TASSQuarterly magazine provides a platform for members and non-members to share thoughts and ideas with each other.

Deadline for submitting articles for next issue due to be published in January is December 15<sup>th</sup>.

For more information on how to submit articles, please visit <http://www.tassq.org/quarterly/>

Please submit your articles by e-mail to [tassquarterly@tassq.org](mailto:tassquarterly@tassq.org). If submitting multiple articles in one e-mail, please make sure that the total size of the e-mail attachment should not exceed 1.0 MB.

# TASSQ<sub>U</sub>ARTERLY

## They Just Don't Get it: Part Three

by Robert Sparkes

*This is the 3<sup>rd</sup> of a series of articles Robert has published starting in April's edition.*

*There isn't enough time or opportunity in one lifetime for each of us to make every possible mistake. The only unforgivable error is not to learn from our own mistakes, and those of others. The 'they' in my title refers to management; not because they make more mistakes than the rest of us, but because they make the decisions which are ultimately responsible for project outcome. In management's defense, those decisions can only be as good as the information they are given.*

Managers typically ask two rather naïve questions which encourage a rush to implementation: "Why isn't Johnny coding yet?" & "Why isn't Jeannie testing yet?"

Management decisions are often tainted by misconceptions, misinformation, or just plain lack of information especially when it comes to understanding the role of testing within the Life-Cycle. Developers can also contribute to this problem because they tend to view their spiffy code as the ultimate end-product. They are anxious to move it off their plate, into production. So Developers tend to have an investment in smooth testing, demonstrating their spiffy code, while Testers consider it a challenge to break that code. Managers may be given the perception that testers are holding up implementation.

Another side to this problem is that programmers are often brought on-stream when coding begins. Similarly, testers do not get involved in the project, until the code is actually delivered for testing. Granted, the full complement do not need to be involved early but Development and Testing Leads must participate in the Requirements, Specification, and Design phases for their own understanding and to provide project feed-back to Users. Developers can ensure that all the requirements are duly captured, then translated into specifications, and incorporated in design. Testers can make sure that functional requirements are stated in a testable fashion.

Managers may try to track progress by measuring how many lines of code have been written, or how many test-cases

were executed, or how many defects were raised and closed. These are very simple metrics to report but they do not provide a useful measure of progress. A 'Line of Code' or a 'Test Case' or a 'Defect' is not some uniform item to be counted. Sometimes they can be quite simple to produce, or execute, or confirm while at other times they can become a very complex. Progress is not a linear function. 80% of code can be written in 20% of the elapsed time. Coding quickly stalls at the "90% complete level," and testing always requires yet another "Cycle" to close outstanding defects.

Completing that last 10% of the code and closing those last few critical defects often causes project over-runs. But premature implementation will result in unhappy client/users and a poor impression of Quality which lingers long after the euphoria of meeting time and budget.

### **Bio:**

*Robert Sparkes is semi-retired after 30+ years in the computer business. He received his basic training with IBM, while majoring in Computer Science at York University, in Toronto. He has worked in all aspects of software development, showing proficiency in different programming languages, over multiple generations of hardware. He has specialized in software testing and methodology with several major Canadian financial institutions including Banking, Insurance & Securities organizations.*

### **ETTORE DELL'AQUILA**



Quality Assurance  
Automated Testing

104 Rochman Blvd.  
Scarborough, Ontario  
M1H 1S2

Tel: (416) 431-4025

E-mail: ENDA\_Ent@rogers.com

# TASSQ<sub>QUARTERLY</sub>

## Book Review Beyond Microsoft PowerPoint: What Public Speaking is Really About

By Barbara Lake PMP, CSTE

### The Seven Strategies of Master Presenters

Dr. Brad McRae  
Career Press, 2004  
285 Pages  
Paperback \$24.95

Like many people in our profession, I have offered my services as a speaker and trainer on numerous occasions for various organizations. It can be both an exhilarating but devastating experience given the feedback you sometimes receive. I remember when I was just getting ready to speak at the International Quality Conference in 2003 and I was preparing by sitting quietly and visualizing a good session with my audience. A fellow presenter, who was also getting ready for his “performance”, came over to speak to me. Rather than sit quietly, he liked to talk to people, I guess to get his mind off of his nervousness. We started to talk about how we both try to make our presentations as best as we can by paying close attention to the feedback we had received in past presentations. My colleague said he was told he moved around too much during his presentation, so the next time he sat down and was soundly criticized for doing that. He said, “You just can’t win no matter how good your presentation is or how many hundreds of hours you spend on the presentation. Most certainly, you can’t please all of the people all of the time.” I felt rather reassured after that short conversation because here was a seasoned speaker and QA specialist who felt just as nervous as I before a presentation and felt just as defeated when reading the negative feedback comments.

So whether you like to sit comfortably in the audience circling those 1’s or 2’s (and hopefully 4’s and 5’s) or you want to lessen your chances of a poor review, you will probably like this book with its wealth of techniques and real life vignettes from world famous master speakers.

In the foreword, Ted Corcoran, President, Toastmasters International 2003-2004, quotes author Roger Ailes as writing, “Today we’re all tuned to receive information much more quickly, and we get bored if things slow down. The

video has sped up our cognitive powers.” Corcoran goes on to say that today’s audiences listen differently therefore; today’s oral communicators must speak differently. Most of us are probably still operating on the tenets of Presentations Skills 101, taken 10 years ago, which emphasized mechanics such as:

- Don’t read your slides
- Put only four bullets on a page
- Have an agenda and “what we want to learn today” pages
- Don’t play with the mouse, the pointer or light pen
- And absolutely don’t put your hands in your pockets!

*The Seven Strategies of Master Presenters* goes beyond the commonsense dictates of presentation courses. It addresses the skills and strategies necessary to be what McRae calls the “Master Presenter”. He defines master presenters as people who can make their presentations memorable, actionable and transferable to the workplace so they have both an immediate and lasting impact. When is the last time you spoke and felt that what you said was memorable at least until the next TASSQ meeting?

Do you still think that putting together a presentation is nothing more than launching that old favourite, MS PowerPoint and cobbling up a bunch of slides? In the corporate world today PowerPoint is probably the most overused piece of software out there. It seems you qualify to give presentations if you admit to knowing PowerPoint and all of its gimmicky slide transition techniques. Great presentations focus on the ability of the speaker to connect to the audience. McCrae calls this “reciprocal rapport”. He says there are four methods a master presenter uses to do everything in their power to build a strong powerful reciprocal rapport with their audience that allows them to develop and deliver a message effectively:

- Increase the audience's level of expectations/aspirations and deliver over and above what the audience expects
- Continually demonstrate their level of expertise and mastery of the subject
- Demonstrate that their methods of delivery are at an "Oscar level" of performance
- Prove to the audience members, beyond a shadow of a doubt, how they will benefit from the presentation.

From there McRae provides the seven strategies that master presenters employ:

Strategy 1: Know Thy Audience

Strategy 2: Prepare Outstanding Content

Strategy 3: Use Superior Organization

Strategy 4: Develop Dynamic Delivery

Strategy 5: Make it Memorable, Actionable and Transferable

Strategy 6: Manage Yourself, Difficult Participants and Difficult Situations

Strategy 7: Total Quality Improvement

Conclusion: The Power of Lifelong Learning.

*I hear, I forget; I see, I remember; I do, I understand.*  
- Confucius, 970 BC

*Strategy 5: Make it Memorable, Actionable and Transferable* reveals some interesting statistics. Research has shown that 24 hours after hearing a presentation, the listener will forget at least 50 percent of all the information presented. In 24 more hours, *another* 50 percent will be forgotten. This means that in a mere 48 hours after hearing a presentation, no matter how attentive the listener is trying to be, no matter how good his notes are, he will forget 75 percent of everything the speaker said. In light of this, McRae presents several techniques he guarantees will enhance the memorability of your presentation such as, repetition and restatement, active vs. passive learning, stories and defining moments.

I just recently finished preparing a presentation for a software testing conference in the fall. I followed McRae's advice by having a "Content Advisory Board" (CAB) review my material. The purpose of a "CAB" is to provide the presenter with objective, pertinent, insightful and crystal clear feedback on where the presenter's content is working and where it is not. The interesting thing was that everyone, whom I invited to review my work, came at it from a totally

different perspective. One person focused on the technical aspects, for example position of the text and slide transition techniques. Another person gave me more in depth historical/political information that helped round out the point of view I was trying to project. One said simply, "What's your point?" This helped me craft a clear focal point for the presentation. I'm now much more confident that the message I am delivering will be clear & cohesive. However, that same "What's your point?" person sent me, *The 10 Do's and Don'ts Presenting with MS PowerPoint*. Was he trying to tell me something?

Another key strategy that I intend to follow involves the benefit of practice sessions. McRae says that most presenters over prepare on content and under prepare on delivery. He quotes the author Bill Bachrach:

"Practice! Practice by videotaping, audio taping or role-playing with friends and colleagues. Be so comfortable with what you are going to say that you don't have to think about it. This frees your thoughts to be totally in tune with your [audience]."

McRae counsels us to test early and test often, to simulate the setting and the audience as closely as possible, to conduct dry runs, to test on mixed audiences and to look at the presentation from a fresh perspective. This last one involves having a neutral outside party look at your presentation, as they might be able to see something you don't.

Sprinkled throughout the book are many real life examples of great speakers and the techniques they use. One such example was how New York's former mayor, Rudolph Giuliani rose to prominence after the tragic events of 9/11. McCrae says Rudolph Giuliani spoke so well that he became known as America's mayor. In his book, *Leadership*, Giuliani presented strategies for making and delivering a dynamic presentation: Five of those points follow.

- Develop & communicate strong beliefs
- Don't save your best argument for last
- Use facts to build your case
- Be able to explain and simplify.

McRae even includes exercises to help you identify your strengths and weaknesses and to develop an individually tailored program to enhance your strengths and overcome areas of weakness.

All of this sounds like a lot of work, doesn't it? Preparing a great presentation, or even a good one, takes a lot of time, discipline and personal reflection. Regardless of whether you are a seasoned speaker or want to do some public

speaking in the future, I think you will find this book of value. The bottom line is most of us volunteer our time because we want to share our knowledge or help people find solutions to common problems. We don't do this for a living and we don't get a fat pay cheque at the end. According to McRae, master presenters have an overriding attribute in common: they are dedicated to lifelong learning; and they have a passion and dedication to learn from every source available, even those dreaded TASSQ feedback forms!

**Bio:**

*Barbara Lake is a project management and quality assurance consultant who has spoken at various venues including TASSQ and the International Quality Conference in 2003. Her next public speaking engagement is in October at the QAI Software Testing Conference in Orlando, Florida. She is currently in a mixed state of exhilaration and fear.*

**Quality Assurance Institute  
Public Campus Seminars**

Nov 7 - Nov 9	Effective Methods of Software Testing - Test Planning	Orlando, FL
Dec 1 - Dec 2	Defining and Using Software Standards	Seattle, WA
Dec 5 - Dec 7	Effective Methods of Software Testing - Test Case Design	Orlando, FL

407.363.1111 ~ Fax: 407.363.1112  
Web: [www.QAIworldwide.org](http://www.QAIworldwide.org)

**Three-day Programs:**  
QAI Member: \$1,195, Non-Member: \$1,295



**NVP Software Testing; your reliable partner  
in Quality Assurance**

- Test Management
- Software Testing
- Test Automation
- Training

NVP Software Testing, [www.nvp-inc.com](http://www.nvp-inc.com), 416-809-5539, [nvp@nvp-inc.com](mailto:nvp@nvp-inc.com)

# TASSQ<sub>QUARTERLY</sub>

*This paper discusses principles of change management requirements and provides some guidelines for identifying, planning and conducting requirements management activities. It discusses how important it is that changes to requirements are carefully traced, analyzed and their effects on the system overall be properly assessed.*

## Abstract

Requirements continuously change. This change is either because of incomplete and non-deterministic character of requirement that is caught as a defect during the reviews or testing stage or because the needs themselves change. While requirements present that dynamic nature, managing this change all through the software development lifecycle has an important impact on the success of the project. That is why when a change is needed, procedures for managing changes should be both flexible enough to allow new improvements, and also rigorous enough to prevent other product development problems.

Although many organizations today have a very well-defined new product/feature development process, on the contrary they have a poorly defined system to manage changes in requirement specifications. This loosely controlled system not only lacks facilitating communication among team members, but also identifying the dependencies between the requirements and the other software artifacts.

By implementing a process or set of principles, changes in requirements can be managed in a more controlled manner, obtaining a much more accurate picture of the effort involved for the change and potential impact of this proposed change.

This paper is intended to discuss principles of change management in requirements and provides some guidelines for identifying, planning and conducting requirements management activities. The audience of this document is project managers, business analysts, quality assurance person and other practitioners that are in charge of writing, validating and verifying requirement specifications.

# Managing Requirements Changes in SDLC

by Murat Guvenc

## Introduction

Software Development Life Cycle (SDLC) is the overall process of developing information systems through a multi-step process from exploration of initial requirements through analysis, design, implementation and maintenance. Various SDLC methodologies have been developed to guide the processes involved, including the waterfall model, rapid application development (RAD), joint application development (JAD). Frequently, several models are combined into some sort of hybrid methodology.

The development of large, complex systems presents many challenges. The most important ones among those are the ability of the system to ensure that the final product satisfies the needs of the customers and the capability of it for easy maintenance and enhancement during their deployed lifetime. While systems frequently change and evolve throughout their life cycle, it makes it difficult to ensure that the implemented system validates the original user requirements.

According to a survey done by the Standish Group, 31% of software projects are cancelled before ever reaching the customers while 53% cost 189% or more of their original budgets. 40-60 % of software defects and failures are attributed to bad requirements. This reflects the area of the problem the companies face in identifying, communicating and managing project requirements. The capability to clearly define a user requirement is critical to the overall success of a software project.

The root cause of software development problems can be listed as follows;

- inaccurate understanding of the end-user needs
- inability to deal with change in requirements
- poor communication across team members
- lack of build and release process
- poor performance

Current change management techniques have focused largely on design and code level artifacts, rather than requirements level entities. The main reason for this is that the artifacts from the later stages of development are more concrete and provide developers with more of the correct

type of information required for change management. Ignoring requirement change management often leads to systems that fail to meet the real business needs.

Because requirements are changeable throughout the software development lifecycle, and because requirements specification is a dynamic process, requirements are not often complete until the end of the product implementation. The number and scope of changes to the requirements decrease as the project reaches the later stages of the design effort, but the changes never entirely stop until product release. Simply freezing the user requirements at an early point in the development process is practically not possible to enforce. Change in requirements specification is inevitable.

Therefore, procedures need to be in place for making these changes. Implementing and managing changes in a planned and systematic fashion is essential to the success of the projects. The result will enable the project managers to properly set customer expectations and to meet on-time on-budget project demands.

In the scope of this paper, CaliberRM is used to demonstrate how managed requirements help trace the evolution of the project throughout the development life cycle. Borland's Enterprise Requirements Management System CaliberRM is designed to facilitate collaboration, impact analysis, and communication in the definition and management of changing requirements. CaliberRM helps teams manage both the expectations and scope of projects, improve product quality and reduce the risk of project failure.

### Benefits / Outcomes of CaliberRM

- **Visibility** - Requirements can be viewed, sorted and filtered on an individual basis
- **Reusability** - Requirements can be re-used from one project to the next
- **Testability** - Each requirement can have its own verification and validation criteria
- **Traceability** - Each requirement can be traced from inception to deployment in the delivered system
- **Maintainability and Security** - Each requirement can have its own individual change history and level of security
- **Full Characterization** – Several pieces of information of various data types can be stored for each requirement
- **Repeatability** - Requirements management contributes to a predictable and repeatable process that can be used toward the achievement of various standards-based compliancy objectives

## Requirements Change Process

Change is inherent in the development of most software systems. Software requirements change and evolve from inception to deployment. Although the majority of the requirements efforts are performed at the beginning, they often continue just before code freeze. These changes can affect both system functionality and the wider business goals of the organization for which the system is developed and as well the project cost, resources, and schedule. When a change is needed in a requirement, there might be many artifacts affected by this change such as design components, test cases, and source code. For these reasons it is important that changes to requirements are carefully traced, analyzed and their effects on the system overall is properly assessed.

Objectives for managing changes to requirements can be summarized as:

- Identifying the rationale of the changes in the requirements specification
- Identifying responsible parties for the change
- Tracking change history in the requirements specification
- Applying impact analysis on the effect of the change
- Communicating the change among team members
- Reporting changes in the requirements specification

The goal of requirements management within an organization is to provide consistency, predictability, and repeatability of the activities. Guidance should be provided for controlling costs, prioritizing requirements, and standardizing requirement analyses methods, so that requirements can be effectively engineered throughout the project's and product's lifecycle within the context of the defined project management activities.

The requirements change process should include the following activities to be carried out when a change is needed in the requirement.

- Allocating adequate resources (Assign responsibilities)
- Analysis of requirement changes (Manage changes)
- Documenting requirements (Document rationale)
- Creating requirements traces throughout the project lifecycle from inception to the final work products (Trace requirements)
- Establishing team communication (Communicate the change)

- Establishing a baseline for requirements specification (Establish baselines)

### Assign responsibilities

In each project, there are designated individuals, with the role of baseline administrator, project managers or project leads that have the administrative rights to maintain the baseline by changing a version of the requirement or adding/removing requirements from it based on the changes required. These individuals take the leadership to assess the change requests and accept the change. After a change is approved in baseline, these individuals assign certain individuals the task of carrying out the appropriate modifications.

### Manage changes

Managing requirement changes is an activity to identify, analyze, track, and report proposed changes and finally approve those changes to the product specification. As project evolves, requirements may change or expand to accommodate modifications in project scope or design. When there is a request to add a new feature to the product or to enhance an existing specification due to a defect or failure, a change request is created to modify the existing requirement specification.

Those changes to the requirements can impact the project overall cost, resources allocated, and schedule planned for the delivery. Setting the requirements at the appropriate hierarchical level and establishing priorities will help to ensure the success. Proper assessment should be done through content review by certain individuals or representatives of different groups prior to finalizing the change request. Once requirements are updated, a formal notification of the change is sent by e-mail.

### Document rationale

Documenting requirements is an iterative process that embraces establishing requirements specification artifacts (prototypes), identifying representation format (how information is going to be captured and represented) and validation criteria (how each requirement will be verified). Representation formats may include other formats rather than simple text, like graphic files, screen shots, short cuts to certain context embedded in a third-party application or anything else that might be of use in defining or understanding requirements.

### Trace requirements

Key to the success of any requirements management process is requirements traceability. Requirements traceability is a

technique used to provide relationships between requirements, design and implementation of a system in order to manage the effect of change and ensure the success of the delivered systems.

There are many relationships that exist between requirements, design, components and others. Managing these relationships is critical to providing a comprehensive requirements management capability supporting the system engineering life cycle.

When the requirements change, the impact should be evaluated on the analysis, and then the design model. When the change impact has been determined, then updates to the project schedule should take place. There should be a decision making structure in place to review the changes and determine what to do.

It should be ensured that any piece of design work can be traced to one of more requirements in the project. If the developers are building something that cannot be traced back into a requirement, it indicates either that there are potential issues with the accuracy and goodness of the requirements or else that the design team is building something that is outside the scope of the system.

Also it should be ensured that every path through the system has a test case associated with it. A classic symptom of a project in trouble is lack of coherent test specifications. Testing is often considered as a boring, tiresome activity for many developers. As such it is often overlooked; however, lack of testing is one of the major factors in failed and in trouble projects.

### Communicate the change

The most troublesome changes to the requirements specification have been the ones that are not communicated to the entire group. Communications failures typically occur when developers either drop a feature or change a performance requirement without telling the rest of the team. Establishing an automated email-notification system offers instantaneous team communication and it is critical that all affected parties are notified of the change in a timely manner.

### Establish Baselines

After the requirements specifications have been verified and all the reviews are successfully completed, it will be approved by the customer and stakeholders, and then it is ready to be baselined. Baseline is the tested version of a set of requirements representing a conceptual milestone, serves as the basis for further development and that can be modified only through formal change control procedures. A



# TASSQ<sub>QUARTERLY</sub>

*This article was originally published in Better Software Magazine, January 2005.*

*Not all testing need be tied to specifications. Using a heuristic approach to testing, this article demonstrates how exploratory testing can be valuable in addition to specification-driven testing. James Bach's oracle heuristics (plus one) are presented as the bases for discussion.*

Sometimes when faced with an unfamiliar application and a directive to test it, it can feel as if you've been asked to chart a course through unknown waters. You're not sure what you'll find on the voyage, but you've got experience, some notions of what to look for, and an idea of where you'd like to go. If you see something surprising or unexpected, you'll take note of it. Those are exploratory skills, and sometimes they're all you need to begin.

Some testing advocates suggest that you should never test without a "complete written specification." That's unrealistic advice. First, there are *plenty* of contexts in which you may be asked to test without a formal specification: When you're evaluating a piece of commercial software to see if it suits your company's needs; when your organization's production code is so old and so heavily patched that the original specification would be meaningless—even if it could be found when working on Agile projects. Second, "completeness" is entirely dependent upon perspective and context. Even so-called "complete" specifications contain much that is implicit. In fact, the more explicit the document, the longer and more ponderous it is, and the less likely that someone will read it in its entirety. Finally, certain kinds of specifications might be supplied to you through some means other than a formal document—conversation, email, or your own inferences. A quick meeting with the boss, combined with your skills at identifying value, risks, and problems, might give you all the charter you need to begin supplying useful information to management.

An effective tester can always obtain valuable information by exploration, even if the sole purpose of exploring is to gather information for a more detailed test strategy.

## Prepare for the Journey

When the explorers of old set sail for uncharted waters, they did not set out unequipped. They knew the sun and the stars,

## Testing Without a Map

By Michael Bolton

and they carried tools such as compasses, sextants, and clocks, not only for navigation but also for mapmaking. More importantly, they ventured out with extensive background knowledge and skills, which included deduced reckoning, celestial navigation, and horse sense. For exploratory testers, knowledge is often represented in two terms that you'll use as a testing expert: *oracles* and *heuristics*. An oracle is a principle or mechanism by which we can tell if the software is working according to someone's criteria; an oracle provides a right answer—according to somebody. A heuristic is a provisional and fallible guide by which we investigate or solve a problem; it's also a method by which learning takes place as a result of discoveries informed by exploration. James Bach has given us a helpful set of oracle heuristics, to which I've added one, in the form of this mnemonic: HICCUPPS. The idea is that a product should be consistent with:

**History:** The feature's or function's current behaviour should be consistent with its past behaviour, assuming that there is no good reason for it to change. This heuristic is especially useful when testing a new version of an existing program.

**Image:** The product's look and behaviour should be consistent with an image that the development organization wants to project or that the purchasing company wishes to present to its internal users. A product that looks shoddy often *is* shoddy.

**Comparable products:** We may be able to use other products as a rough, de facto standard against which our own can be compared.

**Claims:** The product should behave the way some document, artifact, or person says it should. The claim might be made in a specification, a help file, an advertisement, an email message, or a hallway conversation, and the person or agency making the claim has to carry some degree of authority to make the claim stick.

**Users' expectations:** A feature or function should behave in a way that is consistent with our understanding of what users want, as well as with their reasonable expectations.

**The Product itself:** The behaviour of a given function should be consistent with the behaviour of comparable functions or functional patterns within the same product unless there is a specific reason for it not to be consistent.

**Purpose:** The behaviour of a feature, function, or product should be consistent with its apparent purpose.

**Statutes:** The product should behave in compliance with legal or regulatory requirements. Remember: Heuristics are guidelines, not edicts; they're fallible. They aren't universal; there are plenty of other ways by which we can decide whether a product is acceptable or unacceptable. There is some conceptual overlap between some of the points, but to an explorer, features of the new territory overlap, too.

## Explore and Discover

Armed with these tools, let's imagine that I'm working for a small start-up, and that my company is going to be releasing its software on CDs. The company doesn't have a lot of money, and every dollar counts. My boss has asked me to evaluate the program that comes with the CD burner, the popular Nero CD recording software. Printing CD covers is a requirement; therefore, she asks me to have a look at Nero's Cover Designer, a subset of the CD recording package, to see whether the company should use it. Instead of writing up an elaborate test plan, I'll just plunge in, quit when I have more information, and then (and only then) make some decisions about how to proceed. Figure 1 shows what I see on the main screen just after I start the program.

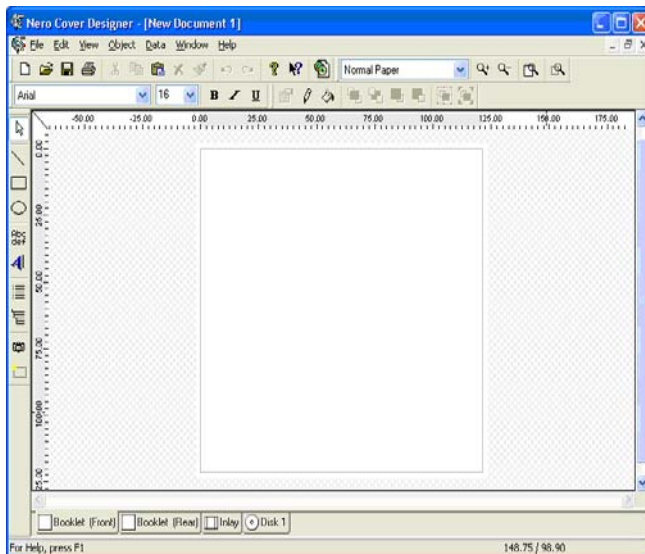


Figure 1

Yogi Berra was right; you can observe a lot just by looking. One of the first things I note is that there appears to be a default setting for the font, Arial for the face and 16 for the point size. My boss doesn't need to tell me to test fonts; I have the consistency with purpose heuristic in my head to tell me if printing CD covers, graphics and text — and, therefore, fonts — are likely to be part of that task. Do I care about the accuracy of the point sizes and color depth of the

graphics? Maybe, but I can ask about those things later, after I've run some other tests. I make a note to ask questions about accuracy and move on.

I'm going to need to put something on my CD cover, so I choose to insert a new object. I click Object, Insert, Text Box. Then I double-click the new object that appears, and the dialog shown in Figure 2 pops up. Something already feels funny. In the new font properties area, the name of the font has disappeared and the point size now appears to be 8. In accordance with the consistency within the product heuristic, one would think that the font properties should be the same on both the main screen and the new dialog. Do we have our first bug? I'd say "Yes", but perhaps we should do some checking. I'll note it.

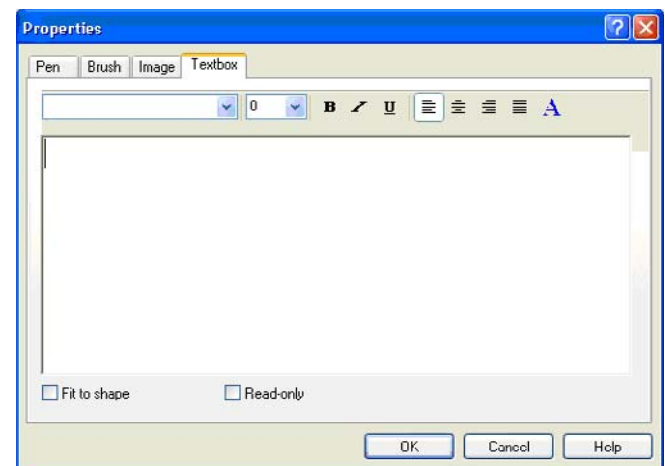


Figure 2

## Investigate New Findings

We don't have a specification, but Windows programs typically come with a Help file. A program should be consistent with claims it makes about itself, and the Help file is usually full of claims about what the program can do. So let's press the F1 key.

Why F1? Windows users have a heuristic that F1 should trigger the Help file, courtesy of the Windows User Interface Guidelines. Cover Designer is a Windows program, and a program's behaviour should be consistent with programs like it. If there is a compelling reason for your program to behave differently, then it might be worthwhile to depart from de facto UI standards. Otherwise, consistency with other products is a favour to the user, saving her the time and trouble of learning a different way of doing things.

When I press F1, a tooltip appears at the hot spot on the mouse pointer: The tooltip says,

Lets you modify the contents of the text.

Shouldn't that say, "Lets you modify the contents of the text box"? That might be a quibble, but in some contexts I'd be willing to call it a quibble, but in some contexts I'd be willing to call it a second bug. If this were my program, I might find the imprecise English a little embarrassing, which would violate the consistency with image heuristic. A program should be consistent with the image that a company wishes to present. And, another thing. Shouldn't F1 display the Help dialog instead of a tooltip? By Windows conventions, a tooltip should appear when you hover over an item with the mouse. I'll write a couple more notes about these Help issues; they might represent another bug or two.

I want to open the Help file. There's another way to do that. I can click the Help button to open it, click the Find tab, and find all the references to "text box." (See Figure 3.)

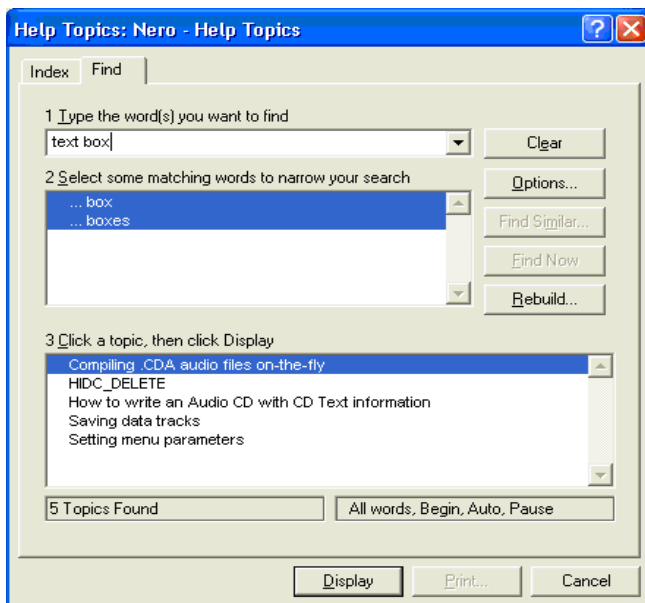


Figure 3

Hmmm . . . there's nothing here that looks like a reference to text. In fact, there's nothing here that seems to refer to anything in the Cover Designer. Let's go to the Index and look for the words "Cover Designer." All I see is Help for the Nero CD-ROM burning software. That means that either there is no Help for Cover Designer, or if there is a Help file, it's not coming up from inside Cover Designer. That's a problem based on the consistency with user expectations heuristic— a user could reasonably expect that a Help file summoned from within an application should be that application's Help file.

Well, it seems as though I'll have to give up on Help, and that's noteworthy. Let's return to the first presumed bug and do some more investigation. I don't know exactly what my company is going to put on the CD cover, but the specifics don't matter, so I'll put in some text that reflects the way that I might use the program. (See Figure 4.)

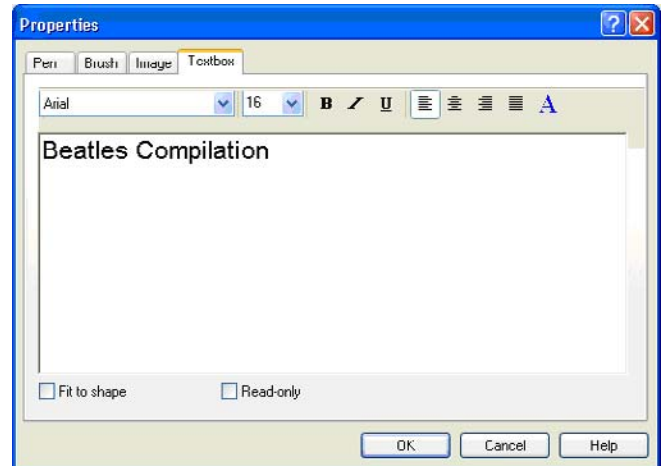


Figure 4

When I type "Beatles Compilation"— in fact, immediately after striking the B key—the font size of 8 turns to 16, and the formerly blank drop-down for the typeface is suddenly set to Arial. The *consistency-with-the-user's-reasonable-expectations* heuristic suggests that typing some text should not change the selected font unless I've asked to do so. Even though this rectifies the problem I noted as the first bug, it does so in a way that gives me pause, and this is arguably yet another bug; I'll write that down. I'll highlight the text that I've entered and choose a different typeface and size; again, specifics don't matter. I'll select Comic Sans MS and 26 points. (See Figure 5.)

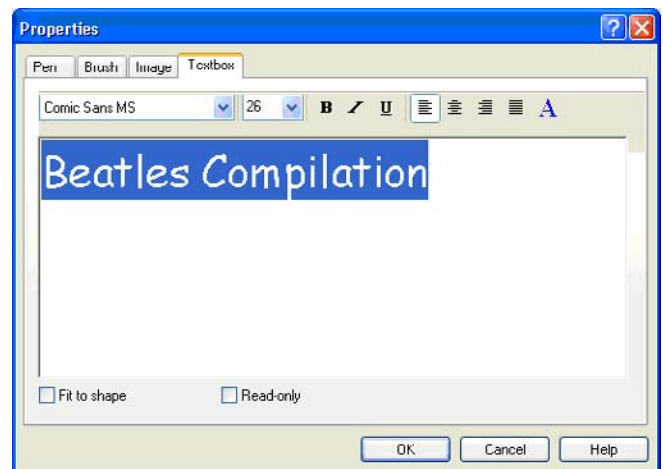


Figure 5

Then, I'll press OK to close the dialog. Now, I'll immediately click the text box to open the dialog again. (See Figure 6.)

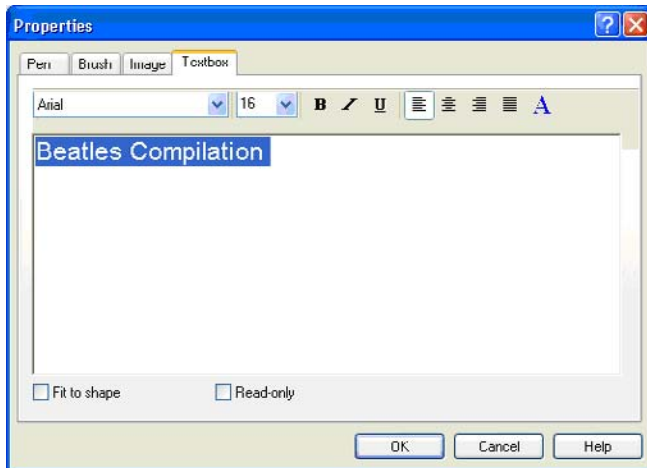


Figure 6

Presto! The typeface is back to Arial, and the size is 16. This violates the *consistency-with-purpose* heuristic. Surely the purpose of pressing OK (rather than Cancel) on an object is to retain the properties that I've selected until I explicitly change them: *A feature or function should be consistent with its apparent purpose*. Another bug to note.

Note that in this dialog there are tabs for pen, brush, and image as well as text. I try this out, and I find that every time I try to reopen the text box to modify one of these attributes, the font information disappears—an inconvenience and an annoyance and, even without a specification, manifestly a bug. I'm disappointed because I seem to remember this feature working in a previous version of Nero Cover Designer. That's a violation of the *consistency-with-history* heuristic. A program should behave in a manner consistent with its own history or previous versions of the product.

The bugs in this program have been pretty easy to find, and this last one is so troublesome that I have some grave doubts about the rest of the program. After five minutes of testing, I'll be able to tell the boss that she should not rely on this product to produce the company's CD covers—and, thank goodness, I didn't waste time preparing an elaborate test plan based on some incomplete specification that some programmer apparently didn't read.

### The Journey Ends

This is a particularly egregious example, but if you're still adamant that you need a written specification before you

can begin testing, consider what you've just read in the context of two questions. First, did we need a written specification to provide important, credible, timely information to management? Second, would the cost of researching and preparing a specification—and waiting for it to be prepared—add significantly to the value of our report? As you can see, in many contexts it's not only perfectly OK but also entirely desirable to test without using a specification. My background knowledge of GUIs on Windows helped me recognize several problems, and my ability to put myself in a user's shoes helped too. A few minutes of exploration, wandering through one feature of the program and looking through the spyglass of those exploratory testing heuristics, has helped me not only to find bugs but also to identify credibly *why* I think they're bugs, even though I had nothing like a complete, formal, written specification. Although I didn't have a map, I was certainly able to explore and compile one along the way.

### Bio:

*Michael Bolton lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries, as part of James Bach's Rapid Software Testing course. Michael is also the Program Chair of TASSQ.*

*He thanks James for his review of this article.*

### QAI's 26th Annual Software Quality Assurance Conference

#### *Use It or Lose It... Exercising Your Quality Muscle*

April 24-28, 2006 ~ Orlando, FL

407.363.1111 ~ Fax: 407.363.1112

Web: [www.QAIworldwide.org](http://www.QAIworldwide.org)

### Project World and Business Analyst World

May 8-12, 2006  
Conference in Toronto

<http://www.projectworldcanada.com/>  
<http://www.businessanalystworld.com/>

Contact [info@projectworldcanada.com](mailto:info@projectworldcanada.com)  
or 905-948-0470 ext 228  
or 888-443-6786 ext 228



presents

***Rapid Software Testing***  
***A One-Day Workshop in Toronto***  
***November 29, 2005***

**James Bach with Michael Bolton**

TASSQ, the Toronto Association of Systems and Software Quality, is pleased to welcome to Toronto the internationally renowned testing expert and author James Bach. Mr. Bach will present a one-day overview of Rapid Software Testing, a complete testing methodology designed for a world of barely sufficient resources, information, and time.

Based on the principles in the book Lessons Learned in Software Testing: a Context-Driven Approach, this workshop outlines an approach to testing that begins with personal skill development and extends to the ultimate mission of software testing: lighting the way of the project by evaluating the product. The philosophy of rapid testing presented in this class is not like traditional approaches to testing, which ignore the “thinking” part of testing and instead advocate never-ending paperwork. Rapid testing isn't just testing with a sense of urgency, it's mission-focused testing that eliminates unnecessary work, assures that everything necessary gets done, and constantly asks what testing can do to speed the project as a whole.

The workshop is aimed at any tester, test manager, developer, or anyone else who is interested in learning how good testers think. This course is an introduction to basics of Rapid Testing. Those interested in detailed knowledge of the approach should consider the Rapid Testing Techniques and Rapid Test Management courses (3-days and 2-days respectively).

**James Bach** (<http://www.satisfice.com>) is a pioneer in the discipline of exploratory software testing and a founding member of the Context-Driven School of Testing. He is the author (with Kaner and Pettichord) of *Lessons Learned in Software Testing: A Context-Driven Approach*. Starting as a programmer in 1983, James turned to testing in 1987 at Apple Computer, going on to work at several market-driven software companies and testing companies that follow the Silicon Valley tradition of high innovation and agility. In 1999, James founded Satisfice, Inc., a tester training and consulting company based in Front Royal, Virginia.

**Michael Bolton** (<http://www.developsense.com>) teaches James Bach's Rapid Software Testing all over the world, writes a regular column for *Better Software Magazine*, and is Program Chair of TASSQ.

**Date:** Tuesday November 29, 2005  
9:00 AM to 5:00 PM

**Location:** Sheraton Centre Hotel  
121 Queen St. W., Toronto

**Registration Deadline:** November 15, 2005

**Visit:** <http://www.tassq.org> for details & registration

**Don't miss James Bach's address, Skilled Testers and Their Enemies**  
**Tuesday, November 29, 2005 7:00pm University of Toronto**  
**Visit <http://www.tassq.org> for details**