

# TASSQUARTERLY

VOLUME 3, ISSUE 3  
July 2005



## Future of Testing

*The future of any industry involves the introduction of new tools and adjusting processes, leading to more and more automation of tasks. This is true for the car industry and it is true for IT. This article focuses on the changes to the development life cycle where processes involve the increased use of tools and a move to widespread automation to allow for more comprehensive testing right throughout the life cycle.*

# TASSQ<sub>QUARTERLY</sub>

## FEATURES

### 4 **The Future of Testing.**

Richard Bornet

### 14 **They Just Don't Get It: Part Two**

Robert Sparkes

### 15 **Managing Your Environments**

Joe Larizza

## DEPARTMENTS

<b>Editorial</b>	Page 3
<b>Conference Watch</b>	Page 12
<b>James Bach: A TASSQ Event</b>	Page 17
<b>How to Submit Articles to TASSQuarterly</b>	Page 18

## 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>QUARTERLY</sub>

## *Editorial*

TASSQuarterly began as a forum dedicated to quality and intended to benefit the Toronto Association of Systems and Software Quality membership (TASSQ). Since its first issue of January 2003, ideas in this magazine have been presented as an attempt to reach out to TASSQ members who are not able to attend our monthly dinner program.

I would like to take this opportunity to officially thank Hassanali Namazi – the individual who has made this magazine a reality. Hassanali has done an amazing job as Managing Editor and without his diligent efforts, TASSQuarterly would not exist.

I would also, like to take the opportunity to thank Tiiu Martin for all contributions during her term as president of TASSQ. Believe me when I say that without her efforts, TASSQ would not be here today. Tiiu has invested an enormous amount of her personal time and energy to the success of TASSQ and the TASSQ Board is very thankful for her continued involvement in the up and coming year.

In this issue, we continue our recent discussions on the topic of Software Delivery Optimization. Richard's article "New Development Life Cycle" continues to explore this subject. I have personally found over the last few years that IBM, Microsoft and Borland (just to name a few vendors) are pitching different solutions to IT organizations. Each vendor solution is geared or focused to certain aspects of the development life cycle. We as quality assurance professionals need to fully understand the objectives and goals of integrated solutions – you will soon be asked to travel down this "quality" road. Richard's article facilitates your understanding of all the **issues** and guides you through your journey.

Over the summer months the TASSQ Board will continue to meet and set the direction for 2006. The Dinner Program schedule for next season will be posted on our web site (TASSQ.Org). We encourage individuals to visit the site during the summer months and to share this information with individuals, associates, and peers who have not attended a TASSQ event.

Remember. What you put into the Association is what you can expect to get out. We are counting on *you* to promote and help support TASSQ.

Last but not least, TASSQuarterly is requesting your help! We invite you to submit articles and encourage you to share your knowledge or quality journeys with your peers to the continuing success of the journal.

**EDITOR-IN-CHIEF**

Joe Larizza

**ASK TASSQ EDITOR**

Fiona Charles

**BOOK REVIEW EDITOR**

Michael Bolton

**MANAGING EDITOR**

Jeanette Mount

**CONTRIBUTING WRITERS**

Joe Larizza  
Michael Bolton  
Robert Sparkes  
Richard Bornet

**ART DIRECTOR**

Nuree Hwang

**PRODUCTION**

Jeanette Mount

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

*Joe Larizza*  
TASSQ President

# TASSQ<sub>QUARTERLY</sub>

## A New Development Life Cycle Model to Improve the Quality and Efficiency of Testing

by Richard Borneot

*This article argues that progress involves the introduction of new tools and adjusting processes, leading to more and more automation of tasks. This is true for the car industry and it is true for IT. This article focuses on the changes to the development life cycle where processes involve the increased use of tools and a move to widespread automation to allow for more comprehensive testing right throughout the life cycle.*

*The article also provides a glimpse of the future where once requirements are finalized they can be automatically tested through automation without many of the intervening steps that testers need to currently carry out*

Every development project has a development life cycle. This is true whether the project aspires to a higher CMM (Capability Maturity Model) compliance level uses RUP (Rational Unified Process) or (EP) Extreme Programming, or just tries to get the software out as fast as possible using some home-grown methodology.

There are many different development life cycle models, however they each usually boil down to five stages:

Requirements -> Design -> Develop-> Test -> Deploy

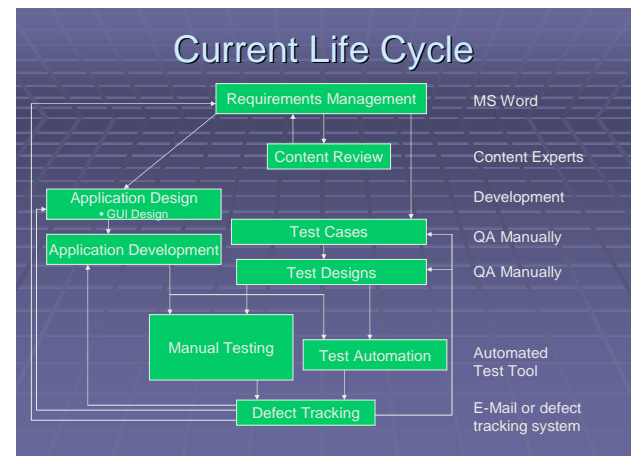
Sometimes development projects do these stages sequentially, which is known as the waterfall approach. Other organizations do this in a more iterative fashion, where they create some requirements, design, develop and test, and then gather more requirements. This is the basis for RAD (Rapid Application Development).

Even in the most disorganized project, what people do and how they do it is governed by a development life cycle.

This development life cycle can be summarized in the figure "Current Life Cycle".

The requirements are written, often in MS Word, and reviewed by contents experts. QA manually develops Test

Cases and Test Designs from the requirements. The application and the GUI are designed, and the application is developed from the requirements. Then QA does manual or automated testing on the developed application. Defects are tracked with a defect tracking system or just with email.



This current development life cycle may be implemented fully or only partly. For example, a project may not use an automated test tool, or defects may not bring about changes to the test cases or requirements.

The question now is "What changes will take place in the future and what role will testers play?" The simple answer is that testing will play a much bigger role and so will automation.

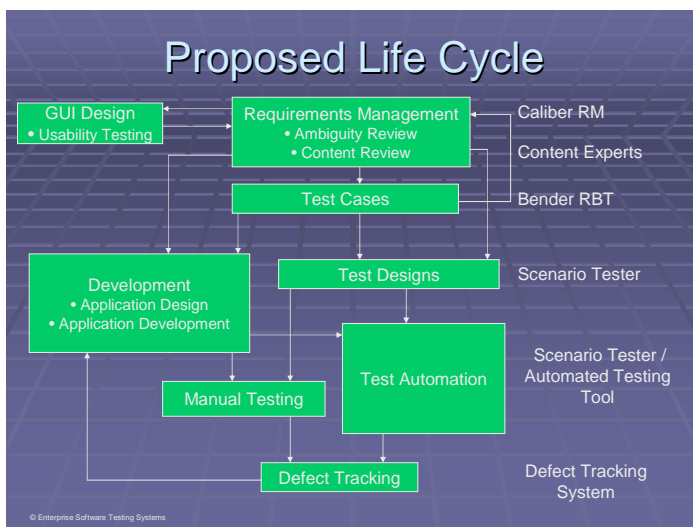
There is plenty of evidence for this, look at any industry that has needed to increase quality and efficiency. The products these industries produce have not only improved but in most cases are cheaper to produce. Whether the products are cars, televisions or computers, the companies have achieved improvements really by doing two things: they improved their processes and they introduced new tools and major automation.

The author believes that increased efficiencies for developing better quality software will be achieved in

exactly the same way. The introduction of restructuring processes, the better use of tools and better automation are the way to increase quality and efficiency.

This article is about the author's and his co-workers' attempts to increase quality and efficiency. The changes we propose involve re-designing some of the processes used by teams, using some best of breed software tools, and implementing full and thorough automation. Some of the tools that were needed were available, but in other cases there were no tools on the market to meet our needs, and we were forced to invent them

This adjusted process can be summarized in the following figure:

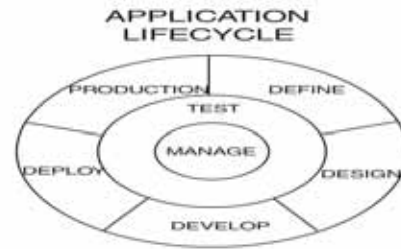


In this life cycle, the requirements are not just reviewed, they are tested, and these tests lead to test cases. The GUI design is usability tested. Manual testing is replaced by automated testing to a great degree. Rather than relying on MS Word or Email, proper tools are used.

The most important difference between this life cycle and the current life cycle is that testing becomes an integral part of each cycle.

Previously testing was a stage between development and deployment. In this model testing plays a much more central role.

- Requirements are gathered, and they are tested
- GUI's are designed and then tested
- Code is written and tested.
- The application is deployed and the deployment is tested.



This article will now look at this life cycle and discuss the processes with an emphasis on tools and automation.

## GUI (Graphical User Interface) DESIGN

In reviewing the proposed life cycle, we start at the beginning with the most important aspect of any software application:

- Does the software meet the needs of the users?
- Can they do their jobs more efficiently and with measurably superior results?

In most software projects, this means getting the GUI *right*. GUI designs that do not meet the users' needs invariably cause the users to use the software inefficiently, or even demand that it be replaced. Often, when software is replaced, no one has pinpointed that the problem was the GUI. The new application is implemented with an equally poor GUI and the life cycle repeats itself. Poor GUI designs may benefit some people through job security, but they do not benefit the organization. They force a resource and cost drain on the organization because applications do not produce scales of efficiency and are constantly being re-designed.

The first stage of the new proposed deployment life cycle is to work on the GUI. This is done at the same time that the requirements are gathered. The relationship the user has with the requirements is through the GUI. Users do not really care about what goes on behind the scenes, as long as they can do what they need to do. Most of us do not care what the background technology is and how it works for our TVs or our cars, as long as it works. It is the same for software.

In the author's experience, nearly all development projects spend inadequate time on designing the right GUI. They also use the wrong approach where they quickly delve into screen design without first working out the overall look and feel and flow of the GUI. This can result in an application where the screens are well designed, but the application actually does not work for the user.

The steps to take to design the right GUI are the subject of a future article. For this article, let us concentrate on describing what a good GUI design methodology can do for us.

A good GUI design methodology produces an application with a high degree of usability. That means that we know clearly up front what the application will look like and how it will function. The screens have been designed in detail, the navigation flow is worked out and not one line of code has been written.

This has the following advantages in the development life cycle:

- During the requirements gathering stage: Having a good GUI design helps clarify requirements, points out missing requirements and makes sure we can represent the requirements so the users can do their jobs. After all, the requirements describe what the application is supposed to do, but the GUI describes how the user will actually do it. As an analogy, requirements describe what a driver should be able to do while driving a car, but the GUI design is how the driver will do it, i.e., which controls the driver will use and how the driver uses them.
- During the development stage: The programmers know up front what they will code in terms of the look and feel of the application. GUIs can sometimes be coded at the beginning, before all the other pieces are put into place.
- During the testing stage: The testers create Test Cases, which are based on requirements. Test Cases define what will be tested. Knowing what the GUI looks like allows the testers to create Test Designs. Test Designs are how the application will be tested.

Additionally, much of the test automation can be built early on, before the application is even coded, if the GUI is known. This is discussed in more detail below.

## REQUIREMENTS

Good requirements are at the heart of any good development project. Getting the requirements right, from the start, saves a great deal of time, and avoids huge pitfalls later in the development. The trick is to gather requirements in an organized and efficient manner.

In the author's experience, most requirements usually follow this path:

- The requirements are written using Microsoft Word.
- There is some type of review process where people are supposed to give feedback on content.
- The requirements are given to the development team who try to follow them.
- Quite often, the development team has to embellish or change the requirements because they were incomplete or incorrect. That means that the requirements writers didn't get the requirements right in the first place.
- Seldom are the documents updated with all the changes.
- Therefore, going forward, there is a disconnect between what is documented and what is in the application.

As the software goes through more and more iterations, this disconnect grows larger, and the organization loses an accurate reflection of what the software should do. Increasingly, the answers are no longer in the documents but in people's heads, or in emails, or in the code.

We ask the reader to accept the idea that getting the requirements right, at the beginning, is better than adjusting them as we go along.

If this is true, then how does one do it?

The following is our approach to solving this problem. Not every project with which we have been involved has employed all the methods listed below, but every one of these processes has given major benefits wherever it was used.

### Deterministic requirements

It is a marvel of language that there are so many different ways to say something. Having read many specifications, we have seen a myriad of ways of expressing a requirement. Requirements should be written in such a way that all sorts of individuals can do their jobs. They should not be written like a novel, where the purpose is to entertain the user with a fascinating story or beautiful language. They should be written as a set of instructions to follow.

Deterministic requirements are the most efficient way of presenting a requirement. A deterministic requirement is one where an exact cause produces an exact effect.

For example, the requirement "old age pensioners should get a larger discount" is not a well-written requirement. The deterministic version might say "If the user's age is 65 or over, the discount is 5 %, otherwise the discount is only 2%."

Deterministic logic helps programmers and tester do their job.

## Ambiguity Reviews

Ambiguity Reviews are based on the work done by Richard Bender. Once the requirements are written, they need to be checked for two things:

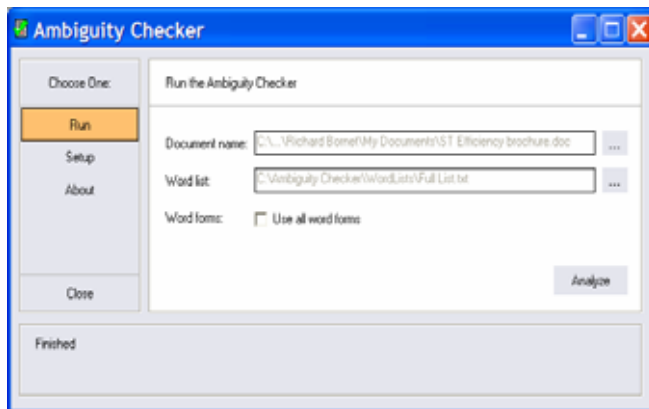
- Do they make sense and hold together (Ambiguity Review) and
- Is the information accurate (Content Review).

Ambiguity Reviews are done first, followed by a Content Review. There is no point in reviewing the content if the requirements are full of fuzzy logic and missing specifics.

The Ambiguity Review concentrates on how the document is phrased. For example, it looks for Ambiguous words such as the word “might” – “the interest rate might be 5%”. When? Under what conditions? It looks for incomplete statements where not all the information is provided – “if the login succeeds then go to the main menu”. What if the login doesn’t succeed?

The Ambiguity Review Checklist identifies 15 specific ambiguous conditions that need to be checked for in any requirements document.

Proof reading requirements documents for ambiguity is a very tedious and difficult task. After reading 20 pages of such a Word document, most people have a hard time concentrating enough to catch ambiguities. There was a need for a tool which could assist in pinpointing ambiguities and making the task simpler.



We could not find such a tool on the market so our team wrote a utility, called the Ambiguity Checker. It takes a Word document and highlights all potential ambiguous statements. In addition, it gives a report on what types of problems have been found. Borland’s CaliberRM, a requirements management tool, also has the ability to do ambiguity reviews, but these are only against requirements in CaliberRM, not in Word documents.

Ambiguity Checker allows us to identify the ambiguities in Word documents and remove many contentious issues.

## Content Reviews

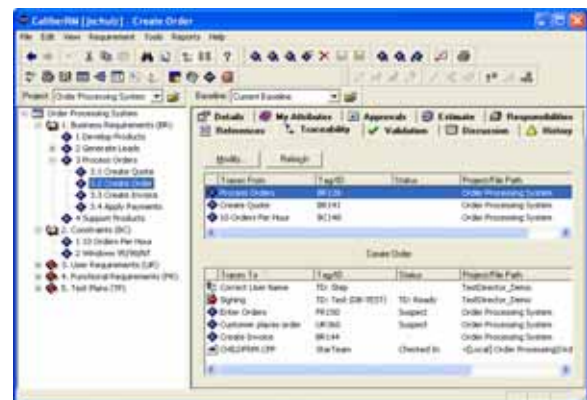
Content reviews check whether the information in the requirements is correct. For example, there is a rule that people over 65 years old will get a 5% discount. Is this true? Content reviews are primarily done by subject matter experts, people who are highly knowledgeable about the business that the software represents.

## Requirements Management Tool

Most people know Microsoft Word, which is why it is the most commonly used requirements management tool. This does not mean that it is the best tool for the job. Most people can walk, but that does not mean that walking is the most efficient and effective way of getting between two points. A good requirements tool is the equivalent of getting a car where previously we could only walk.

A good requirements tool has many useful features.

- Tracking Changes: All changes to a requirement can be tracked, who made the change and when.
- Version Control: All the requirements can be kept for a version; then, they can be changed and added to for the next version.
- Traceability: The tool shows which code is affected by a change in requirements, and which requirements are affected by a change in the code.
- Ambiguity Review: The tool tests the requirements for Ambiguity.
- Discussions: All requirements can be annotated and discussed, and a record of the discussion is kept.
- Notification: Interested parties can be automatically notified if changes have been made to a requirement.



There are several requirements management tools on the market and we have used many of them. Our experience indicates that Borland's CaliberRM is the cream of the crop and we recommend it.

## TEST CASE DEVELOPMENT

It is important to note that so far no code has been developed. In our model, we can now move directly to creating the test cases.

Test cases are defined as "what do you want to test". They are the tests that need to be executed to test the requirements. Test cases can be created once the requirements are drafted, reviewed and corrected for ambiguity, and reviewed and corrected for content.

Creating test cases is at the core of what testers do.

The main issue with test cases is coverage. What is coverage? Test coverage is what percent of the application is exercised when executing the test cases. Research has shown that most testing departments can only produce about 30% test coverage. How do people know? Well, they execute their test cases while opening a code coverage tool which then gives the statistics for coverage.

Test departments produce a low level of test coverage because it is believed that greatly increasing test coverage would involve greatly increased use of resources and time. Testing is, after all, about mitigating risk while employing an affordable number of resources.

Our team was looking for a way to increase test coverage while at the same time not increasing the needed resources, or even decreasing testing time and resources. To find the solution we again turned to Richard Bender.

Cause-Effect Graphing is a test case design technique and a tool that allows the tester to model the requirements. The BenderRBT Tool then produces the minimum number of test cases which will give 100% test coverage. It is based on the same algorithm that is used to test micro-processor chips. Micro-processor chips, even though they have a high degree of complexity, work virtually without error. The few chips which didn't work became instant celebrities. The reason for this high degree of success is that the manufacturers do complete testing.

The time it takes to model and use BenderRBT is about the same as the time it takes to develop test cases manually. The difference is that Cause-Effect Graphing produces a higher percentage of test coverage with usually fewer tests cases compared to creating the test cases manually, thereby saving time and money.

The other benefit of using Cause-Effect graphing is that it points out missing paths. For example, the requirement document may say if A and B then X; if A and Not B then Y, if B and Not A then Z, but it forgot if Not A and Not B then what? Cause-Effect Graphing can pinpoint these missing scenarios and allow the requirement writers to correct the specs.

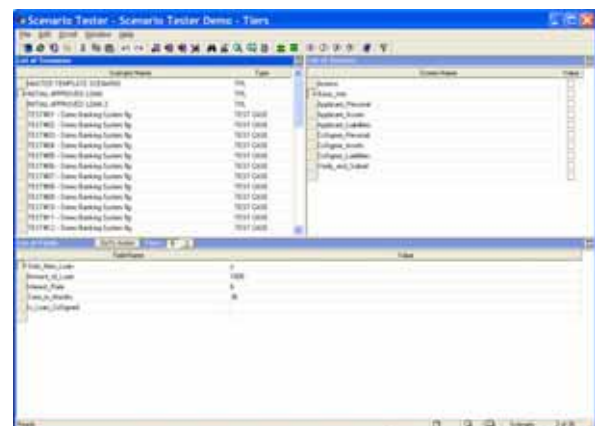
## TEST DESIGNS

If Test Cases are "what to test", then Test Designs are "how to test them". Test Designs include the inputs, navigation and expected results. It is easy to say "test that an annuity is compounded at 2% per annum", but one still needs to plan out where to go in the application, what specific inputs need to be entered, and finally what are the expected results. Until now, this has been highly labour intensive work.

A common assumption appears to be that the testers will simply "know" how to execute these tests on the fly. If they don't know, then the testers have to spend a considerable amount of time mapping out and documenting the Test Designs. The more complicated the application, the more complicated and time consuming are the Test Designs.

This is a problem that our team set out to solve. How do you create Test Designs quickly and efficiently?

We originally ran into this problem during the days of Y2K testing. All the information that we needed, the list of tests that needed to be run, with the navigation and actual inputs and expected results, were in the heads of subject matter experts. We needed to extract this information, but we only had access to the experts for a very short time.



Additionally, we wanted to use this information as the data for test automation. State-of-the-art test automation, at that time, used either CSV (Comma Separated) files or spreadsheets to store the test data. However, in this example, there were over 200 screens involved and the each

screen had about 40 fields. That would mean about 8000 fields for every scenario. That volume made both CSV files and spreadsheet impossibly cumbersome and complicated.

There were no tools in existence that which truly solved this problem, so our team developed one. We called it Scenario Tester and the purpose was to have both a test design tool and a front end for test automation.

Scenario Tester gives the tester a view of the application under test as all the fields of the application are duplicated. The test data is stored in a database and is presented in a clear manner.

When our team implemented the tool, we found that the design worked splendidly. Testers with next to no training could enter test scenarios in a fraction of the time it would take to write them down.

This had two effects:

1. They tester actually created the test scenarios rather than avoiding them.
2. The number of scenarios dramatically increased to give testers increased test coverage.

It became cost effective to create a complete set of scenarios to fully test the software.

Subsequent projects have shown the same results.

Another consequence of this design is that if the GUI is known, then Scenario Tester can be set up very early in the development life cycle. The tests can be fully designed before even one line of code is written.

One of the more recent advancements is that we can now take the test cases and outputs created by BenderRBT and run an automated routine against them, which creates the test designs with full data in Scenario Tester. The most time consuming activity is now reduced to simple automated input.

The advantage of using Bender RBT to create the actual test cases is that the number of scenarios needed to complete full testing is dramatically reduced.

## TEST AUTOMATION

One of the most important aspects of testing is the need to run the same tests over and over. Doing this manually has turned out not to be a reality for the testing world. Testers do not have the time or the motivation to re-test everything thoroughly every time there is new build. This is why over

50% of bugs in production software are caused by fixing other bugs. People do not have time to do a full re-test.

The answer that is always given to this problem is test automation. If the tests are automated, then a full set of tests can be run for every new build, right? Well, no. Here too, the reality has proven to be harsh. Studies have shown that over 75% of test automation efforts have become shelfware within a year. They failed. Those test automation efforts which are still on-going almost never offer a complete set of tests. They are more like glorified smoke tests, meaning that they may go into most screens and do a few things, but they do not fully exercise the application.

The reasons for this abysmal failure are easy to see: no matter how much the test automation companies want to tell us otherwise, test automation involves writing and maintaining code. Testers are not developers and do not want to spend time creating and controlling code. But they are the ones who are supposed to write it all.

We have seen time and again, where some poor Business Analyst or tester is handed an automated test tool and maybe is sent to a week-long training class, then is told "now use this to test". The tester is now expected to become an excellent programmer overnight and write a clear, maintainable and modular script that is sufficiently rugged to withstand changes in the application... It just doesn't happen.

The test data (the actual values typed into the application or the output from the application) should not be hard-coded into the scripts, so it has to be stored externally. The maintenance of large quantities of data in the usual way (CSV files or spreadsheets) becomes, as mentioned above, nearly impossible.

To fix this problem, our team used the same pattern that exists for cars or televisions. Nearly anybody can drive a car, nearly everyone can watch TV. One does not have to know how to build one in order to use one. In most cases, people do not know or care what the background technology is that makes these products work, as long as they do work and the human interface allows the users to perform the tasks that they need to perform.

Realizing this, our team set out to do the same thing with Scenario Tester: provide the testers with a highly effective and useable interface, while at the same time hiding the engineering from them. The engineering (the test automation code) is created by skilled professionals.

This approach has worked very well; projects that use this approach have lasted. Some projects have now been going for over three years. The depth of the testing has also

increased. On some projects, test sets that provide 100% test coverage have been automated. Test automation is no longer running glorified smoke tests.

The tester presses a button and the tests run.

## DEFECT TRACKING

Good defect tracking systems are essential in well run projects. Here again good tools make all the difference.

Many projects use e-mail, word of mouth, Word or Excel to track defects. They do this because the advantages of a real defect tracking system are not understood, or because of the expense, or for other reasons.

Some of the consequences are:

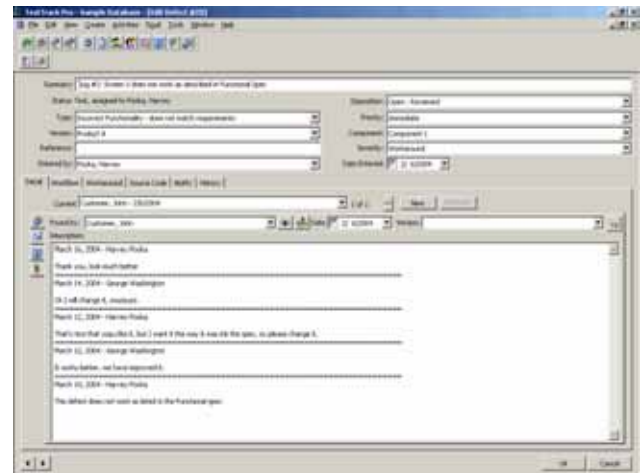
- There is no central repository for the defects
- There is no easy way to search through defects, for example for all those belonging to an individual, or for all closed defects
- There is no easy way to maintain defect statuses
- Communication breakdown, especially if emails only go to certain individuals
- Project managers are unable to track the progress of bug fixes or feature development
- Defects get “lost”, either by not getting fixed or by not getting tested
- Developers complain that they do not get proper information on how to reproduce the defect

In contrast, a good defect tracking system provides the following important features and benefits:

- A simple and friendly user interface.
- All defects are tracked in one place
- It is a highly efficient method to shepherd problem resolution
- It increases communication and information sharing among all project team members
- It puts all project team members “on the same page”
- It provides an easy way for all people to track progress in the project

By actual observation, a good defect tracking system will pay for itself many times over, by reducing the cost and time of the project.

Our team has used and evaluated many different defect tracking systems. The best so far, in our opinion, is TestTrack Pro from Seapine Software and we recommend it.



A related issue is that the information needs to be given to the developers by the testers to resolve a defect. When a defect is identified, it needs to be communicated in such a way that the developers can reproduce the defect and know exactly what it is. A common complaint from developers is that there is not enough information provided, requiring either a flurry of emails or a phone call to clarify. For example, a defect that reads “the calculate button on the customer screen doesn’t work” is not a defect that can be fixed, since it works perfectly when the developer clicks it. What was done before? What were the steps leading up to the problem? What customer was showing at the time?

The ideal would be an exact step-by-step description of everything that was done leading up to the defect. But this is very tedious for the testers to type in.

Our team therefore came up with an automated solution to this problem. During the automated test script execution, a log file is created with a full description of everything that went on, screen by screen, with screen shots, every step listed, and all the necessary information. The file even contains a comparison of the Actual vs. the Expected screen.

The developer receives this file as part of the defect write-up in the defect tracking system. The question, “what did you do to cause the defect?” does not come up.

## A GLIMPSE OF THE FUTURE

This article so far has discussed how the use of advanced tools can produce better software. But introducing better tools is only a start. The car companies are way past the stage of better tools but have introduced full automations using robots to build cars from beginning to end.

Testing will also move in this direction. To describe how this could look, we give an example of an actual project we

saw, that was tested in a very traditional manual way, but could have been fully automated.

This IT project had created the requirements using Visio. As discussed above, requirement should be written in a deterministic manner. These Visio diagrams satisfied that condition – they were complete and deterministic.

It would have been easy to take those diagrams and write a program which would validate the Visio diagrams and then translate them and input them into Bender RBT.

Bender RBT would then have created a set of Test Cases which could be imported into Scenario Tester.

Scenario Tester could then have executed those tests against the application.

But we could have gone one step further. Because our focus is automation, we could have totally automated all the above processes. The end result would be that once the requirements were created, automation would take over to test those requirements against the application.

Like the robotized assembly lines of the car companies, we would have had minimal human input once the requirements were designed. Automation could take over and create and execute all the tests.

The important fundamental here is the structure, the form and the presentation of the requirements. If these are correct, then the rest can follow. And they can be done correctly – the just mentioned project did it very well, using Visio.

So the dream of “here are my requirements, now test them for me automatically” is no longer just a dream, but is highly doable.

It is not a question of whether this will happen, but when. It is the nature of progress of any industry that wants to increase quality, while at the same time decreasing costs and time to market.

## CONCLUSION

Testing has now become an integral part of all stages of the development life cycle. Finding defects early leads to cost savings and improved requirements and applications.

Testers will need to expand their expertise so they can add value at each phase of development.

The GUI design should be done up front. The GUI is tested through usability testing which allows for more useable software and the early design of test scenarios.

The requirements need to be written deterministically. These are tested through Ambiguity Reviews and Content Reviews. A good requirements management tool such as Borland’s CaliberRM dramatically raises the effectiveness of analysts, testers and developers by managing the collection of requirements more efficiently. Test cases are generated using Cause-Effect Graphing, which is supported by BenderRBT Inc.’s test case design tool. BenderRBT allows for the further testing of the requirements. Not only are issues found earlier but it is possible to obtain 100% test coverage with no more effort than the industry normal 30% coverage.

Test designs can be created and stored in Scenario Tester even before any code is developed. BenderRBT test cases can be imported into Scenario Tester to create Test Design automatically. Comprehensive tests can be developed earlier, and with less effort.

Test automation, which is robust and easy to maintain, can execute the Test Scenarios which have been created in Scenario Tester.

Eventually the whole process will be automated. So once the requirements are built they can automatically be executed against the application.

“Earlier, faster, better” becomes a reality. But the real advantage is in cost savings.

The increased use of tools and increased automation of the tasks that testers perform is the way of the future. Testers will need to adapt if they want to have an important and dominant role in the future of IT.

### *Bio:*

*Richard Borney 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 [rbornei@eol.ca](mailto:rbornei@eol.ca).*

## Michigan’s 5th Annual Conference on Quality

October 7, 2005

Focus Hope Conference Center

Detroit, MI

Contact: Nancy Poma

[nmpom@comcast.net](mailto:nmpom@comcast.net)

## Conference Watch

<b>Michigan's 5th Annual Conference on Quality</b> Detroit, Michigan, <i>October 7, 2005</i>	<b>Page 11</b>
<b>The International Quality Conference</b> Toronto, Ontario, <i>Oct. 5-7, 2005</i>	<b>Page 13</b>
<b>Project World and Business Analyst World Conference</b> Toronto, Ontario, <i>May 8-12, 2006</i>	<b>Page 17</b>

### Quality Assurance Institute

#### Certified Software Quality Analyst (CSQA)

Acquiring the designation of Certified Software Quality Analyst (CSQA) indicates a professional level of competence in the principles and practices of quality assurance in the IT profession. CSQAs become members of a recognized professional group and receive recognition of their competence by business and professional associates, potentially more rapid career advancement, and greater acceptance in the role as advisor to management.

#### Certified Software Tester (CSTE)

The Certified Software Tester (CSTE) certification is intended to establish standards for initial qualification and provide direction for the testing function through an aggressive educational program. Acquiring the designation of Certified Software Tester (CSTE) indicates a professional level of competence in the principles and practices of quality control in the IT profession. CSTEs become members of a recognized professional group and receive recognition of their competence by business and professional associates, potentially more rapid career advancement, and greater acceptance in the role as advisor to management.

For more information visit web site: [QAIUSA.com](http://QAIUSA.com)

## 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.

### CSQA and CSTE Exams in Toronto

Sept 17, 2005 and December 3, 2005

For details, visit web site: [QAIUSA.com](http://QAIUSA.com)



## 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)



## 2005 International Quality Conference Champions of Quality

October 3-7, 2005  
Toronto, Canada

Join other Champions of Quality in the pursuit of truth,  
process and the quality way in Toronto, October 3-7, 2005  
at the 2005 International Quality Conference.

More information is available at:

<http://conference.dkl.com>



# TASSQ<sub>U</sub>ARTERLY

*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.*

Software is often considered a nebulous product, unlike something made out of concrete or steel. People recognize that putting up a building structure requires a lot of planning and calculation. But software development is closely akin to both architectural and engineering disciplines.

These fields utilize a well established methodology which begins with truly understanding the customer's desires and needs. In software development, that includes the business, functional, and performance Requirements; these are then translated into Specifications, which evolve into a Design, and finally become executable code. Then can begin the process of dynamic testing. Unit, Integration, System, & Acceptance testing must be based on these explicit definitions.

Each discrete step in this process makes an important contribution to the final product quality. Often-times, the temptation is to apply pressure to by-pass some of these necessary steps, to 'speed up' the process, and/or to 'save money' in order to deliver the final product 'on time'.

In the long run, this inevitably back-fires deferring all defect removal until the testing phase or, even worse, promoting buggy software into production. The result is often that the project manager moves on with a 'successful' resume, for delivering 'on-time and within budget'; the programmers are stuck with 'maintenance' while testers are ultimately blamed for promoting a shoddy product. This is not the most cost-effective way to develop and implement software.

Part of the problem is that the executable code is the most visible deliverable; documentation is secondary. It is not easy to quantify these without performing the proper verification and validation. These are softer deliverables in which performance is often difficult to measure.

## They Just Don't Get It: Part Two

By Robert Sparkes

Studies done by IBM (et al) have long established that actual program coding occupies only 10% of the total task; while Testing constitutes 50% of the overall project costs. This ratio has stood up, in good projects, regardless of the programming language and development methodology.

Software development is usually an expensive, time-consuming, and labour-intensive process. That investment must be amortized across the expected life-time of the product. Before the advent of the PC, most software was developed for in-house corporate use. But since the era of Personal Computing & the Internet, we have seen a veritable software explosion from productivity tools, to interactive games & entertainment. Companies which produce software that satisfies customer expectations (i.e. Quality) will be successful while the others will fail.

### *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.*

*He can be reached at [rsparkes@csolve.net](mailto:rsparkes@csolve.net).*

### **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>

## Managing Your Environments

By Joe Larizza

*Development and testing environments are key ingredients to software delivery. In many cases, the failure to successfully manage these environments will cause delays in delivery and/or poor quality.*

I like to think of it as air traffic control for an airport with one runway and where airplanes land once per day. As business picks up, multiple airplanes are requesting to land simultaneously. In our world, the landing strip is our development or test environment. The airplanes are our team members trying to complete their assignments. If the airport or IT team does not have the appropriate facilities in place, we have firefighting issues.

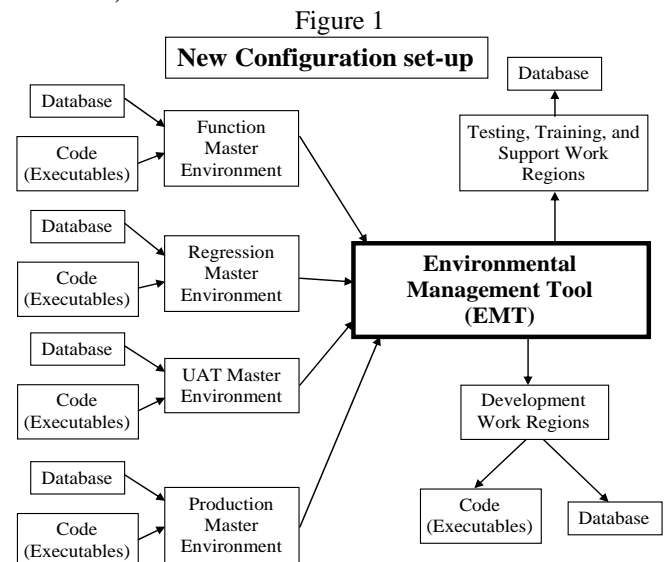
Let's take an example of an organization which has multiple environments for application "ABC" which are manually maintained. New versions of the application "ABC" are applied directly to each of the approximately twenty development and testing environments. We observe that the deployment process is time consuming and inconsistent leading to unstable environments – as in the airport example above. In addition, minimal security and controls are in place, leaving test data open to tampering or uncontrolled changes. Subsequently, this can lead to misleading test results, in both defects and individuals (business, testing and programming analysts) with unique tasks/goals interfering with each other's data.

Figure 1 describes the concept of Environment Masters and an Interface Utility that allows individuals to create their own work areas. Code releases and database management activities such as upgrades, product definitions and test data will be completed in the master environments. Master environments should be created for each development life cycle stage: business, development, functional testing, regression testing, UAT (User Acceptance Testing) and production (data base and source code matches production version).

Individuals or individual teams across the organization requiring their own development or test environment will be able to create one by cloning from the master set. So how do we clone the master environment? One way to do this is to have a simple GUI interface (Environmental Management Tool: EMT) allowing the user to create, copy or delete their work regions automatically – command line access will no longer be required. The GUI interface will remove the need to set up environments manually and should have safety

features built in that will prevent individuals from accessing the wrong databases (work regions). In my experience, the GUI interface will need to be constructed in-house. The configurations and uniqueness of most development and test environments demand a customized solution.

Get the GUI interface right and your analysts will be able to copy a master environment into their work region to conduct investigations: coding, testing or training (depending on their task).



### Master Environments

Typically four categories of master environments are required that will feed the working regions:

#### Functional Master

The code and database will contain the latest approved build from Development, i.e. the current version of code.

- One master environment (generic) for each production installation
- Product Definition - generic

#### Regression Master

The code and database will contain the latest approved build from the QA Team, i.e. the expected version of code to be delivered within the next release to UAT.

- One master environment (generic) for each production installation
- Product Definition – generic or client-specific

**UAT Masters**

The code and database will contain the latest approved build from QA Team (regression testing). Note: UAT Master will be synchronized to Client's UAT testing regions.

- One master environment (generic) for each production installation
- Product Definition – client-specific

**Production Masters**

The code and database will contain the latest approved build from UAT Team and will be synchronized to Client's Production regions.

- One master environment (generic) for each production installation
- Product Definition – client-specific

**Work Regions and Test Environments**

Possible categories of work regions that are fed by master environments:

**Testing and Support Regions**

By selecting the appropriate master environment, the database is copied to the testing region and pointed to the matching code library that is shared with the master environments.

**Training Regions**

By selecting the appropriate master environment, the master database and code (executables) library is copied to the training regions.

**Development Regions**

By selecting the appropriate master environment, the master database and code library is copied to the developer's work regions.

Note: Source code alterations, enhancements, etc. can be completed in the programmer's work region, validated individually prior (unit tested) to code being checked into source code repository for widespread use, i.e., functional testing.

**Environment Management Tool (GUI Interface)**

The Environment Management Tool (EMT) should be simple and straightforward to use, i.e., make sure the GUI or interface is correct. The GUI conceals the complexity from the user and allows the user to literally create an environment through the click of a button.

All the command line entries to replicate, delete, restore, upgrade databases and source code will be replaced with a GUI interface screen (EMT). The EMT will allow individuals to execute scripts. An example of the latter is the creation of a testing region. The Testing analysts will be

prompted by the EMT to select the appropriate master and then the EMT will execute the necessary scripts to create and configure the work region, i.e., copy database and point to the master's code library.

In addition, release management activities will be reduced to allow daily upgrades and maintenance activities to be streamlined to facilitate additional efficiency gains. In this case, manual work is reduced from the maintenance of twenty databases to four masters.

The benefits of implementing a similar solution in your organization are as follows:

- Reduce Overhead Resources
- Release Management – apply code to masters only
- Reduce database maintenance
- Expand Release Management testing
- Improve Test Predictability
- Control work areas
- Increase predictability of testing results
- Reduce disk space – work regions share common directories
- Support and Maintenance- work regions created quickly

The next challenge is to incorporate test data into the masters and create regular maintenance schedules to ensure data integrity. A process must be put in place to ensure upgrading of the masters as one goes forward. A little food for thought. Baseline the test results in the master environment prior to a code deployment and run a baseline comparison test after code has been deployed.

The Environment Masters and an Interface Utility solution may not fit your particular business but it should help further discussions to resolve development and test bed management issues. A critical success factor for software development is secured test and development regions. The lack of control of the regions will impede progress. QA's efforts rely on predictable results which come from managing the regions effectively.

**Bio:**

*Joseph (Joe) Larizza is a Quality Manager, for The RBC Financial Group. He is a member of the Board of Directors for the Toronto Association of Systems and Software Quality and is an Advisor for the Quality Assurance Institute. He is a Certified System Quality Analyst and holds a Bachelor of Arts Degree in Economics.*

*He can be reached at [Joe.larizza@sympatico.ca](mailto:Joe.larizza@sympatico.ca)*



**Two Special Events**  
**Featuring internationally Recognized**  
**Testing Expert**  
**James Bach**

**Rapid Testing Overview – One-Day Workshop**  
**Tuesday, November 29, 2005, 9:00am – 5:00pm**  
**Location: Sheraton Centre Toronto Hotel,**  
**121 Queen St. W.**

**James Bach with Michael Bolton**

This class gives you an 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 class 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 course 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 the basics of Rapid Testing.

Fees: \$250 for TASSQ members, \$350 for non-members

**Skilled Testers and Their Enemies**

**Tuesday, November 29, 2005, 7:00pm—9:00pm**  
**Sheraton Centre Toronto Hotel,**  
**121 Queen St. W.**

**James Bach**

According to James Bach, there are various schools of thought on the right way to test software. The Factory school sees testing as better when it is turned into a routine. Two variants of the factory school are the Traditionals and the Agiles. They are at war with each other. The Traditionals favor extensive documentation and independent testers, whereas the Agiles favor extensive automation and no independent testers, as such. James thinks both sides of the Factory school are fundamentally ignorant about software testing. He thinks the reason for this is that proponents of both sides typically avoid studying software testing, and instead rely on unexamined folk beliefs that seem scientific based on a view of science at least 100 years out of date. In this talk he will argue, as he has argued since 1990, that good testing is not factory work at all: it is intellectually demanding and fundamentally un-automatable exploratory investigation. It is learnable and teachable. It is a disciplined, systematic process. Good testing makes use of tools, but it cannot be reduced to merely the things tools do.

James believes that the Factory school ignores the role of skill, but thinks that skill-based process improvement (not based on slogans, documents, or metrics) is the wave of the future, and he will try to show, in this talk, specifically what skills he's talking about, and briefly demonstrate a couple of them.

Fees: \$35 for TASSQ members, \$50 for non-members

Note: This event replaces TASSQ's monthly dinner meeting. Dinner will not be served at this event.

For information and reservations, see the TASSQ Web Site at <http://www.tassq.org>, or call (416) 444-4168

Printed by permission of QAHumour  
Sourced from QAHumour.com



© 2002 Creativity Age

### How to Submit Articles to TASSQuarterly

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

The deadline for submitting articles for next issue due to be published in October is September 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.

### 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