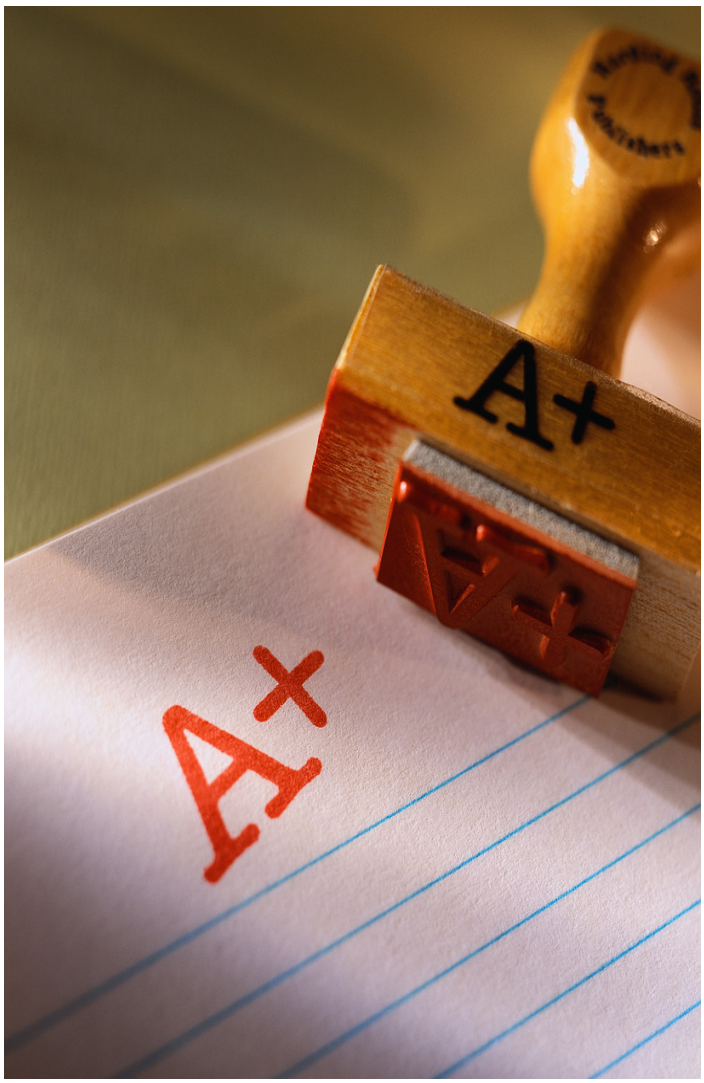
QUALITY
Software and Testing

VOLUME 4, ISSUE 1
April 2006



What is Quality Software?

We often talk about "Quality Software" but very seldom define it. What characteristics have to be in place for a piece of software/application so that the application could be considered to be "Quality Software"?

Defined by

James Bach
Cem Kaner
Jerry Wienberg
Mike Pregmon
Rex Black
Magdy Hanna
Joe Larizza
Michael Bolton
Richard Bornet

QUALITY Software and Testing

F E A T U R E S

How to Submit Articles to Quality Software and Testing

Quality Software and Testing provides a platform for members and non-members to share thoughts and ideas with each other.

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

DEPARTMENTS

TASSQ

Toronto **A**ssociation of **S**ystems and **S**oftware **Q**uality

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

Phone: (416) 444-4168
Fax: (416) 444-1274

Email: tassquarterly@tassq.org

Web Site : www.tassq.org

QUALITY Software and Testing

Editorial

PUBLISHER
Joe Larizza

EDITOR-IN-CHIEF
Richard Bornet

ASK TASSQ EDITOR
Fiona Charles

BOOK REVIEW EDITOR
Michael Bolton

ART DIRECTOR
Nuree Hwang

PRODUCTION
Jeanette Mount

Welcome to our new magazine.

For some years now, the Toronto Association of Systems and Software Quality (TASSQ) has had a great little magazine that brought its members the latest hot news and ideas in Software Quality. We've had terrific response from our readers, and for that we thank you. The one down side, if there has been one, has been the small size of our circulation – it has been for members only.

Well, it's time to grow up, to reach out, to embrace the world. With great articles from top names in the field, we have something to say and we want everyone to hear it. As of this issue, we are dramatically increasing our distribution. We're also changing the name of the magazine from *TASSQuarterly* to something that better communicates what we do: *Quality Software and Testing*.

Some of you have been reading this magazine for some years. With a greater readership, we can bring in works from prominent professionals in the field. This means more significant articles. We hope you like it.

Many of you will be reading this magazine for the first time. We welcome you! We hope that you will find the articles stimulating, thought-provoking and useful. We also want to hear from you, the potential authors. We know you're out there, and you have ideas that you want heard. Well, with our new distribution, you can be.

So what is this magazine all about? First of all, we want to stimulate debate in our field. There's nothing like a lively interchange of ideas and practices to improve how we do our work. QA can be dry stuff; a little controversy wouldn't hurt. Secondly, we want to focus on the practical side of QA. We want to provide people with *solutions* that they can *use* in their jobs. Thirdly, and closest to my heart, we want to talk about the future. We want to look at the direction our profession is taking, and present to you innovations which may enhance and reshape how we do our work. Above all, we want this magazine to be interesting and a must-read for people in our profession and in the wider IT community.

Our featured article is a look at "Quality Software." We asked several prominent members of the QA and Testing community to reflect on how *quality* software looks and feels. We got some interesting responses and we present them in the article "What is Quality Software?"

In addition Murat Guvenc writes about "Writing Testable and Code-able Requirements".

Our regular features include our humour section with our regular cartoon. This month we are adding a new section, based on the thousands of office walls and bulletin boards out there that are littered with cartoons and wise humorous statements. We thought we would add some insightful thoughts for you to paste on your office wall. We call these Life and IT Truisms or LITruisms for short. They are sprinkeled around the magazine. If you have any to share with us please send them along.

We hope you enjoy the magazine. Please feel free to drop us a line, we would love to hear from you. And if you have an article where you share your ideas, send it in.

Richard Bornet
Editor-in-Chief

What is QUALITY Software?

We often talk about what is "Quality Software" but very seldom define it. What characteristics have to be in place for a piece of software, so that application could be considered to be "Quality Software"?

We all talk about quality. We have Testing Departments to ensure software quality. We have Quality Assurance Departments to ensure product quality.

But what is a quality product? How does it look and feel? We do all this work, what should we end up with?

I have been amazed at how little discussion there is about this topic in IT departments. I have actually said to a tester about a piece of software she was testing, "This is one awful piece of software." She replied, "I know, but it works as required."

So much time and effort, and too often, what we get is mediocrity. So in this issue we thought we would look at what people think quality software really is. We asked various notable people in our profession and received various responses. We thank James Bach, Jerry Wienberg, Mike Pregmon, Cem Kaner, Rex Black and Magdy Hanna for their contributions.

We then supplemented their thoughts with our own. Joe Larizza, Michael Bolton and myself entered into the debate by adding their thoughts. I added my thoughts in an article called "Magical Software".

The question we sent out to everyone was fairly vague. We did this on purpose so as not to direct anyone in a particular direction, but to get people talking about the topic.

Here is the text that we sent out:

This month we thought we would pose an interesting question and obtain the thoughts of people of distinction in our community.

The question we want to pose is:

We often talk about what "Quality Software" but very seldom define it. What characteristics have to be in place for a piece of software/application so that the application could be considered to be "Quality Software"?

This is a fairly open ended question, but we did this on purpose. Whenever we have asked the question it has generated much debate and thought. We hope that the magazine will also engender much thought and debate when people read yours and other's responses.

We are not trying to define software quality, but are trying to collect ideas about it, and we believe that yours would be valuable.

Below are the responses we got. Again, we are very grateful to the people who took the time and effort to write to us.

Richard Bornet

From James Bach

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.

Oh, this is an easy one: The characteristics that must be in place are the characteristics that the person or persons who matter have decided must be in place. In other words, there are no universal characteristics that comprise quality.

In fact, there can be no universal characteristics, because to do that is to run afoul of the naturalistic fallacy-- we cannot derive an "ought" from an "is" or say that any "is" must be an "ought".

Quality is not a substance, it is a relationship: quality is value to some person. This is Jerry Weinberg's famous analysis, and from this principle follows this advice: the first thing to do when analyzing quality is to decide whose opinions matter.

What people value may change over time, and that changes the quality of the products they use. It doesn't merely change their perception of quality, mind you, it changes quality itself (because quality is a relationship).

Furthermore, even if what I value doesn't change, the manifestation of it might, because the product itself is a set of relations. So a high quality product might become a low quality product not because it changed within itself, but because the world around it changed in some way that "broke" the product for me.

From Magdy Hanna

Magdy Hanna is a recognized educator, speaker and consultant in several related areas of software engineering. Dr. Hanna brings over twenty years of experience with building and maintaining software systems. Dr. Hanna is the Conference Chair for the International Conference on Practical Software Quality Techniques . As a consultant, he helped many organizations define and improve their software processes using disciplined software engineering approaches. As an associate professor at the University of St. Thomas, he teaches graduate courses on several software engineering topics with emphasis on practical software quality techniques. His distinguished seminars on various topics have been highly rated by software professionals. He developed new approaches and methods in software development including the Software Quality Engineering Methodology (SQEngineer), the Unified Data Model (UDM), and the Data-Driven Object Model (DOM). Dr. Hanna holds a Ph.D. and a Masters degree in Computer and Information Sciences form the University of Minnesota.

"Quality Software"

Position 1:
Quality software means different things to different people and that is no surprise. After all, quality in general is a very subjective thing.

I always find it interesting in my classes to tell students what I call the story of "*My Wife and the Ground Beef.*" It is a real story that probably happens with every one of us every day when we try to deliver the best quality to our customers. Here's what happened:

My wife asked me to stop by the grocery store and pick up a package of ground beef on my way home from teaching one of my classes on quality management. This is a class where I spend a significant amount of time speaking to students about listening to the voice of the customer when deciding on what quality characteristics our software must exhibit.

Well, scanning through the different labels on packages in the meat bin, I found my self naturally, as a high quality seeker, moving toward the leanest grade of ground beef until I finally spotted a label that read "Less than 9 % fat." I was excited that I would be bringing home something that my wife would like, what a great thing! Being hungry as well as eager to make my wife happy, I rushed to get back home to deliver my highest quality product to my customer.

Imagine my surprise when I found my wife disappointed in my choice of beef. She did not care at all for what I thought was the highest quality package of ground beef. Did she want more fat in the meat? Guess what? Yes, she did. If I had not known what a great cook my wife is, if I had not enjoyed her cooking for 25 years, I would have doubted her taste.

I decided to be a good provider and listen to my customer one more time. So, I asked, what is wrong? Why don't you like my ground beef? This is the leanest ground beef you can get. She answered in a firm tone that really indicated that she really knew what she wanted,„"I do not want the leanest ground beef." You can only imagine the look of frustration that appeared on my face. This was like my customer is telling me I do not care for the good quality you are giving me. Not knowing much about cooking, I decided to ask, why? The explanation from my wife came as a strong indication that I did not really practice what I preached to my students just about an hour before.

My customer is now telling me her requirements for the first time. Knowing exactly what she wanted, my wife said I am making a dish that takes 45 minutes to cook in the oven and if the meat does not have enough fat it gets too dry.

Where is the problem? Do you think my wife should have told me how lean she wanted the meat to be? Was I supposed to ask her how much fat she wanted in the meat? The point here is that we as software professionals very often think we know what high quality software is, so we just go ahead and deliver software that meets our own definition of quality. This incident with my wife caused me to believe that like beauty, quality is in the eye of the beholder.

So, where does this leave us? It leaves us with the responsibility of exploring with our customers what they

really mean by quality. This means that we have to define quality as our customer really sees it. If the customer says he wants the system to be easy to use, then we need to understand what ease of use really means to our customer. We would create a list of characteristics that our customer would be looking for in the product that make the product easy to use. The same applies to all other quality requirements such as reliability, security, interoperability, survivability, configurability, safety, etc. In my classes, I provide my students with 16 types of quality requirements and ask them to take the time to develop a list of characteristics that characterize each one of those quality requirements.

Position 2: This is the one that most likely will create some controversy.
Whenever I ask the question "what does quality software really mean?" to software professionals in any of my seminars, the most common answers that almost every one agrees upon are quality software is software that meets requirements, does what it is supposed to do, or meets the business needs. Phil Crosby also defined quality as *conformance to requirement*. In my opinion, that view of quality is flawed for at least two reasons. The first reason is that we all know that requirements are never complete, precise, or detailed enough. It has been my experience that customers do not know what they want until they don't get it. In addition, requirement documents mostly address functionality of the software; things we expect the system to do and things we expect the system NOT to do. Requirement documents rarely address quality requirements such as security, reliability, performance, survivability, interoperability, etc. So, what are requirements conforming to?

The second reason became obvious to me when I repeatedly asked test and quality professionals in my seminars about what they do to assure the quality of the application. Again, the most common answer I get is we make sure all requirements have been tested or *covered.* In my opinion, focusing on requirement-based testing to assure the quality of the application is a losing battle. Test professionals often do not believe this. Requirement-based testing might be sufficient only of you can guarantee that the requirement document contains every thing that is in the code. This means that every function supported by the code can be traced back to the requirement document. We all know that this is not possible. There is always a gap between what has been stated in the requirement document and what developers have put in the code. You may wonder why. The reason is that the development team always starts with somewhat high level, vague, incomplete and most often, ambiguous requirements. One of two scenarios often happens depending on what type of developer's personality we have. Good developers will investigate further, ask questions and try to obtain more details and resolve ambiguities. Other developers will make their own interpretation of requirements and fill in missing details based on their own experience and opinion. In both cases, the result is the same: the design and the code will have many different things that were not reflected in the requirement document. Even when projects utilize good requirement change control practices, these things always escape the process, simply because no one perceives them as changes to the requirements, only clarifications. As a result, when we use code coverage tools to determine how effective our requirement-based testing is, we always get an unpleasant surprise. My own experience suggests that requirement-based testing barely covers 50% of what is in the code. Now, the definition of what quality software is really comes down to how much of the code have we tested? How big is the gap between the code and the requirements? How much collaboration was there between the test team and the development team? How aware are testers with every aspect of the code? And finally, how much testing have developers done based on their code? Remember, every path in the code that has not been tested presents potential unpredictability in the behavior of the system and potentially undesirable behavior. It all makes sense to use code coverage, as opposed to requirement coverage, as a measure of quality, because when the release goes out, what gets installed and executed is code, not the requirement document. What a new challenge for software test professionals.

From Cem Kaner

Cem Kaner, J.D., Ph.D., is Professor of Software Engineering at the Florida Institute of Technology. His primary test-related interests are in developing curricular materials for software testing, integrating good software testing practices with agile development, and forming a professional society for software testing. Before joining Florida Tech, Dr. Kaner worked in Silicon Valley for 17 years, doing and managing programming, user interface design, testing, and user documentation. He is the senior author of Lessons Learned In Software Testing with James Bach and Bret Pettichord, Testing Computer Software, 2nd Edition with Jack Falk and Hung Quoc Nguyen, and "Bad Software: What To Do When Software Fails" with David Pels.

Dr. Kaner is also an attorney whose practice is focused on the law of software quality. Dr. Kaner holds a B.A. in Arts & Sciences (Math, Philosophy), a Ph.D. in Experimental Psychology (Human Perception & Performance: Psychophysics), and a J.D. (law degree).

Visit Dr. Kaner's web sites: www.kaner.com and www.badsoftware.com.

I like Jerry Weinberg's definition of quality, "Quality is value to some person." This emphasizes the subjective nature of quality--it's different for different people--and the practical implication of quality--software that is better (for you) has more value (to you).

Years ago, I worked at Electronic Arts, making software for the Amiga. I've frequently heard expected time to failure on the early versions of the Amigo OS estimated at 12 minutes. EA had a bunch of techniques for working around the weaknesses in the Amiga operating system--despite those, we estimated a mean time between operating-system-caused failures at 4 hours. Most classical descriptions of quality would include high reliability as a fundamental characteristic. But despite the unreliability of the Amiga OS, the platform developed a large and loyal market. It provided other values, such as unparalleled (in its time) multimedia support, that these individuals found more valuable.

Let me distinguish "value" (quality) from "conformance to written requirements." When they exist at all, the piece(s) of paper that we label "requirements" map to what were some of the actual needs and preferences of some of the stakeholders associated with the product at some time in the past. Conforming to these may or may not provide much value to any particular person at any particular time.

So, what are the characteristics of "quality software?" We might create a list of characteristics to consider on a product-by-product basis, but not an absolute list of quality characteristics. The world just isn't that simple and we do ourselves and our profession(s) a disservice when we pretend that it is.

From Mike Pregman

Dr. Mike Pregmon is the Executive Vice President of the Quality Assurance Institute. www.qaiworldwide.org. He is also an IT quality industry contributor for the television program World Business Review hosted by General Alexander Haig on CNBC, Bravo and Asia television. www.worldbusinessreview.com.

This is a very interesting question because "quality" software may have a different connotation depending on where or by whom in the development lifecycle quality is being assessed. However, the most important and overriding issue tends to be software "use" satisfaction. Specifically, did the software perform in the manner expected?

Research conducted at the Quality Assurance Institute reflects that the top three software quality issues from a user's standpoint are:

1) Reliability – Is the software performing correctly from an operational standpoint when needed?
2) Response to Problems – Is the provider of the software quickly responding and willing to assist in rectifying any issues encountered by the user?
3) Ease of Use – This is the lack of customer difficulty in learning and using the system or application or is often described as "user friendliness."

Ironically, these three issues have continued to surface as the top concerns repeatedly over the years in studies in customer satisfaction of delivered software.

These three challenges indeed are measurable from both a producer's as well as a user's standpoint. Further, item #2, which should be the easiest of these to satisfy by providers, is often the most concerning for users. When producers satisfy this issue, a huge step in customer satisfaction is realized.

Nevertheless, if the software gets high marks with these three challenges, chances are it will be considered "QUALITY" software in the marketplace.

From Jerry Weinberg

Jerry Weinberg for more than 45 years has worked on transforming software organizations, particularly emphasizing the interaction of technical and human issues. After spending between 1956 and 1969 as software developer, researcher, teacher, and designer of software curricula at IBM, he and his anthropologist wife, Dani Weinberg (see her bio for more about Dani), formed the consulting firm of Weinberg & Weinberg to help software engineering organizations manage the change process in a more fully human way. www.geraldmweinberg.com/index.html

I'll stick with the definition I gave in my Software Quality Management Series: "Quality is value to some person or persons."

What's the fuss? This definition is pragmatic and has held up well for 50 years. Why change? Or do you still think quality is some objective measurement, something in the software rather than in its relationship to people who use it, pay for it, or are victimized by it?

From Rex Black

Rex Black is the President of RBCS (www.rexblackconsulting.com), a leader in the area of testing and quality assurance. RBCS has over a hundred clients in about twenty countries around the world, offering them services like training, assessment, consulting, staff augmentation, insourcing, off-site and off-shore outsourcing, test automation, and quality assurance. Rex's best-seller, Managing the Testing Process, has reached over 22,000 readers on six continents (but the penguins in Antarctica won't buy it).

I suspect that, in the near future, many types of software will become commoditized, just as many types of computer hardware have. The open-source phenomenon is leading the way, with Linux and Apache ascendant on the Internet. Regardless of the motives of the partisans of open-source software, the motives of the important business users of these open-source applications are clear: They want cheap software with the same quality levels as the commercial alternatives.

Basic economics tells us, for commodities, prices and profit margins are low, features are standardized, and quality is an absolute must for participation in the market. Failure to deliver consistent quality damages a business' ability to compete in a commoditized market. To deliver quality software, we need to start with a working definition of what quality software is.

Among other services, my consulting company offers training courses for software and systems professionals. Most of those courses focus on testing. We have taught thousands of attendees in dozens of countries around the world. Towards the beginning of these courses, we often ask people, "For the systems you build, what comes to mind when you think about the word, 'quality'?"

We usually separate the responses into two main groups: *outcomes* and *characteristics*. By outcomes, I mean what would the result be, after the software was delivered. By characteristics, I mean what would be true about the software that was delivered. Let's look at each group.
In the outcomes group, responses boil down to one of two definitions:

- The software conforms to its specification.
- The software fits its various uses and purposes.

The first definition closely follows Phil Crosby's definition of quality, while the second closely follows J.M. Juran's definition.[1]

The second definition is my favorite. Fully articulated, it means the software has those attributes, characteristics, and behaviors that satisfy the customers, users, and other stakeholders, and has few if any of those attributes, characteristics, and behaviors that dissatisfy them.

The first definition sounds good initially, but turns out to be a will-o'-the-wisp when applied to software. According to Capers Jones' studies, almost half of all defects are introduced during requirements and design specification. Testing the quality of software against the specification only is like measuring with a flawed yardstick.[2]

However, how do we measure against the "fit for use and purpose" definition, either? This is where the "characteristics" part of the discussion comes in.

Depending on the software or system in question, some course attendees list characteristics like reliability and performance. Some list usability and scalability. Some list data integrity. Interestingly, many fail to mention functionality; i.e., the ability to fulfill correctly the stakeholders' business needs for the software. When we mention that to attendees, the response is usually a type of "no duh!" reaction. It seems some people think that some quality characteristics—and, of course, the need to test them—are simply obvious.

Unfortunately, what's obvious to some people is not obvious to all, and what perhaps should be obvious to project participants is sometimes forgotten entirely. So, determining which quality characteristics are important, and how important they are relative to each other, is crucial to the proper focus of the testing effort. When my associates

---

[1] For their actual definitions, see Crosby's book, *Quality is Free*, and Juran's book, *Planning for Quality*. Interestingly, it is typical for only one or two course attendees to be familiar with either Juran or Crosby, and very few have read either book, which seems to make Crosby and Juran the Andy Warhol and Lou Reed of software quality.

[2] Maybe you are thinking, "You are missing Gerald Weinberg's definition, 'value to some person.'" It's missing for two reasons, the first being almost no one ever mentions it. The second reason is that we don't mention it in our courses, either, because it's flawed. Weinberg is correct to include the "some person" aspect of the definition, because that emphasizes the need for users, customers, and other stakeholders to make the final determination of quality. However, people value a number of product characteristics, such as those related to feature, price, and schedule, that are not quality characteristics. Weinberg's definition therefore mixes up considerations that should be kept separate to encourage clear thinking about project trade-offs.

and I manage testing projects, we typical do risk-based testing. In my approach to risk-based testing, we start by analyzing, for each possible quality characteristic, the various risks to the quality of the system. For each of these quality risks, we then determine what the level of risk is. This allows us to focus our test effort, and prioritize our tests, based on the risk posed to the system.[3]

Of course, determining which quality characteristics are important, and how important they are, is not only crucial to testing, but also to the rest of the project team. Quality cannot be tested into software at the end of the project. Simply grinding out as many bugs as possible, in addition to being inefficient, will not result in software that yields the delightful quality that we experience with the most well-designed products that we use.

So, where can you find a generic list of quality characteristics? Some companies use the ISO 9126 standard. This standard specifies six main quality characteristics—functionality, reliability, usability, efficiency, maintainability, and portability—and, for each characteristic, two or more subcharacteristics. For example, response time (performance) and resource usage are both subcharacteristics of efficiency.

I have found that a generic checklist of about two-dozen quality risk areas has worked well, too. I use this list to structure my conversations with project stakeholders about quality, particularly during quality risk analysis. What could go wrong in each quality risk area? How likely is that particular quality risk? How much trouble would it cause? Whether you use my checklist or the ISO 9126 standard, either will provide a framework for understanding system quality and how to test it.[4]

This brings us to my final point. In about one presentation out of ten, someone will respond to the question about quality in a totally different way, giving a response that I would classify in a *knowledge* group. By knowledge, I mean how would you know whether the software had quality. A typical response in this group might be, "Software that was thoroughly tested in a way that covered all important quality risks, with few if any blocked tests, critical failures, or high-priority bugs at the conclusion of testing." These attendees understand that, while testing cannot change the quality of software, testing can offer the organization the opportunity to correct quality problems, and can build confidence where the system is observed to work properly. As a test professional who believes that testing plays an essential role in delivering quality products, I find this to be not only a good response, but a professionally gratifying one, too.

From Michael Bolton

Michael Bolton provides worldwide training and consulting in James Bach's Rapid Software Testing. He writes about testing and software quality in Better Software Magazine as a regular columnist, has been an invited participant at the Workshop on Teaching Software Testing in 2003, 2005, and 2006, and was a member of the first Exploratory Testing Research Summit in 2006. He is Program Chair for the Toronto Association of System and Software Quality, and an active member of Gerald M. Weinberg's SHAPE Forum. Michael can be reached at mb@developsense.com, or through his Web site, http://www.developsense.com

I'm honoured to have been asked for my definition of quality. Like many of us in the Context-Driven School, I use Jerry Weinberg's definition of quality, "quality is value to some person". Since I expect my colleagues to provide the same definition, how can I provide something distinctive? Maybe I can tell you about what I've learned about the definition.

One of them is that, on most projects, there are lots of "some persons". Their values and their standards all matter to some degree. When I teach Rapid Software Testing, I encourage testers to think of as many different user roles as they can-and we don't stop until we get to 30. I do better testing when I recognize that there are plenty of people in the project community, and that their values may differ.

I've also learned that, as a tester, whatever I might think about the product, ultimately the product doesn't have to satisfy me. My job is to provide information to the project's sponsors so that they can make informed decisions about the product. I have opinions about quality, but I'm not the product manager. This helps me to focus my work. When I see a bug that I think is really important, and management doesn't agree, I need to make sure that I've communicated the significance of the bug absolutely as well as I can. That often involves thinking about the larger system in which our

[3] You can read more about risk-based testing in my books, *Managing the Testing Process, 2e*, *Critical Testing Processes*, and the forthcoming *Foundations of Software Testing: ISTQB Foundation Certification* (working title). You can also read the articles "Investing in Testing: The Risks to System Quality" and "Quality Risk Analysis" posted at our Web site, www.rexblackconsulting.com.

[4] You can find this list of risk areas, Generic Quality Risks List, on the Library page of our Web site, www.rexblackconsulting.com, and in my book, *Managing the Testing Process, 2e*.

product operates, and thinking about the diversity of the people that use our software or are affected by it.

Jerry's definition also encourages me to be reasonable. Once I've advocated for a bug fix on behalf of someone in the project community, I need to take a step back and realize that management may have other important priorities. Shipping the product, rather than fixing every last bug, is often a rational and pragmatic decision. That's because to project management-and to customers, for that matter-a good product now may be more immediately useful or valuable than a perfect product six months from now.

It's a privilege to be paid to learn how people and systems work.

From Joe Larizza

Joe Larizza) is a SR Quality Manager, for The RBC Dexia Investor Services. He is a President of 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

I define quality software as its ability to meet the end users' requirements. The characteristics of quality software are "*due*" to requirements having the following attributes: dependable, usable and expandable.

Dependable software provides constant results accurately and meets the individuals' time expectations. Also, future change can be added to the software without lowering the effectiveness of its purpose.

Usability can be best described as software that is intuitive, understandable and where knowledge transfer takes place. Quality software does not require an individual to read manuals. Instead, day-to-day experiences and expectations are reflected in the design of the software.

Expandable can be seen as software that has the ability to change with time. Over any period, our expectations and requirements for the software will change. Therefore, quality software has the ability to change over time. During the designing stage, individuals must consider the dynamics of change and build software where modifications or enhancements can be added without jeopardizing the software quality.

There are many factors that prevent quality software from being produced, with financial restrictions often being paramount. However, regardless of whether a Chevy or Rolls Royce is required to get you from point A to B, quality attributes result from the understanding of the market place requirements.

From Richard Bornet – 'Magical Software'

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. or by phone at 416-986-7175

I have been asked to say a few words about quality software. When does software make the grade so it can really be called "Quality"?

Most people I have asked this question of basically state that quality is in the eye of the beholder, that it is a subjective opinion. To that I say, "No way!"

Knowing quality when we see it does not seem to be a problem when we refer to something other than software, cars for instance. Most people can tell you what a "quality" car is. My five year old knows this. Every time we drive in from the highway, we pass a Lamborghini dealership and he looks out the window and says, "Wow". For all he knows, that car could drive like a Trabant, and could break down by the end of the block. But he looks at the car and he knows there is something special about it. Interestingly, my other kids and I all have the same reaction.

We also know that for $200,000, the car is not only going to make it to the end of the block, but should easily make it around the block pretty darned fast. So my five year old recognizes quality.

I have a friend who owns a BMW with a number that starts with something greater than 3. Every time he talks about his car, his eyes shine. He goes on and on and on about how amazing it is to drive that car. "Wow, what a car to drive" is not something we hear often when we refer to a Trabant unless we are being sarcastic. I think my friend would say that he bought a "quality" car, and so would the rest of us.

With cars, it is not so difficult to come up with some criteria for "quality". Here is a brief list.

1. It works and breaks down infrequently.
2. The experience of driving is wonderful.
3. It has beautiful styling.
4. It has many gadgets that you want and are easily able to use.

5. It does what you want it to do.
   (This could differentiate between types of cars: sports car, van, truck etc., depending on what your need is)

We could apply the same list to software.

1. It works and breaks down infrequently.
2. The experience of using the software is wonderful.
3. It has beautiful styling.
4. It has many features that you want and are easily able to use.
5. It does what you want it to do in a way that you want to do it.

So if it is not too hard to define "quality" when it comes to cars, why is it that when it comes to software, we have conniptions?

I believe we should go one higher and ask ourselves, what makes software extraordinary, or even magical? One of the greatest compliments I ever received in my life was to have a piece of software I had designed called "magical". So I have thought about this issue for quite some time.

So what is software that is of such high quality that it takes on a "magical" feel?

Let us use the same definitions we used for the car:

1. It must work and not break down.
There is nothing more annoying than a car that breaks down frequently. If the window does not open, or the light burns out, or the car overheats, it's not a quality car. The same goes for software. If it keeps crashing, or the menus do not work, or you get wrong error messages popping up where they do not belong, it is not a quality piece of software. But just because it works is not enough to give the software a "magical" quality.

2. The experience of driving is wonderful.
We all take test drives in cars to experience the feeling of driving. I remember I took one car for a spin on a highway, and before I knew it, I was significantly speeding, but I had no awareness that I was doing it. The car ran so smoothly that it just accelerated without giving any sensation that I was cranking it up. People who speak about "wonderful" cars often talk about being "at one with the car". The car becomes a superior extension of themselves. Have you ever heard anyone saying that about software? Software is supposed to be the great empowerer, but how often is it designed to be that?

The question could be, how fast can I get done what I need to do? And how natural is it for me to do this? You can develop software which feels like an extension of a person. People can become "at one with the software" but no one ever talks about this, or designs software to feel like a natural part of a person. If you can pull off this oneness, you do get the "wow" feeling.

3. It has beautiful styling.
I have a new version of Outlook. What is noticeable is that the designers added some styling to the display. Now it tells me "Date: Today" and "Date: Yesterday" using some pretty but conservative colours. This may not sound like much, but Microsoft did make some effort to change the display to make it easier to read and look more appealing. And it does look very nice. But honestly, how many pieces of software can you say that about?

4. Extra gadgets
Most "quality" cars have loads of extra features: seats which warm up, seat position memory, TV, GPS, being able to change the radio station right on your steering wheel, and my personal favourite - being able to monitor how far the car is behind you when you reverse park. Features that we now take for granted like cruise control, variable windshield wiper speeds and air conditioning were once upon a time luxury items.

The same with software. You can build features into software which make the experience of using the software much more efficient and pleasant. We are not talking about short-cut keys but real short-cuts. Wizards which feel natural, or software which truly adapts to how you work and makes the work easier.

5. It does what you want it to do in a way that you want to do it.
All the above can be true, but if it is not solving your real problem and making you more efficient, wiser, more knowledgeable, a better salesman, accountant or whatever you are doing, then what good is it? On the other hand, if it does make you a lot better at what you are doing, then that is where you begin to get that "wow" feeling. You think, "This is an extraordinary and wonderful piece of software."

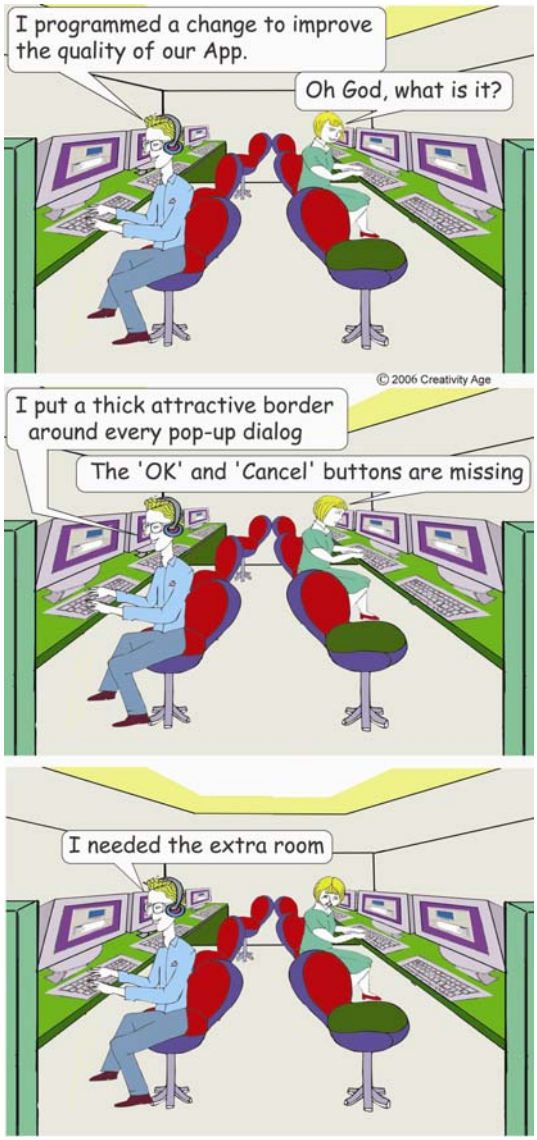
In all my years as a designer and tester, I have never heard even one person in the corporate world besides myself talking this way. I am sure that game creators talk like this every day. I am sure they say things like, "We are going to knock the socks off people" or, "People will be transcended when they play this game."

But when we talk about more mundane applications like large database management systems, the topic does not even come up. Don't we want our corporate users to have their socks knocked off by how much more efficient they are, or by how much better they are at what they do?

I guess not. That is why we shoot so low. We often don't even reach the level of quality as defined above, let alone magic.

So how do we achieve magical software? This is the subject of another article. For now, suffice to say that to get there, we have to know what would qualify as magical for the particular piece of software we are developing. Would it be being able to do one's job 10 times faster? Cut training by 95%? Being able to perform tasks that before one could only dream about?

Whatever the dream, if we don't even discuss it, then it cannot be achieved.



Next Issue – Call for Articles

The main topic of the next magazine will be based on words from James Bach who constantly reminds us that testers have to think and be clever. So we would be interested from anyone who has come up or heard of a bright solution to a testing or QA, or even IT problem. We will include these in an article which we want to call "Clever Practical Solutions to Software Testing and Quality Problems."

The emphasis here is on inventive, bright and practical.

LITruism

Creating quality software can be a costly and lengthy activity. Creating mediocre software will take much longer and cost much more.

Writing Testable and Code-able Requirements

Murat Guvenc

This article discusses the principles and practices for writing requirements specifications in a deterministic and explicit manner so that the requirements document is testable and is designed to eliminate costs in constructing correct solutions in a Requirements Based Testing environment.

Abstract

Initiating testing process earlier in the software development lifecycle is critical in the success of implementing high quality software systems in today's fast-paced, distributed development environment. This effort is doable only when the requirements are written to a level of enough detail that a sufficient set of test cases can be generated to validate the system's functionality. Requirements must be correct and should define the scope of the project precisely if the rest of the development effort is to succeed.

This paper discusses the principles and practices for writing requirements specifications in a deterministic and explicit manner so that the requirements document is testable and is designed to eliminate costly overheads in constructing correct solutions and introduces the concept of a Requirements-Based Testing approach.

The intended audiences for this document are project managers, business analysts, systems analysts, quality managers and leads, developers and anyone else involved in requirements development and management.

Introduction

Software testing is known to be expensive and time-consuming task. The reality is, most of the time and effort is spent in fixing the defects, rather than testing the system's functionality. As accepted by the majority of the practitioners, the cost of fixing a software error is lowest in the requirements phase. As the project moves into subsequent phases of software development, the cost of fixing an error rises dramatically, since there are more deliverables affected by the correction of each error, such as a design document or source code. The earlier an error is detected, the less damage it can do to the system, because there are very few deliverables to correct.

The principal cause for the time and expense is that software requirements are rarely written with testing in mind. This has two consequences. The first is that specifications which are not suitable for testing are also not suitable for software development. If the specifications don't allow you to identify unambiguous tests, then they don't allow you to build unambiguously correct solutions. The second is that testers must perform lengthy and difficult analysis of the specifications before they can start writing meaningful tests.

Requirements are the foundation upon which the entire system is built. To validate the requirements, test plans are written that contain multiple test cases; each test case is based on one system state and verifies some functionality

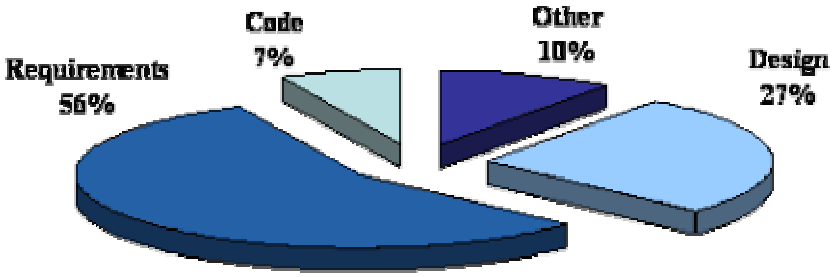


Figure 1 – Distribution of Software Defects

that are based on a related set of requirements. Requirement verification and validation is needed to assure that the functionality representing the requirements has indeed been delivered.

Writing Testable and Code-able Requirements

However, time and limited budget are always constraints upon writing and running test cases. That is why getting the requirements right, complete and concise early is important and will provide the testing group a clear idea with which to validate the system.

The most important factor in producing high quality systems is the quality of the requirements. Requirements must be written in a deterministic fashion to improve the quality of development and testing. The development of good requirements is essential because everyone on the project team works from the same set of requirements that describe what the system is expected to do. If requirements are not properly defined, the project has no foundation.

Benefits of having a good set of requirements that are written in a deterministic level of detail are, you can;

- fully document requirements so that you are able to determine exactly what the outputs will be
- resolve ambiguities, conflicts and other possible errors so that you are sure you are meeting the right requirements
- design and build test cases to validate the requirements

On the contrary, if the requirements are not written at a deterministic level of detail, you cannot;

- design tests until the design specification is complete and/or source code is implemented
- validate that the right system is being built and all the expectations are met
- manage change control on the development and release of software

Yet there are some projects that have only high-level requirements in place or none at all. The purpose is to deliver some executable components as early as possible to realize the need due to concerns like time-to-market, limited resources and/or knowledgebase. The success of the project is pretty much defined at the construction stage. Those are the projects that have not been considered to be reused/shared, often explore new technologies, require relatively a small size of team, have relatively short development cycle and require less communication and collaboration effort. For those types of projects writing testable requirements approach is not applicable.

Figure 1 illustrates the distribution of defects in projects. Over half of all project defects can be traced to the requirements process, as shown below.

The root cause of 56% of all of the software bugs identified in projects is a result of errors introduced in the requirements phase. Of the bugs rooted in requirements, roughly half of them are due to poorly written, ambiguous, unclear and incorrect requirements. The remaining half of these bugs are due to requirements that were completely omitted.

Characteristic of a Testable Requirement

Requirements should be written in a testable and deterministic manner. Deterministic means that for a given starting condition and a set of inputs, the user can determine exactly what the expected outcomes will be. Testable means that each statement in the requirements can then be used to prove or disprove whether the behavior of the software is correct.

Testable requirements are essential for the testing process, not only because test engineers must predict the expected outcome of their tests, but also the tester must verify the results of each test. These activities cannot be done with pre-specified test requirements. It must be measurable and observable. Measurable means that the test engineers can qualitatively or quantitatively verify the test results against the test requirement's expected result.

Here is the list of main characteristics of a testable requirement as described by Richard Bender who brought

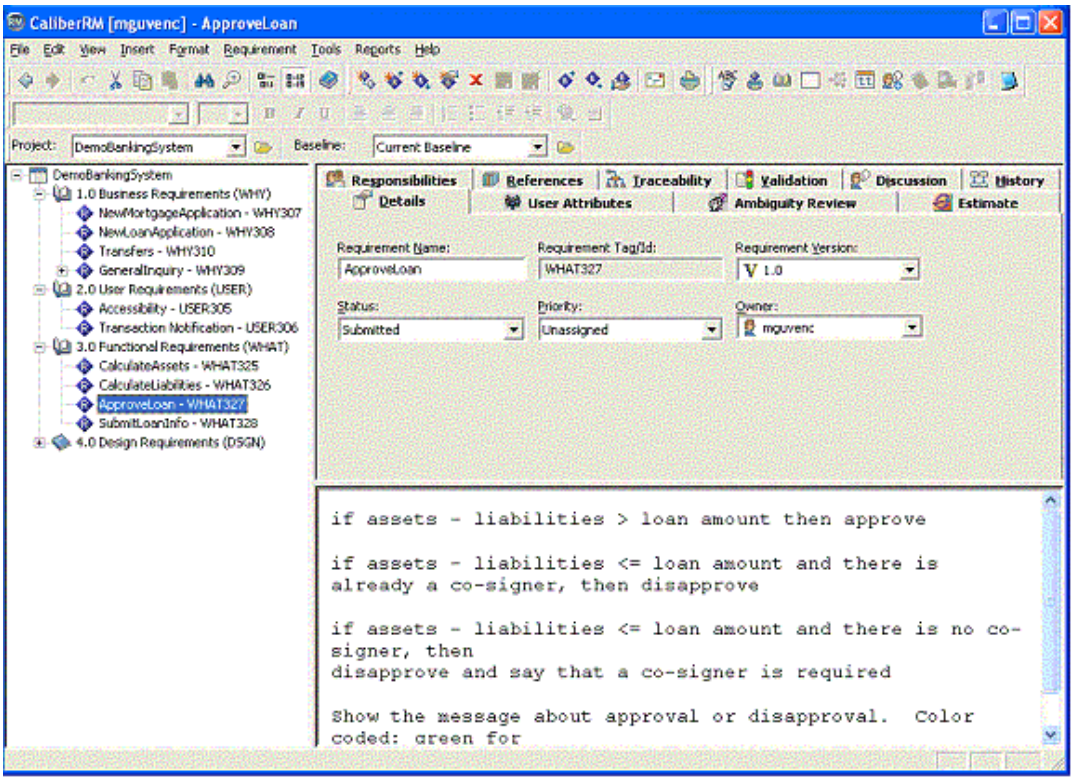


Figure 2

over thirty-five years of experience in software with a primary focus on quality assurance and testing.;

- deterministic
- unambiguous
- correct
- complete
- consistent
- explicit
- traceable

Unfortunately, in the real world many software requirements specification documents are written only after the software has been constructed. Test case results vary depending on the exposure and experience of each tester. Test coverage varies from application to application depending on the tester, and there is no way of determining test coverage at this time. The requirements documents are released, and then changes to the requirements are communicated via email, but the original requirements are not updated to reflect the email. There is risk of promoting untested code into production without providing full test coverage from a functional perspective. Code coverage analyzers are not used to monitor actual code coverage when tests are run, so there is no way to determine actual code coverage of the test case design effort at this time.

Requirements Based Testing

Requirements-based testing is the process of designing and building test cases based on the requirements of the application. Requirements-Based Testing (RBT) improves functional test coverage and reduces the risk of untested code being promoted to production. In the RBT first task is to ensure that the specifications are correct, complete, and logically consistent. Once the specifications have been clarified, the second challenge is in defining a necessary and sufficient set of tests that are needed to verify that the design and code fully meet the specifications. RBT contains two components;

- Ambiguity Review that is used to identify all potential ambiguities in requirements
- Cause-Effect-Graphing for deriving minimum number of test cases that covers 100% of the application functionality

Ambiguity Review is a technique to eliminate potential ambiguities in requirements, thus avoiding defects from the application at the earliest phase of the software development lifecycle. After the ambiguities are identified, it is the responsibility of the requirements author to correct the ambiguities, and then have the domain experts review the requirements for content. Ambiguity Review improves the quality of requirements so that the domain experts have a better quality document to work from, and help them make whatever changes are needed to the requirements content, so that requirements are not missed.

Cause-Effect Graphing is the process of transforming specifications into a graphic representation. This graphic representation depicts the functional relationships and conditions present in the requirements. The Cause-Effect Graphing technique uses a mathematically rigorous algorithm to determine the necessary and sufficient set of test cases for covering 100% of the functionality defined in the requirements. The tester no longer tries to manually determine the right set of test cases.

The list of the activities from requirement gathering to test automation is detailed below;

- Gather all requirements in a requirements management tool
- Write requirements in a correct deterministic manner
- Perform ambiguity review
- Create requirements model using cause-effect graphing technique

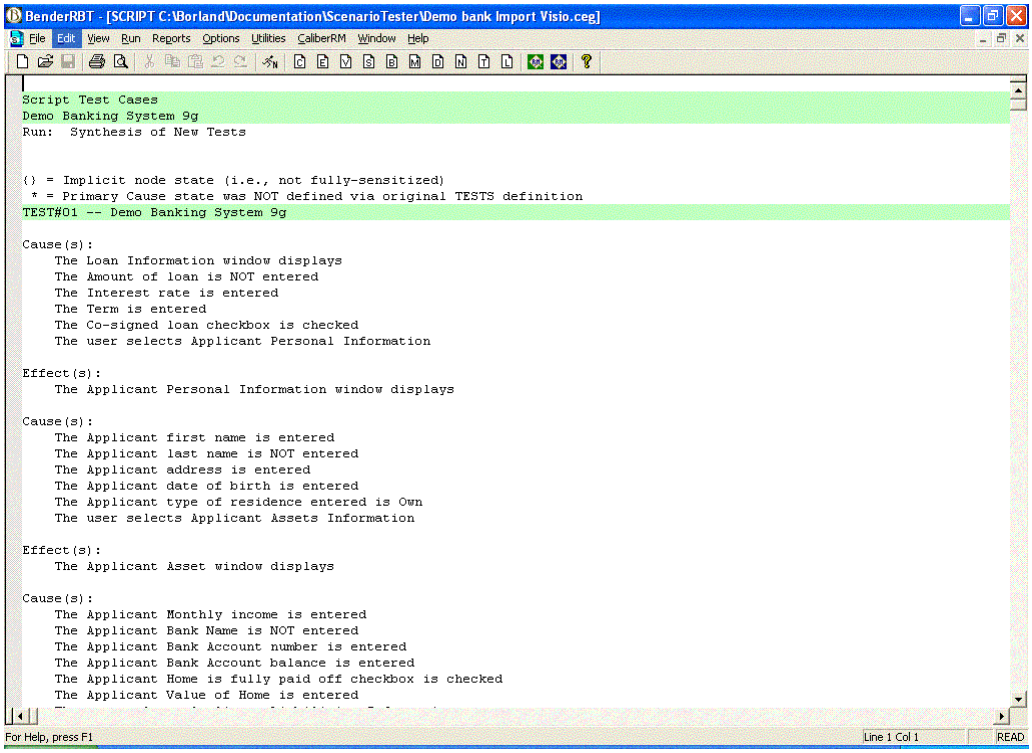


Figure 3

- Generate test cases for functional testing to provide 100% test coverage
- Integrate application code
- Generate test scripts
- Run test

In the scenario described above, CaliberRM is used to collect the requirements and store them in centralized repository, so that requirements can easily be traced from inception through deployment. Managing requirements at uniquely identifiable object allows requirements to be viewed, sorted, and reused on an individual basis, having their own level of security, change history and their own verification and validation criteria. See Figure 3.

CaliberRM provides a utility called “glossary” to perform ambiguity reviews, in which ambiguous terms, phrases, acronyms can be stored and run against project to eliminate potential ambiguities in requirements.

Bender-RBT is a requirements-based, functional test case design system that drives clarification of application requirements and designs the minimum number of test cases for maximum functional coverage. Bender-RBT uses the requirements as a basis to design test cases needed for full functional coverage. Using cause-effect graphing in Bender-

RBT, the project team can transform requirements specifications created in CaliberRM into requirements model and discover the functional relationships and conditions present in the requirements. Test cases are generated from the cause-effect graph. See Figure 4.

The next step is creating test scripts. This can be a very involved, time consuming and resource heavy task. It is possible though, as the designers of Scenario Tester have shown, to automate the generation of the automated test scripts directly from Bender-RBT output files.

This automation should involve:

- Environment Setup
- Import Test Cases (from Bender-RBT in this scenario)
- Generate Test Data
- Integrate Application Code
- Generate Test Scripts (Completely automated)
- Run Test (Completely automated)

Automation allows projects to move from requirements to actually executing the tests quickly and with a manageable complement of resources.

LITruism

We trained hard, but it seemed that every time we were beginning to form up into teams, we would be reorganized. I was to learn later in life that we tend to meet any new situation by reorganizing; and a wonderful method it can be for creating the illusion of progress while producing confusion, inefficiency and demoralization.

Petronius Arbiter (210 B.C.)

LITruism

I sometimes wonder if the manufactures of foolproof items keep a fool or two on their payroll to test things.

Alan Cohen

References

1. Chaos Chronicles, Standish Group International
2. James Martin, Information Manifesto
3. Barry W. Boehm, Software Engineering Economics
4. Dean Leffingwell, Don Widrig, Edward Yourdon, Managing Software Requirements
5. Karl E. Wiegers, Software Requirements
6. Richard Bornet, Enterprise Software Testing Systems
7. Richard Bender, Bender RBT Inc.
8. Gary Mogyorodi, Requirements-Based Testing - Ambiguity Reviews, The Journal of Software Testing Professionals

Bio:

Murat Guvenc has over ten years of professional experience in developing and implementing software systems. His experience ranges from designing business and object models to process improvements. For the past five years he has contributed within many organizations to facilitate process initiatives and best practices. He authored and delivered seminars on several topics and participated in conferences, symposiums and user groups meetings to discuss new technologies, industry guidelines and tools automation.

Events, Conferences, Education and Services

TASSQ Presents

Rex Black

Five Trends Affecting Testing

May 30th, 2006
Location: TBD

Five strong winds of change are blowing in the software and systems engineering world. As winds affect a sailboat, these winds of change will affect testing as a field, and testers as a community. Your career as a tester is at stake, and both risks and opportunities abound. In this talk, Rex Black will speak about these five trends and how they affect testing. He will offer cautions about the risks and identify the potential opportunities you face as a tester. For each trend, he will provide references to books and other resources you can use to prepare yourself to sail the ship of your testing career to the destination you desire: professional success.

Rex Black (http://www.rexblackconsulting.com) is the President of RBCS, a leader in the area of testing and quality assurance. RBCS has over a hundred clients in about twenty countries around the world, offering services like training, assessment, consulting, staff augmentation, insourcing, off-site and off-shore outsourcing, test automation, and quality assurance. Rex's best-seller, *Managing the Testing Process*, has reached over 22,000 readers on six continents (but the penguins in Antarctica won't buy it).

A Workshop with Rex Black

ISTQB Software Testing Certification

Advanced Functional Testing
Toronto, ON May 31 – June 2

This is a course for senior test engineers and test leads who want to learn powerful techniques for testing system functionality. It is especially for people who have achieved ISTQB Foundation certification and want to take the next step to ISTQB Advanced. Through a combination of lecture, discussion, annd hands-on exercises with a realistic example project, you'll learn:

- Advanced Functional Testing
- Requirement-based Tests
- Syntax Tests
- Random Tests
- Bug Attacks
- Selecting Techniques
- IEEE Testing Standards
- Test Assessment
- Review Techniques
- IEEE Standard for Software Reviews

At the end of the last day, you can take the ISTQB Advanced Functional Tester exam (pre-qualification required).

Fees and the Bring-a-Buddy discount for TASSQ Members

Advanced Functional Testing FEE US$ 2,000

Fees include tuition, course materials, certificate of completion and US$ 250 ISTQB exam fee.

Also included are breakfast, lunch and afternoon snack on each day of the course. TASSQ members are eligible for a 10% discount on the course tuition.

If you bring a friend, your friend will also receive a 10% discount.

Location
The course will be held at the Courtyard Marriott at 475 Yonge Street.

Dates
May 31 – June 2, 2006.

Time
9 AM to 5 PM

Easy Registration at:

www.rexblackconsulting.com

Phone 1 (830) 438-4830

Conference Watch

Here are some up-coming conferences:

QAI's 26th Annual Software Quality Assurance Conference
"Use It or Lose It...Exercising Your Quality Muscle"
April 24-28, 2006, Orlando, FL

http://www.qaiworldwide.org/conferences/apr_2006/index.html

Project World Conference
Business Analyst World
May 8-12, 2006, Toronto Conference

http://www.projectworldcanada.com/
Contact info@projectworldcanada.com
or 905-948-0470 ext 228
or 888-443-6786 ext 228.

Software Testing Analysis and Review (STAR East)
May 15-19, Orlando, FL

http://www.sqe.com/stareast/

XP Day Montreal
June 3, Montreal, PQ

Information soon at http://www.diasparsoftware.com

Conference of the Association for Software Testing
June 5-7, Indianapolis, IN

http://www.associationforsoftwaretesting.org/conference/

QAI Canada is now the exclusive Canadian representative of the **Quality Assurance Institute (QAI)** offering 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 10% off regular fees

2006 Public Education Schedule Q1/Q2

Date	Toronto Courses
April 3 – 5	Defining & Validating User Requirements - $1,695
April 6 – 7	Essentials of Leadership in Testing - $1,195
April 10 - 12	Essentials of Software Testing - $1,695
June 5 – 6	Essentials of User Acceptance Testing - $1,195
June 7 – 9	Boot Camp for Project Managers - $1,695
July 24-25	Essentials of Testing Web Applications - $1,195
July 26-28	Boot Camp for Business Analysts - $1,695
August 15 -18	Effective Methods of Software Testing - $2,195

For customized on site training: If your Test Team (6 or more) would like any of the above listed courses delivered at your facility, please contact aphomin@qaicanada.org

For more details and online registration: http://www.qaicanada.org/ Call Al Phomin: 1-866-899-1724 or email us.