Software Quality Live

1st Quarter 2009

SPONSORS:

CMMI Training

20% ASQ Discount

Introduction to CMMI-DEV

SEI Licensed

More Information

Advertise Here, email

swd.newsletter@yahoo.com

In This Issue:

Questions or comments?

See:	**Or Contact:**
Software Division Web Site	Bill Trest, Chair
Software Division Management	Nicole Radziwill, Chair-Elect
	Tom Ehrhorn, Secretary
	Brenda Fisk, Treasurer
	Newsletter Editor, Tom Ehrhorn

How can we improve Software Quality Live? Did the 3Q2007 issue provide helpful information? Let us know!

ARTICLES & NEWS

Message from the Chair: Changing *Good to Great* in ASQ Software Division

By W.L. 'Bill' Trest, ASQ CSQE, Chair

Welcome to the New Year

This message provides suggestions for our division members for changing from good to great. I would like to start 2009 by sharing the last few lines of a poem titled "Pretty Good" by Charles Osgood:

"... If you want to be great, Pretty Good is, in fact, pretty bad."

Software Division's strategic planning for 2009 contains several initiatives for improving value to its members. For example, our strategic plan addresses several face-to-face meetings, information sharing, and ASQ convention opportunities. Regardless of these strategic initiatives, the software division can still provide or make available to its members, much of what members deserve or may need. On the other hand, if we give members everything they need, *they will deserve even more*. Understanding this "*deserve even more*" concept is the key to establishing great, versus good, improved effectiveness and value of our software division and its members.

In their book *First, Break All The Rules*[1], Marcus Buckingham and Curt Coffman described the results of extensive interviewing done by the Gallup Organization to determine the effectiveness of workplace management. They found that if members of a workplace's organization answered the following questions affirmatively they were part of an effective organization. We can adapt these questions to the software industry, as well as the software division.

1. Do ASQ software division members know what is expected of them?
2. Do our division members have access to the necessary software industry information to do a great job?
3. Are members provided the opportunity to do a great job?
4. Do members receive appropriate recognition or praise for doing a valuable service to the software division?
5. Does someone in the software division seem to care about each member as a person?
6. Is there someone in the software division who encourages member development in the software industry?
7. Do member opinions seem to count in the software division?
8. Does the mission/ purpose of the software division make members feel membership and personal contributions are important?
9. Are other division members as committed to doing quality work?
10. Does each member have a valued friend in the software division?
11. Has someone in the software division provided information about professional progress and growth in the software industry?
12. Does the software division provide opportunities and programs, such as professional certification or training, for members to learn and grow in the software industry?

In closing, not everyone can answer all these questions affirmatively, all the time. However, moving from good to great for the software division, as well as its membership, requires continuing growth toward affirmatively answering each of these questions.

[1] Buckingham and Coffman, *First, Break All the Rules* (Simon and Schuster, May 1999)

(Bill Trest is the current Chair, ASQ Software Division. Bill is a Senior Specialist Software Quality Engineer with Lockheed Aeronautics Company, Fort Worth, Texas. He is also an ASQ Certified Software Quality Engineer (CSQE) as well as a Senior Member of ASQ, with Greater Fort Worth, Section 1416.)

Back to top

The ITEA Criteria for Software Process & Performance Improvement - Nicole Radziwill

For software professionals, particularly those of us who manage product development or development teams, it is important to track progress towards our goals and to justify the results of our efforts. We have to write effective project charters for software development just to get things moving, evaluate improvement alternatives before making an investment of time and effort in a process change, and ultimately validate the effectiveness of what we have implemented.
This past fall, I had the opportunity to serve as a preliminary round judge for the ASQ International Team Excellence Award (ITEA). My subgroup of judges met at the Bank of America training facility in Charlotte, North Carolina, where we split up into teams to evaluate almost 20 project portfolios. A handful of other events just like ours were held at the same time across the country, giving many people the opportunity to train and serve as judges. Before we evaluated the portfolios, we were all trained on how to use and understand the ITEA criteria, a 37-point system for assessing how well a project had established and managed to its own internal quality system.
The ITEA criteria can be applied to any development project or process improvement initiative in the same way that the Baldrige criteria might be applied to an organization's strategic efforts. For software, this might include improving the internal processes of a software development team, using software improvements and automation to streamline a production or service process, and improving the performance or quality of a software product. (For example, I can envision the ITEA criteria being used to evaluate the benefits of parallelizing all or part of a software system to achieve a tenfold or hundredfold performance improvement.)
You can review these criteria on the web at http://wcqi.asq.org/2008/pdf/criteria-detail.pdf yourself. There are five main categories in the ITEA criteria: project selection and purpose, the current situation (prior to improvement), solution development (and evaluation of alternatives), project implementation and results, and team management and project presentation. An important distinction is in the use of the words *Identify/Indicate*, *Describe* and *Explain* within the criteria. To *identify* or *indicate* means that you have enumerates the results of brainstorming or analysis, which can often be achieved using a simple list of bullet points. To *describe* means that you have explained what you mean by each of these points. To *explain* means that you have fully discussed not only the subject addressed by one of the 37 points, but also your rationale for whatever decisions were made.
Sustainability of the improvements that a project makes is also a major component of the ITEA criteria. Once your project is complete, how will you ensure that the benefits you provided are continued? How can you make sure that a new process you developed will actually be followed? Do you have the resources and capabilities to maintain the new state of the system and/or process?
The ITEA criteria may also serve as a useful checklist to make sure you've covered all of the bases for your software development or process improvement project. I encourage you to review the criteria and see how they can be useful to your work.

Back to top

Software Test Automation 101

Kenneth White

Test Automation Architect

Advanced Solutions International

Austin, TX

KWhite@AdvSol.com

Table of Contents

Introduction

What do the following three things have in common: Bigfoot, the Loch Ness Monster, and Automated Software Testing? Answer: Everyone has heard of them, but no one actually has any proof they exist. At least there are pictures of the first two. Nearly every company that writes software has, at one time or another, flirted with the idea of using automation to test their software. However, many of these projects end in failure because the company jumps in head first not knowing what to expect or worse, expecting the wrong thing.
Therefore, the first step in doing test automation right is setting ground rules, and that is what this paper proposes to do. The intent is to help the reader answer the question: Just what am I getting myself into? The author has done test automation over the course of his 15-year career with half-a-dozen different tools and has noticed many common themes. The reader can find these tools via a simple web engine search, so this paper will not focus on any particular tool or make tool recommendations. Rather, the intent is to focus on broader concepts that apply universally, regardless of the tool specifics.

First Things First- What Exactly is Automated Software Testing?

At the basic level, Automated Software Testing (AST) is using one program to 'drive' another. It does this either by mimicking a human user through the User Interface (UI) or by interacting directly with the source code via an Application Programming Interface (API).

Generally speaking, tests done through the API are performed by either the development team or a very technical test team. More often, when a quality assurance department is considering whether to automate testing, they are thinking of the various UI-oriented automated testing tools. For that reason, the remainder of this paper will focus on that segment of AST.

A typical UI-oriented automation tool will have two basic pieces: 1) A way to recognize and interact with the program under test, and 2) A language in which to write the automated scripts. The methods of interaction are as varied as the available tools but typically the AST will take control of the computer and move the mouse or type in input much like a user would. It finds the correct place to click or type by recognizing the object types (buttons, text boxes, etc.) and interacting with them appropriately. How well it finds the objects and how 'appropriate' the interactions are varies from tool to tool.

Regardless, behind the scenes there is a set of instructions written in the tool's language. This set of instructions is commonly referred to as a 'script,' even though 'script' is not necessarily technically accurate in some cases. Scripts can be very much like application source code or simply a list of actions in a table or some hybrid of both. And just like the UI portion, the usability and flexibility of this automation language vary with the tool.

Automation Methodologies

Like programming, there is a wide variety of Automated Software Testing methodologies. While every organization works a little different, the methodologies fall into four broad categories: record and playback, test library architecture, keyword driven testing, and data driven testing.

Record and Playback

Record and Playback Testing is very much what it sounds like. The user starts the record mode and then performs some sort of test. At the end of the test, the user clicks the stop button and saves the recording. To automatically test the application, you simply play back the recording. This is how early AST was done. However, the problems were quickly apparent. Making one minor change to the script required completely re-recording it. If the application changed even slightly, or the window moved, or any of a hundred minor variances, the script would break by either ceasing to run or running on oblivious to the fact that it wasn't actually doing anything. Modern AST tools have moved away from this model, but some still use this as a quick-start method; the tool will record the user actions and then 'write' the code doing the same thing. The code is then presented to the user for further manual editing. As a result of these deficiencies, Record and Playback can be a useful learning tool, but relying on it in the long run never ends well.

Test Library Architecture

Another methodology, which was borrowed from the programming world, is Test Library Architecture. The term quite literally refers to splitting the tests into logical modules. Modules can be divided by common functionality or common areas of the program. For example, the team may write a common 'File Open' module that can be reused any time the automated test needs to open a file. Each module does little on its own, but when assembled together, the whole becomes greater than the sum of its parts. Thus, the automation team could create a test by combining the following modules: Load Program, Login, File Open, File Close, and Close Program. Test Library Architecture is sometimes called a Test Modularity, or Componentized testing, while programmers sometimes refer to this method as writing reusable code. Most modern development, Quality Assurance (QA) or otherwise, is done this way.

Keyword Driven Testing

Keyword Driven Testing is currently an elusive target at best. However, it is so prevalent in QA lore that it deserves at least a brief look. The idea is that anyone can create an automated test using common language or easy to understand keywords. This allows non-techies to write automated test scripts thereby increasing the number of people involved in the testing and making broader use of the test automation investment. A small technical team writes the 'code' in such a way that they can simply read in and parse these keyword scripts and translate them into the automated test tool's language. During the author's own experience implementing this methodology the work load tripled as the QA team needed to create not only the actual automated scripts but the non-techies vocabulary and syntax as well as a parsing engine that could translate between the two. After a year wasted the team eventually reverted to the more conventional Test Library Architecture mentioned above. Keyword Driven Testing is sometimes referred to as Table Driven Testing or Action Word Engines.

A more successful variation on this theme involves controlling the vocabulary and formatting of the keyword scripts with a simple intermediary program. Users select keywords and appropriate actions from a list of valid possibilities and the intermediary program builds the keyword script behind the scenes. Alternatively, a spreadsheet or table listing the keywords and actions can provide the necessary structure. Either way however, the team must invest time and resources debugging and maintaining yet another application. Despite these issues, some large organizations with sufficient resources and a great many non-technical test writers have found success with Keyword Driven Testing.

Data Driven Testing

Data Driven Testing is a framework that stores data in an external file (such as a spreadsheet) instead of hard coded in the scripts.

Also, each file typically contains multiple lines of data per test script. For example the QA team would write a generic script to enter a new contact in the application under test. The test would read in the values for the new contact (name, address, phone, etc.) from a file. It would iterate through the file adding each contact. A test which adds
five contacts can quickly be expanded to fifty contacts just by adding the data to the file. The test script doesn't care; it simply keeps repeating the test as long as there is data in the file. Because the data is external to script, Data Driven Testing insulates the test script from changes in test data. One caveat to using this methodology is that test scripts need to be robust; they need to include error handling and recovery logic. Thus, as in the above example, if contact number three causes an error, the script not only needs to handle that error but also recover sufficiently to continue adding the remaining contacts. This usually results in more robust scripts all around, so it is not typically seen as a downside.

Amalgamation

In the real world, a QA team will rarely implement any methodology in isolation or stick to just one methodology. See Figure 1. The various implementations the author has completed have typically involved combining the Test Library Architecture with Data Driven Testing. This combination seems to result in the most robust and comprehensive automated test suite.

Additionally, for new teams that have never done any development or test automation in the past, a tool that allows some sort of record-playback functionality can be useful for learning. The author used this method years ago when first entering the test automation world and found it helped identify methods or calls needed in sticky situations. Relying on this method exclusively can only lead to disaster but it is useful for answering 'How would I do that?' type questions.

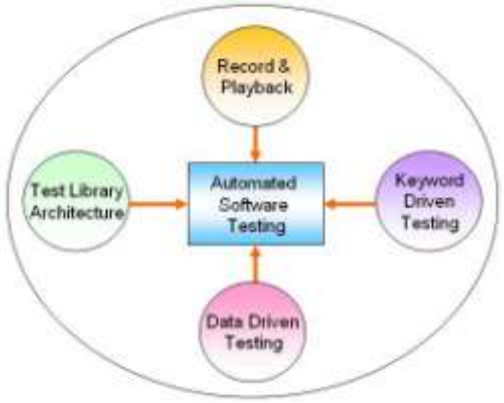


Figure 1: Amalgamation of Automated Software Testing Methodologies

Limitations and Problems

Like any software testing activity, Automated Software Testing has its share of limitations and problems. Identifying these ahead of time can help a test team in a variety of ways. First, it can help answer the basic question of whether the team wants to go down the path to automation or not. Sometimes the potential problems are more daunting than the team can handle at present. Second, understanding the limitations ahead of time can help set expectations. Potential setbacks are generally easier to cope with if the team sees them coming rather than having them appear out of the blue. Finally, it focuses the QA team's automation efforts. Knowing the potential problems helps the team focus their efforts in other areas where they can be more productive.

Complexity

Among the most obvious, and in the author's opinion most overlooked, problem to implementing a software test automation strategy is the complexity of the automation tools. Most of the higher end tools have a steep learning curve. Not only do they have their own full fledged scripting language, with syntax and usability issues, but the more robust and flexible the tool, the more complex and difficult it is to learn. The team that underestimates the time and effort required to become proficient with an automation tool should first clear some space on a shelf. They'll need it to store the box and manuals since they will quickly become disenchanted with their little test automation experiment. Every vendor touts ease of use and user friendliness as a benefit of their particular product. All of them are exaggerating.

Sticking to Best Practices

Test scripts need to be written correctly or else they end up being more trouble than they are worth. The 'correct' way to write an automated test script varies with the tool, but there are some general principles to follow. Scripts should be easy to maintain, make use of variables, and be modularized.

The scripts need to be easy to maintain. Not only does someone who didn't write the script need to be able to edit and update it, specific lines or functions need to be easily accessible. Scripts written in one person's short hand or without adequate comments are not easy to maintain.

In addition, correctly written scripts will make use of variables. So often during the creation phase of test automation, the tester hard codes values in their scripts. Almost as often, they never go back and replace the hard-coded values with variables. When the value needs to change, updating a variable is generally much easier than hunting through lines of code looking for all the instances of the hard coded value!

Finally, test automation scripts that are written correctly will be modularized. In the previous example of the File Open script, any changes made are automatically included every time the script is called. Conversely, fifty different scripts each with their own File Open routine requires fifty different edits if something changes. If the QA team doesn't write their test automation scripts correctly to begin with, maintaining those scripts will become a job unto itself, and the difficulty can quickly outweigh the benefits gained from automation.

Do as I Say…and Nothing Else

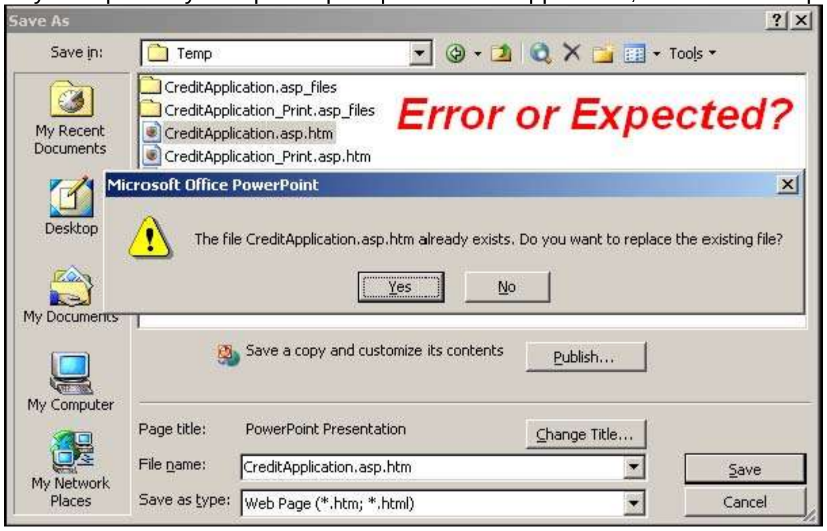
Another limitation of software test automation (or benefit of human testing if you prefer) is that humans can notice issues that test automation ignores. Automated scripts only verify what they are told to. They have almost no ability to notice things for which they haven't been programmed. It's true that automation will fail if an expected object is removed from a page, but what if an object is added to a page that shouldn't be? The tool will simply keep right on testing without error. Humans often notice issues by accident. The tester may be looking at a screen and notice a completely unrelated problem. The automated script will not.

Automated testing tools do not have the 'human variability' factor. Two different testers can test the same piece of software and get different results. Why? One may click OK with the mouse while the other just presses enter. One may enter a contact into the system as John Smith and the other may choose to use the name John O'Donnell. It may be that the apostrophe throws the system off. Maybe the test team would have known to test for that and maybe not. The point is that automated testing tools perform their tests in exactly the same way every time, regardless of which tester kicked them off. The automated test tool will never stumble across a problem. Human testers do so all the time.

The Unexpected

Very few test automation tools have built in error handling and even for the ones that do, the error handling is often incomplete. While the team can write error handling routines, there is a catch. The team has to know to expect an error in order to write a routine to handle it. Human testers on the other hand can and do handle unexpected errors dynamically. They can judge the severity of the error and if or how to continue the test. The test script just stops.

In the same vein as unexpected errors are just plain unexpected events. Sometimes things happen that weren't expected at test creation times that don't necessarily have to be errors. What if the test script saves a file and gets prompted to overwrite the existing file? See Figure 2: Is this an Error or Expected? When the script was written, the file didn't exist and so the prompt didn't appear. It may be a perfectly acceptable prompt within the application, but the test script just failed all the same.


Save As
Save in:
Temp
Tools
My Recent Documents
Desktop
My Documents
My Computer
My Network Places
CreditApplication.asp_files
CreditApplication_Print.asp_files
CreditApplication.asp.htm
CreditApplication_Print.asp.htm
Error or Expected?
Microsoft Office PowerPoint
Yes
No
Save a copy and customize its contents
Publish...
Change Title...
Save
Cancel

The file CreditApplication.asp.htm already exists. Do you want to replace the existing file?

Page title: PowerPoint Presentation

File name: CreditApplication.asp.htm

Save as type: Web Page (*.htm; *.html)

Figure 2: Is this an Error or Expected?

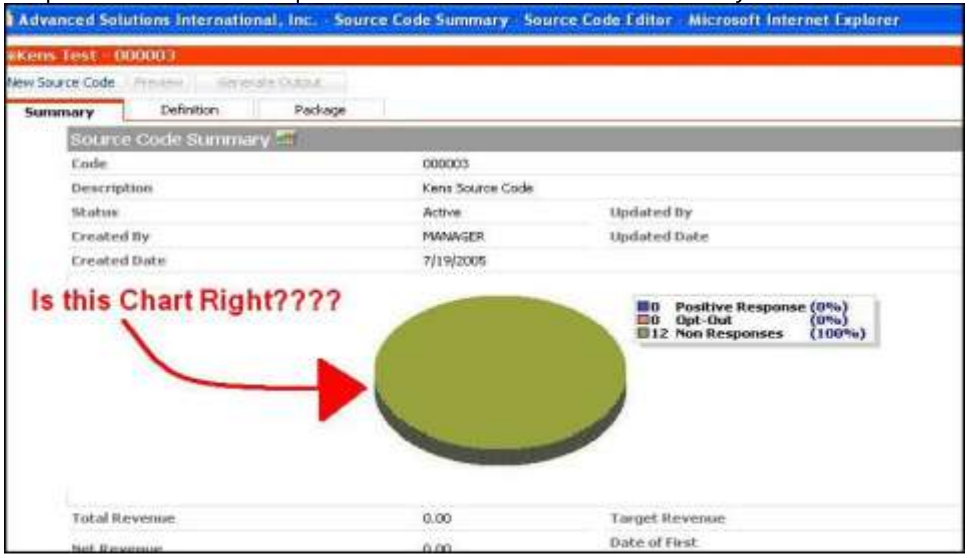
Timing and Workflow

Traditionally automated test tools have had issues with work flow and timing related problems. The responsiveness of web pages and databases can vary moment to moment. Automated testing tools will go along at a fast pace whether the application keeps up or not. A script that attempts to click on a button that hasn't appeared yet will fail just as if the button never appeared. Newer versions of some of the tools have some built-in timing logic. The tool will look for the object first before acting on it. Generally, the user can specify how long the tool looks for the object before reporting a failure. Other tools require the user to write this sort of logic into the script. Either way the test team should expect to have to deal with timing issues one way or another.

Images

While test tools have evolved significantly over the last decade or so, they all still have an Achilles Heel: Images. See Figure 3:

Graphs and images can be problematic. Graphics are an increasing part of modern user interface; unfortunately, automated test tools remain unable to deal with them in a meaningful way. Applications that chart or graph present a challenge to the test automation team. There is simply no way for a test tool to look at a pie chart of expenditures, for example, and verify that it is displaying correctly. Some test tools offer a screen capture and compare feature that is supposed to handle graphics; however, this screen shot compare feature is fragile. Comparisons can fail because of differing resolutions, fonts, colors, whether the window is maximized, etc. The author has never found a satisfactory way of dealing with graphics when doing automated testing. The best compromise has been to capture the screen to a file to be reviewed by a human after the test has completed.



Code	000003		
Description	Kens Source Code		
Status	Active	Updated By	
Created By	MANAGER	Updated Date	
Created Date	7/19/2005		

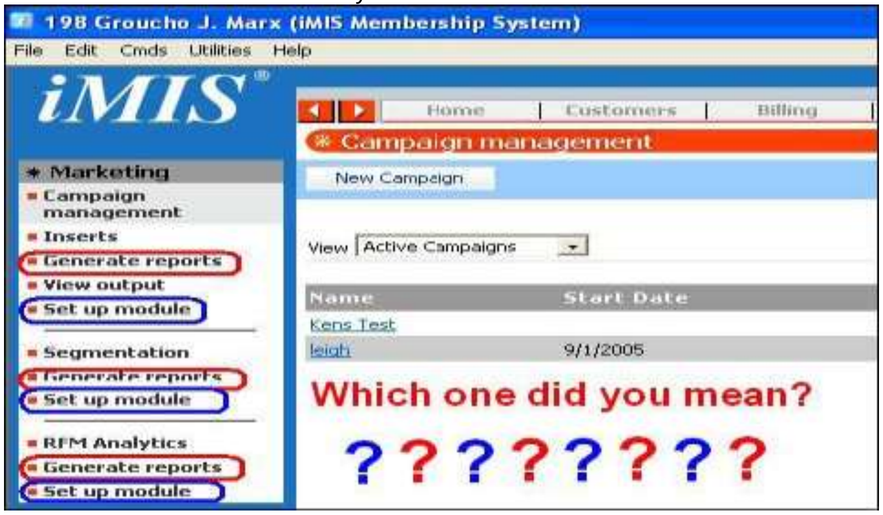
0	Positive Response	(0%)
0	Opt-Out	(0%)
12	Non Responses	(100%)

Total Revenue	0.00	Target Revenue	
Net Revenue	0.00	Date of First	

Figure 3: Graphs and images can be problematic

Context

When writing automated tests, the QA team needs to be very specific with context. See Figure 4: Context Matters. For example, when adding a contact to a database, the tester may intend to add a business address for the contact. The automated testing tool, however, insists on adding this address as a home address. Why? Because the home address fields are listed first and have the same labels as the business address fields. A human could have easily caught the problem, but the testing tool has little concept of context. Careful planning on the part of the test team can avoid such problems, but again the team needs to know about it before they can work around it!



View: Active Campaigns

Name	Start Date
Kens Test	
leigh	9/1/2005

Figure 4: Context Matters

Test Application Complexity

One potential limitation to using automated testing is that the QA team can sometimes end up rewriting parts of the program under test in order to verify the program is working as expected. The more complex the application, the more complex the test script ends up being. For example, suppose the team is attempting to automate testing on an application that calculates interest and payments on a loan. The team can either enter static data, such as a loan of amount X and interest rate Y leads to a payment of Z or the team can write the script such that it randomly generates the loan amount and interest rate and verifies the payment is correct. In order to write such a script, the team would need to have the automated testing tool perform the loan-related calculations. However, this is exactly what the application under test is supposed to be doing! So the team has just re-written part of the application in order to test it!

Return on Investment

Another obvious but often overlooked limitation of AST is that it only pays off on subsequent runs. Unless a test is going to be run over and over again there is no point in automating it. If the team finds a tool they truly love, and can use, they may fall into the trap of automating everything on the premise that they might need to run it again some day. This is a trap that can suck lots of the team's time writing scripts for tests that will rarely be run. Automating a test and running it only once will cost more in time and resources than simply running the test manually.

A final limitation of automated software testing is that there is no early payback. Implementing an automated testing tool and methodology takes time. It takes time to learn the tool, set up the environment, write the test scripts, debug the scripts and make them robust. In the beginning of the process, even simple tasks seem to take forever. All interested parties need to understand that the benefits from test automation don't materialize over night. Exactly when they do materialize depends on great many factors such as the expertise of the test team, the complexity of the tool, the priority of the automation project within the organization, and the complexity of the automation attempted. If the company needs the results yesterday, they will be disappointed when they don't materialize until tomorrow. AST does yield a return on investment, but it is a long-term gain, not short-term.

Benefits and Advantages

With all the problems and limitations of automated software testing why would anyone attempt it? Because there is an upside and a test automation suite is a thing of beauty when it works. So, just what are the upsides? AST obviously can't do everything, so what can it do?

Regression Testing

Regression testing is where it's at. Automated test tools really shine when used for regression testing. If a QA team needs to know quickly if new code broke previously working code or if a previously fixed bug has reemerged, then Automated Software Testing is the answer.

AST works well for regression testing for several reasons. First and foremost, the application being tested is more or less stable. Nothing disheartens an automation effort more than having the UI completely change in an area that has been automated. Also, unless the company's design and planning stages are flawless, the product isn't usually 'done' until it ships out the door. Automating an in-development application is possible but the team has to expect setbacks and lots of rework. It's best to have a stable UI before starting a test automation project. On the other hand, regressing an existing application provides a stable platform for developing the test scripts. Existing functionality can be retested with the click of a button and if a new feature or bug fix broke something else, the team will know it. Also, during time crunches more of the application can be regressed than with manual testing alone. Rather than having to pick the most likely trouble spots to test, the automation team can completely regress the application in a matter of minutes or hours.

A variation on the regression testing theme is what the author glibly calls **Bug Report Automated Testing** or **BRAT**. Despite the silly acronym, BRAT is a useful regression technique that involves writing an automated regression test for each bug that is found and fixed in an application. As a company releases patches or versions (version 1.1, 1.2, etc.) of an application, they normally fix a number of bugs along with any added enhancements. Each of these bug fixes can become an automated script that then becomes part of a larger automated test suite. Each script is somehow labeled or commented with the bug report ID so that any failures can be tied back to the issue that was fixed. Any tester who has ever seen a 'fixed' issue reappear can appreciate the value of this exercise!

A side effect of using test automation for regression testing is that it allows for greater test coverage. Not only will the tool handle the basic regression testing, but the team can also be performing more elaborate tests that aren't eligible for automation. This is especially useful in emergency situations where a critical patch may need to go out in a matter of hours or days, test automation will allow for much more test coverage than could be done by the test team alone.

Configuration Testing

Configuration testing is another area where automated testing shines. Most modern PC software supports a variety of configurations, based on the user's preference. The software may support multiple types of databases or run on multiple versions of the operating system. It may even interact with different web browsers or versions of the same web browser. Regardless, each of these 'configurations' will need to be tested. With an automated tool, you can write a core set of tests and then run the tests over each configuration. Select a 'base configuration' and then create an 'error free' core set of tests. Then change one of the variables, say the type of database on the back end or the type of server the client is talking to. Then run the tests again. If there are any errors, they will almost certainly be attributed to the configuration change and the team will have steps to reproduce the error!

Load Testing

Load Testing is a popular use of automated testing and is an art unto itself. This typically involves testing how an application performs when it has a 'high load.' What the 'high load' is exactly depends on the application, but typically it can be requests to the server (such as a web site or a client server application) or requests to a database (such as product or inventory management applications). Rather than have 1000 testers all try to log on at once, the team could write a script that does the same thing and scale it up. Certainly, 'scaling it up' is far easier said than done and most automated tool makers provide a companion tool to help the automated tester do just that. We'll leave this topic for now as it commands a more thorough analysis than can be done here.

Speed

And let's not forget the first advantage that comes to most people's minds: automated tests are fast! A test suite can be run very quickly and give results in a fraction of the time a human tester could do the same thing. Granted, it does take some time and effort to get an automated testing program to the 'push a button and let it go' stage, but once it is there, the results usually come in faster than the human operator can review them. One caveat to speed is that sometimes the automated tools can be too fast. If the tests aren't written properly, they can sometimes get ahead of the application under test and start reporting false errors. However, having to slow a test down is a much better position to be in than the other way around!

Benefits for Human Testers

Relieves Monotony

Another advantage to AST is that it can relieve the test team of some of the monotonous testing duties. Every tester knows that there are some tasks that are dull as dirt but need to be done anyway. It's during this time that the term test monkey seems all too apt; after all, a monkey
pressing random keys could do this part of the job! On the other hand, these tasks could be automated, freeing up the test staff for the tasks that really require their expertise.

Tunnel Blindness

The automated tester notices the details that humans learn to ignore. It is very easy for humans to develop tunnel vision or focus on one area to the exclusion of all else. Automation never misses 'the little details' once programmed to look for them. Tests and checks for the minutia can easily be added to any test suite; don't worry, the program never gets tired of looking at them!

Ease of Scheduling

Automated testing tools also have the advantage of not caring when they have to go to work. Schedule them to kick off overnight, or on the weekends, or make them work over Christmas for that matter. They won't complain one bit. Schedule the automated suite to kick off after each new build, even if the build is done in the middle of the night, the testing will start as soon as the application is built. For that matter schedule them to work continuously twenty four seven, they don't even take a coffee break.

Repeatability

Automated Software Testing has the advantage of providing repeatable tests. Who hasn't run across a bug and then not been able to reproduce it? Or had a bug fix that seems to pass at the time, only to have the customer report it again? The team is certain they tested it, but did it slightly differently this time and so missed the bug! Automated test suites allow tests to be repeated the same way every time or nearly every time, anyway. There are still some factors that are beyond the control of the test tool. For example, if the issue was caused by the server load spiking at that moment, running the test again may not reproduce the error. Those limitations aside (which a human would be subject too as well), the automation suite provides the advantage of having repeatable tests.

Expandability

The final advantage to automated testing is how easily Data Driven Tests can be expanded. To return to the simple example of a test suite that enters new contacts into a database. The list of names, addresses, phone numbers, etc. is in a table of some sort and the test simply iterates through the table entering each contact in the database. Now what happens when the team needs to add tests that use foreign characters or someone with three middle names? Just add another row or two or ten. The test script doesn't care and the team has just easily expanded the testing scope. Commerce applications can have any combination of products, quantity and payment methods the test team needs. And so on, as long as the basic steps are the same each time.
Reassembling existing test components can create new test suites with minimal effort. If the test suites are designed correctly from the beginning, the existing automated components can be reassembled into new tests with minimal effort. As a simple example, the module that logs a user on does not always have to be followed by a script that enters a new contact. In this case, reassemble the tests into a suite that logs the user on and then places an order. The team should reuse automated components whenever possible.

Structure, Technology and Best Practices

This final section briefly hits on the basic structure common to all test automation programs and some basic technology and terminology the test team will need to be familiar with: data, base states, decision making, logging and reporting.

Data

One of the key components the test team needs to consider is the test data. Most tests of any significance will require inputting data of some type. While the data types can vary from text to numbers to dates to whole files, the data input falls into one of three broad categories: Hard Coding, Data Driven, or Dynamically Generated.

Hard Coding

To Hard Code the data is practically a curse word. It simply means that some piece of data, such as the executables location for example, is written directly into the test script or code and is never accessible or modified when the test is run. Changing the value requires editing the source code directly. This is a very bad idea and should be avoided at all costs. None of the data should be hard coded into an automated test. Even seemingly permanent pieces of data such as the program's name or installation location should be provided via a variable of some sort.

This is not to say that hard coding does not have some uses. In the early stages of development, when test scripts are first being created, basic data can be hard coded to facilitate the script creation. A script that enters a contact into the program could hard code the name and address information while the script is initially created and debugged. That simply allows the tester to tackle one issue at a time; make the script work first and then worry about retrieving and plugging in the data. Following this approach though, requires replacing every piece of hard coded data with variables at a later time. Like Record Playback, it has some uses, but shouldn't be the norm.

Data Driven

A more useful data model is to create Data Driven Tests. As mentioned earlier, this involves providing the data in a file of some sort such as a spreadsheet, text file or XML. When the automated test is executed, the automation tool reads the data as needed and plugs it in to the test in the appropriate place. To return to the contact entry example, the contact name and address would be listed in this data source. Changing the data simply requires updating the file.

The team does need to closely coordinate on data format however. Dates are one notorious example of the need to enforce some sort of standard. Is the format month-day-year or daymonth-year or year-month-day? Do you require four digits on the year? Is the month in digits or spelled out? Is it spelled completely or abbreviated? Do single digit days have a leading zero in front? Enforcing a standard format for data will reduce the number of test failures that are really nothing more than faulty data.

Using a data driven approach allows tests to be easily expanded as well. Instead of hard coding a single contact to be entered in the program, we can now provide an entire list of contacts. Depending on the exact nature of the test, the data may be completely non-standard. The date example above may mean that dates are provided in every conceivable format because the test is whether or not the data gets normalized. Or perhaps the test is looking for the appropriate error message to be displayed.

Dynamic Data

Finally, test data can be dynamic. A 'random' function in the test case can sometimes help find errors that using the same test data over and over will miss. For example, perhaps the automated test creates new contacts each time by selecting a first and last name from a long list of possibilities. Perhaps the birth date is randomly generated. Perhaps the price of a product or the payment amount is a random number.

The test team needs to decide whether using dynamically generated data serves their needs. While random data does sometimes uncover unexpected errors, it also introduces additional uncertainty into a test. If the tester needs to login with one of the contacts just dynamically generated, how does he or she know what the login name will be?

Using dynamic data also requires placing bounds on the data. Creating a random birthday for the contact is fine up until the point that they end up being born in the future. Generating random characters to serve as a 'name' is not useful if name ends up being three thousand characters long. Likewise, perhaps valid order quantities should not be negative numbers or over three digits. Generally, the bounds should match the specified bounds for the program under test. (An exception would be when performing a 'negative' test; i.e. - when the team is trying to break the program or assure it responds appropriately to 'bad data.').

Base State

Making use of a 'base state' is a common practice that involves beginning and ending all tests suites at the same point. That point may be a home page or login screen or maybe the computer desktop after the program has been closed down. So, for example, when entering a contact, the automation tool loads the program, logs in, navigates to the contact page, enters and saves a contact and then navigates back to the login screen. While the test itself may be over after the contact is saved, the test script is not done until it returns to the login screen.

Using a base state helps multiple tests run more smoothly by always starting and stopping them in the same place. The 'contact entry' script may be followed by the 'purchase a product' script. If the Contact Entry Script leaves the application on the contact screen but the Purchase Product Script starts off trying to log in, it will 'fail' because the information isn't where the script expects it to be. Also, using a base state allows for multiple team members to create scripts in parallel with the confidence that they will always know what state the program will be in at the time their script is called.

Decision Making

Any test of sufficient complexity will, at some point, have a decision to make. Every test tool provides a mechanism for handling these decisions to a greater or lesser degree. They may be referred to as If/Then statements, such as in the phrase, “If this happens, then do this.”

Many times the decision will be a type of error handler. Perhaps the test script checks for the existence of a contact. “If a message saying the contact exists is displayed, then select OK and move on to the next contact in the list.” Or perhaps it checks to verify the user is already logged in and if so, it skips the login procedure. Other times, if/then statements simply create a ‘smarter’ test script. “If the product ordered is sold out, then verify a backorder is created.” Checking the list of backordered products won’t be something that the script checks every time, only when the situation calls for it.

In either case, allowing test scripts to make decisions based on specific criteria makes the scripts more robust. Otherwise, the team is likely to encounter more false failures, when in reality, the situation was valid, but the test script did not it handle properly. Learning to use a test tool’s decision-making capability will be a critical part of ensuring the test automation effort is successful.

Logging and Reporting

Every automated test effort will need some sort of logging and reporting capability. The capability may be built into the tool and the test team simply needs to enable it. Other times, any logging will need to be explicitly done. Most of the time, the log only becomes important when a test fails.

What to log and how much data to keep has been an ongoing debate. On one hand is the ‘keep everything’ school of thought that believes that if the data can be captured, it should be logged. The downside, of course, is reading through these logs to discover what caused a test failure. A test log filled with completely irrelevant minutia simply frustrates the test teams’ efforts at uncovering the problem. Additionally, a test log that simply says ‘Test Failed’ without any detail is equally useless.

Taking the lead from a typical test case will generally provide a usable automated test log. As with most test cases, whether automated or manual, the log should collect any setup information, any previous steps, the expected results and the actual results. Creating a log that is human-readable is critical if the log is to be a useful debugging tool.

Depending on the test tool, it might be able to save a screen shot of the failure. This is especially useful in cases where the automated tests are run unattended or overnight. A screen shot reveals exactly what state the program was in when it failed. If the automated test tool of choice does not provide a screen capture capability, consider making use of a simple utility that does so. After all, a picture is truly worth a thousand words (in a test log).

A test report will typically be generated from test logs. At a minimum, it should include which tests passed or failed. In addition, it may also include data such as the test execution time or summaries of test failures by module, which would show whether a particular screen or module is exceptionally prone to error or whether the application is experiencing performance degradation.

Making use of Programming Best Practices

There is no need to reinvent the wheel with AST. It has quite a bit in common with actual programming and should be treated just like any other programming task. The fact that many programs shield the tester from the actual ‘code’ doesn’t change the fact that the project should be treated as a ‘coding’ project.

The test team will need to come up with a basic test design and agree on an implementation plan. They will need to consciously migrate to the automated test suites and communicate the changes to everyone involved.

The team needs to set aside time for writing and debugging the test scripts just like any other program. They will also need to plan to make use of subject matter experts if they hit unexpected obstacles.

And perhaps the most overlooked programming practice, in the author’s opinion, is source code control! Just as in any programming project the team will need to be able to easily revert to previous versions of the code and be able to log and control who makes changes.

Conclusion- What Now?

Software Test Automation can be an invaluable tool in the software quality assurance team’s arsenal. Though automating tests can be a challenge, the team stands a better chance of success by being prepared, informed and realistic about what test automation can accomplish. Picking an automation methodology that fits the team’s situation and avoiding poor automation practices is a key to success. Understanding the limits of automation will help the team the avoid pitfalls; just as knowing where automation is useful will help the team maximize their investment in both time and money. The team will need to develop core skill sets in not just the technical aspects of the tool itself, but in broader areas such as data management. Being prepared and having a realistic understanding of the challenges and benefits of Software Test Automation will increase the odds that the team will successfully implement their automation strategy and not leave the box of automation software collecting dust on a shelf.

The first place to start is with the software under test. Conduct an honest review based on the guidelines discussed in this paper. Keep the following questions in mind when evaluating the software:

- Is the application truly a candidate for automation?
- Is the application too graphic heavy?
- Does it change so often that the automation team would never make any headway?

Second, decide on a proper proof-of-concept. The first implementation needs to be useful, doable and impressive. The team needs to pick a test to automate that will provide the most return for the effort. Is there a set of tests that get run with every build or every drop? Are they perhaps simple to do but monotonous and time consuming? Will upper management be impressed when they see the automated tool doing these tests? The team's first attempt at automation should be something useful to the team, even if nothing is ever automated again. It should also be doable; there is a lot to be said for keeping it simple. And finally the proof of concept should leave management with a positive impression of the team's efforts and automation in general. This will make it much easier for the team to garner support for the cost and time required to adequately implement an automation plan.

Third, evaluate test automation tools based on the team's needs. Not every tool will work well in the team's testing environment or with the program under test. Some tools may be too complex for the non-technical testers on a team to use. Some may lack the flexibility required for long term use and thus become obsolete quickly.

Fourth, come up with a plan. The team should decide who will do the implementation, what time will be allocated for the work, when the proof of concept should be completed, etc. Doing test automation in the team's spare time is the quickest way to assure the team has just purchased shelf-ware.

Finally, just do it! Implement the plan and enjoy the fruits of your labor!

References

• Alam, M.N., "Software Test Automation Myths and Facts," (Unknown),

http://www.benchmarkqa.com/pdf/papers_automation_myths.pdf

• Bach, James, "Test Automation Snake Oil," (1999),

http://www.satisfice.com/articles/test_automation_snake_oil.pdf

• Black, Rex, "Manual or Automated? Investing in Software Testing, Part 5," (July 22, 2002), http://www.stickyminds.com/sitewide.asp?ObjectId=3583&Function=DETAILBROWSE& ObjectType=ART

• Dustin, Elfriede, "Introducing Automated Testing to a Project," (May 25, 2001), http://www.awprofessional.com/articles/article.asp?p=21479&rl=1

• Dustin, Elfriede, "Lessons in Test Automation: A Manager's Guide to Avoiding Pitfalls When Automating Testing" (May 25, 2001), http://www.informit.com/articles/article.asp?p=21467&rl=1

• Dustin, Elfriede, "The Automated Testing Lifecycle Methodology (ATLM)," (May 25, 2001), http://www.informit.com/articles/article.asp?p=21468

•

• Hays, Linda, "Why Automate?," (2004),

http://www.worksoft.com/ContentDisplay/G13/G13L49.asp

• Hildreth, Sue, "Software QA 101: The Basics of Testing," (September 3, 2004),

http://www.informit.com/articles/article.asp?p=333473

• Kelly, Michael, "How effective is Your Test Automation?," (May 6, 2005),

http://www.informit.com/articles/article.asp?p=379757

• Marick, Brian, "When Should a Test be automated? ," (2005),

http://www.testing.com/writings/automate.pdf

• Mercury, Inc., "Implementing an Effective Test-Management Process," (2005), http://whitepaper.informationweek.com/cmpinformationweek/search/viewabstract/60217/i ndex.jsp

• Mercury, Inc., "Implementing Automated Functional Testing Solutions: Increase the Speed, Quality, and ROI of Testing Business Applications," (February 01, 2005), http://whitepapers.businessweek.com/detail/RES/1110382082_24.html

• Segue, Inc., "How to Successfully Automate the Functional Testing Process: How automated functional testing can optimize software quality and drive business value," (2005), http://www.knowledgestorm.com/sol_summary_73354.asp?trkpg=provsummary

• Segue, Inc., "Silk Test 6.0 Reviewers Guide," (2003),

http://www.segue.com/pdf/silktest_reviewers_guide.pdf

• Segue, Inc., "Transforming Application Quality Through Software Quality Optimization (SQO)," (2005), http://www.knowledgestorm.com/sol_summary_73003.asp?trkpg=provsummary

Back to top

Beyond Usability: Improving Software Quality by Closing the "Expectation Gap"

Nicole M. Radziwill Assistant Director, End to End Operations,
National Radio Astronomy Observatory Charlottesville, VA – nradziwi@nrao.edu

Abstract – Even software projects that meet their requirements, and come in on time and on budget, are not guaranteed to be successful. This may be a result of the "expectation gap" between the written requirements, the assumed requirements, and the ever-increasing consumer expectations for quality in the 21st century. Where does this gap come from, and how can software quality professionals effectively address it? Theory and findings from human factors, ergonomics, and design science (an emerging focus area for software intensive systems development) can help us understand and shrink this expectation gap, improving the perceived quality of the software that we produce.

Keywords Affect, affective design, customer satisfaction, objective quality, perceived quality, perceived value, user-centered design.

Introduction

Software quality professionals fundamentally work to make software and the software development process more effective and productive for the companies and organizations we work with. To continually improve products and processes, we seek interventions in all aspects of the software development lifecycle: we pursue better methods for initially articulating requirements, better approaches to translate ambiguous and incomplete requirements into meaningful designs, and leaner, more agile development processes that aim to involve the user more completely and more effectively from the beginning stages of development. Each of these thrusts is related to usability, and to ensuring that the system which is developed can satisfy a specified user's needs within a context of use.

Using systems thinking to explore the role of usability in the software development lifecycle, this paper seeks to help the reader understand the causes, effects and implications of usability factors that contribute to achieving quality in software development. These findings about usability and quality are synthesized to provide actionable recommendations for how the software quality professional can help to close the expectation gap throughout the product development process.

Quality Defined

There are many definitions of quality, including fitness for use, zero defects, and conformance to requirements. Despite the range of definitions, the goals underlying the pursuit of quality are the same: reducing variation, eliminating waste and rework, preventing human error, preventing defects, improving productivity, and increasing efficiency and effectiveness Unfortunately, all of these definitions seem to leave one or more aspects of quality out. For example, conformance to requirements is less useful if the requirements are incomplete, and zero defects is a laudable goal for traditional manufacturing systems where processes are well known, but a less useful measure for unstructured problems where the definition of a defect could be emergent.

Before we can take a critical and fundamental look at the relationship between quality and usability, we must reflect upon the definition of quality itself. ISO 8402 defines quality as "the totality of features and characteristics of an entity that bear on its ability to satisfy stated or implied needs." These needs can be relative to any stakeholder: the customer, a user of the system, a different software system, the organization sponsoring the project, or even another organization (e.g. if a stated or implied need is to become qualified or certified as a supplier).

Usability and its Relation to Quality

ISO 9241-11 defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." There are four elements that define usability within this definition: both the users and goals must be explicitly identified, there must be a context of use identified, and it is implied that the user can use the system in question to meet those stated goals. Note that these same four elements are implied by the definition of quality: stated and implied needs are relative to specific users with specific goals, are dependent upon a context of use, and the entity in question is the system being defined and developed in response.

Usability is the extent, or the degree, to which the above criteria are satisfied. The software development lifecycle inherently addresses usability through these four elements: the requirements process outlines the specified users, their goals, and the context of use; the design process defines a specific technical solution to meet those needs. As a result, usability can be considered an implicit factor in software quality, ultimately reflecting how well the design process interpreted requirements within a specified context of use.

Difficult Systems Make Waste

According to the U.S. Department of Health and Human Services at http://www.usability.gov, *people will avoid using difficult systems.* This behavioral avoidance is inherently a waste generator: people waste time using the software
because the required functionality is not easily accessed or executed; they waste their own time and the time of other people as personalized user support is required to wade through (and hopefully resolve) the issues; and changes and rework will be required to systemically respond to these usability problems. The frustration that comes about when an individual attempts to use a difficult

system can have longer-term impacts, as well – once they balk, it will be a much more difficult process to get that user to try the same software again.

When usability is considered as an essential quality attribute of a software product, it is possible to help users experience a feeling of achievement without frustration. This promotes *positive affect* which is the psychological key to usability.

Perception and the Role of Affect

A software product may be feature-rich and perform effectively, but still not produce the customer satisfaction or generate the sales that were desired. Understanding the psychology of quality and value, based on affect, provides insight into how this situation can arise. Merriam Webster's Medical Dictionary defines affect as "the conscious subjective aspect of an emotion considered apart from bodily changes." In short, affect characterizes *how something makes you feel*. Research in the field of psychology indicates that positive affect corresponds with the ability to solve problems more readily and effectively, while negative affect can impede problem solving, even for simple tasks. As a result, usability can be considered a function of the positive or negative affect that is generated when a user interacts with a software product.

Two empirical studies have reinforced the finding that positive affect promotes the positive perception of usability. Kurosu & Kashimura (1995) studied whether the attractiveness of a device (specifically, an automated teller machine at a Japanese bank) influenced its perceived ease of use. They observed hundreds of users of two ATM machines, one with an ordinary design and one with aesthetics taken into consideration. Both machines had identical functionality, and performance was also comparable between the two. Through surveys, the found that people believed the prettier machine functioned better and was easier to use. Tractinsky (1997) performed a follow-on experiment in Israel, hypothesizing that the Japanese culture was more easily influenced by aesthetic factors, and so the results would not be reproducible in a different culture with a more austere sense of aesthetics. His study, which also varied only the designs of two ATM machines and not functionality or performance, yielded identical results to the Japanese study.

These studies suggest that effective design translates to positive affect – meaning that before use, perceived quality and perceived value are more closely related to the perceived quality and value that will be experienced after use. Aesthetics thus play a role in promoting positive affect. As interpreted by Norman (2004), "the emotional system changes how the cognitive system operates… [it is] easier for people to find solutions to the problems they encounter… [there is a] tendency to repeat the same operation over again is especially likely for those who are anxious or tense."

Objective and Perceived Quality

Affect plays a part in the perception of quality, but not in measurable (or objective) quality, which is the degree to which a product satisfies specified attributes. This is a repeatable measure, meaning that different individuals assessing the quality of a product against specified attributes will generate reasonably consistent results. Conformity to requirements is a measure of objective quality. Perceived quality is "an individual's overall assessment of product quality, which is based on the objective and subjective attributes of a product relative to the expectation of quality."(Mitra 2003) For example, an individual browsing through specifications of various desktop and laptop machines available for purchase will form assessments of perceived quality, possibly integrating independent reviews
or evaluations (objective quality) as input to the formation of the perception.

Objective quality will be the same both before and after purchase (or use). Perceived quality is entirely a pre-purchase factor, because customer satisfaction is the "post-purchase evaluation of product quality relative to pre-purchase expectations and perceived quality." (Mitra 2003) Did you get what you thought you were going to get? Did you get the level of functionality and performance that you thought you would, based on your perceived value of the product (the prepurchase perceived quality with respect to price)? Customer satisfaction is a post-purchase or post-use measure because at this point, the user will have had experience working with the software, and will be able to effectively assess whether or not the package truly meets his or her needs.

The Expectation Gap (ΔE)

The expectation gap was qualitatively defined by Watson (2003) as *the difference between what the customer wants and what the customer gets*. This is further broken down into the conformity gap (the difference between what the customer gets, and what the customer is promised) and the design gap (the difference between what the customer is promised and what the customer actually wants). The design gap indicates that you failed to realize the requirements through the design in a way that the requirements were readily achieved. Your design methodology was possibly lacking. The conformity gap, on the other hand, is a reflection of the customer's perceived entitlement, and means that you promised something that you did not or could not deliver; you did not conform to specifications effectively or consistently, or your specifications failed to capture the customer's entitlement. A schematic is shown in Figure 1.

Because both the entitlement and the expectation are mediated by expectation setting and thus can be managed, I refer to these together as the *expectation gap*. Since the gap can be managed through effective communications, it is not uncontrollable.

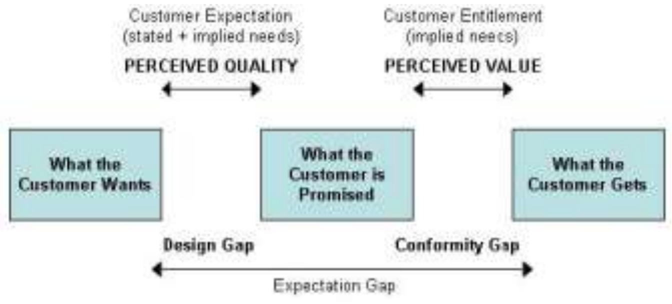


Figure 1. The “Expectation Gap” is the change between expectations, entitlement, perceived quality, and perceived value *over time*: before and after purchase or use. Adapted from Watson (2003).

The expectation gap is the net effect of the design gap and the conformity gap between the time that quality perceptions are first formed, and the time that post-use or post-purchase quality assessments are made. As a result, it can be expressed as follows:

$\Delta E = (PQprewpq + PVprewpv) - (PQpostwpq + PVpostwpv)$

PQ and PV represent perceived quality and perceived value assessments, pre- and post-purchase or use. Perceived value is a function of quality and cost, since an individual may be more willing to pay a higher price if they perceive that the quality of the product is better. Specific metrics for these terms can be extracted by examining the literature in quality, marketing, and economics. For example, the “willingness to pay” method, or “contingent valuation”, can be used to determine a specific measure for perceived value. Assessments of perceived quality can be extracted from survey data. Note also that perceived quality can be the net impact of meeting or not meeting many quality attributes that affect expectations.

For these expressions to be useful, the values of PQ and PV should be normalized so that ΔE is dimensionless. Because the metrics selected for PQ and PV can differ from project to project, ΔE values should only be compared within-project. As an example of how this expression might be useful, survey data characterizing perceived quality with respect to an array of quality attributes could be compared to survey data after software use by a group of beta testers evaluating
a pre-release version of a software product. If the expectation gap is excessive, corrective action might be prescribed.

To account for the factors that are most important to the target audience or target user group, a weighting scheme can be applied. Each of the terms in the expression for ΔE is weighted by which factor is more important to the user: if money is no object, and there are no barriers to use for a particular software product, then the weight assigned to perceived value (wpv) for this decision will be zero, and the expectation gap can be simplified to the difference in perceived quality before and after the time of purchase or use:

$$\Delta E = PQpre - PQpost$$

Customer satisfaction is the post-purchase or post-use, subjective evaluation of the expectation gap. By the time the customer determines his or her level of satisfaction, it may or may not be correlated with their pre-purchase assessment; in fact, the user may not even recall their previous expectations.

What Causes ΔE?

Using the quantitative description of the expectation gap, we can explore how the expectation gap comes to exist.

Improperly Set Expectations. When users are programmed to expect more than can reasonably be delivered, PQpre will be excessive and this will inflate the expectation gap. Low-cost or ease-of-use value propositions, if not met adequately, can inflate PVpre which will also increase the expectation gap.

Time. Over time, new information is acquired, and the way in which people interpret and respond to requirements and the designs that embody those requirements will change. If the requirements and design process can adapt to those changes in learning and needs, the change in PQ over time will be minimized and the expectation gap will be punctuated.

Shift in Relative Importance. Over time, an individual’s prioritization of quality attributes and value assessment can change, and this will change the weighting in the expression of the expectation gap. Consider the example of a consumer thinking about purchasing a statistical software package. If the cost is high and many advanced features are provided, but the customer is a student, the product may rate high in terms of perceived quality, but the expectation gap will be cost and value-driven. Once the student is employed and their income level increases, that value-driven weighting might change, and willingness to pay could increase, shrinking the expectation gap as a consequence of the individual’s change in status.

Closing the Gap through Design

The design process itself presents a mechanism for shrinking the expectation gap. One of the recommended paths to achieve usability is through the principles and practices of user-centered design. ISO 13407 characterizes user-centered design by describing its features: “human-centered design is characterized by the active involvement of users and a clear understanding of user and task requirements, an appropriate allocation of function between users and technology, the iteration of design solutions, and multi-disciplinary design." According to this definition, agile methodologies could even be considered ways to achieve user-centered design.

Actionable Recommendations to Improve Software Quality

Users of software systems are unlikely to read the requirements documents or elaborations that a development team so painstakingly seeks to satisfy. The users will never know what the system was specifically intended to do; they will only know what they believe the system *should* do *for them*. The process spanning requirements engineering and design is in place to communicate the vision for satisfying the user. The following specific actions can be taken to improve the quality of software by considering the requirements and design processes holistically and in a psychological context:

- Consider affect as part of the design process, and assess the impact your software product will have on your users through visceral, behavioral, and reflective (or self-actualization) satisfaction.
- Integrate assessments of perceived quality and perceived value into the product development process, possibly through user surveys.
- Most significantly, measure the expectation gap as part of user acceptance testing (or beta testing) to determine whether corrective action should be taken to shrink the gap before product delivery.

References

Kurosu, M. & Kashimura, K. (1995). Apparent usability vs. inherent usability: experimental analysis on the determinants of the apparent usability. Conference on Human Factors in Computing Systems, Denver, CO. New York: ACM Press, p. 292-293.

Mitra, D. (2003). An econometric analysis of the carryover effects of quality on perceived quality. PhD dissertation, Stern School of Business, New York University.

Norman, D. (2004). Emotional Design. New York: Basic Books.

Tractinsky, N. (1997). Aesthetics and apparent usability: empirically assessing cultural and methodological issues. Proceedings of the SIGCHI conference on human factors in computing systems, March 22-27, 1997, Atlanta, pp. 115-122.

Watson, G.H. (2003). Customers, competitors and consistent quality. In Conti, T., Kondo, Y. and Watson, G.H. (Eds.), *Quality into the 21st Century* (pp. 21-46). Milwaukee, WI: ASQ Press.

Back to top

CONFERENCES & WEBINARS

Software Track at QMD Conference in Irvine CA March 4-5

March 5-6, 2009 in Irvine, CA -- Quality Management Conference

ASQ QMD has partnered with the Software Division to bring Software Quality presentations designed especially for Quality Managers. Expert presenters will make software issues come alive in a 'non-techie' way, providing an opportunity to learn more about emerging topics of critical importance to Quality Managers worldwide.

Register at http://www.asq.org/qm/index.html

Back to top

ASQ Discount for SEPG 2009 in San Jose March 23-26

SEPG NORTH AMERICA 2009
ASQ Software Division

10% Discount for ASQ Members to Attend SEPG 09

Come visit the ASQ Software Division booth #401 at the SEPG North America 2009 conference March 23-26 in San Jose, California. ASQ Members receive a 10% discount on registration. Use this code when you register to receive the discount: NA09ASQ. To save even more, register before the early-bird deadline of February 20 to receive both the early-bird rate and your 10% discount. SEPG North America 2009 is the premier learning and networking event that brings together industry leading organizations who are putting CMMI, People CMM, TSP, PSP, Agile, Six Sigma, ITIL and ISO standards, and other performance improvement methods together to manage quality and performance. Browse the technical program, download the Preliminary Program, learn about keynote speakers, and find complete details on the website: http://www.sei.cmu.edu/sepgna

Back to top

WCQI 2009 Institute for Software Excellence in Minneapolis May 18-20

This is a full track dedicated to software quality at this year's largest ASQ conference. Find out more at

http://www.asq.org/conferences/institute-for-software-excellence/index.html

Back to top

ICSQ 09 Call for Papers and Mark Your Calendar for November

International Conference on Software Quality 2009
Theme: Controlling Software Before Software Controls You!
Conference Dates: November 10-11, 2009 (with pre-conference tutorials on November 9)
Conference Location: Northbrook, Illinois (Chicago)

Technical papers and panels should be practitioner-oriented. They may be based on research of interest to practitioners or on experiences that relate to any aspect of software quality. Tutorial/Workshop Sessions will also be presented. These are either half-day or full-day sessions that provide practical knowledge to participants. Workshops that promote the active participation of learners through problem solving, case studies, or other interactive learning methods are encouraged.

Important dates:
March 1, 2009: Abstract Submission Deadline
March 20, 2009: Presenters Notified of Acceptance
April 1, 2009: Technical Program Available Online

More information about
• Suggested conference topics
• Submission requirements
• Volunteer opportunities
• Conference exhibitors
• Conference sponsors
is available at http://www.asq-icsq.org

The ICSQ 2009 Call for Papers can also be accessed directly at
http://www.espresso-labs.com/icsq2009/ICSQ2009CFPV7.pdf

The conference web site will be updated as new information becomes available. Please check http://www.asq-icsq.org periodically.

Back to top

ANNOUNCEMENTS

Visit ASQ Software Division Management Committee and other Members

Booth 401 at SEPG NORTH AMERICA 2009

San Jose, CA

March 23-26, 2009

Come visit the ASQ Software Division booth #401 at the Software Engineering Process Group (SEPG) North America 2009 conference March 23-26 in San Jose, California. ASQ Members receive a 10% discount on registration. Use this code when you register to receive the discount: NA09ASQ. To save even more, register before the early-bird deadline of February 20 to receive both the early-bird rate and your 10% discount. SEPG North America 2009 is the premier learning and networking event that brings together industry leading organizations who are putting CMMI, People CMM, TSP, PSP, Agile, Six Sigma, ITIL and ISO standards, and other performance improvement methods together to manage quality and performance. Browse the technical program, download the Preliminary Program, learn about keynote speakers, and find complete details on the website: http://www.sei.cmu.edu/sepgna

Back to top

WCQI Surrounding States Special Pricing

All current and potential ASQ members who reside within surrounding states of
Minnesota will be offered a special conference rate of $735 per person. With this offer, members save $90 on the early-bird rate and $190 after the April 3, 2009, early-bird deadline. Nonmembers are also eligible for this offer and save $235 on the early-bird rate and $365 after April 3, 2009. This could be a great opportunity to recruit new members by allowing them to experience an ASQ event for a minimal cost.

To receive this rate each registration *must* include a special priority code that will be provided to you and you *must* reside within ND, SD, MN, IL, IA, or WI.

Back to top

ASQ Division Satisfaction and Loyalty Study

What can your ASQ division do to make your experience better? Take our survey and let us know. Your input gives division leaders guidance for creating opportunities focused on member needs. This allows your ASQ division to provide the products and services that are most important and beneficial to you.

In February, all ASQ members are being asked to provide feedback on how well their ASQ division is serving their needs. We would appreciate your participation in this survey. Your answers will be kept confidential and reported to ASQ only in the aggregate.

Participants will be entered into a drawing to win one of four 2009 ASQ World Conference on Quality Improvement registrations (a $925 value). The conference is May 18-20, 2009 in Minneapolis, MN. The study will be open through March 2. To participate, please visit http://www.asq.org/mr/09-forum-div-sat.html.

Back to top

COLUMNS

Get Certified

CSQE Quiz for 1st Quarter 2009 By Linda Westfall

1. The cost of fixing a requirements defect:
   - A. will decrease as more and more of the product is implemented and tested.
   - B. will increase the later in the software development life cycle the defect is detected.
   - C. is dependant on the severity of the defect.
   - D. remains constant as development progresses.

2. When documenting the process for conducting an audit on-site visit, which of the following would be included as part of the entry criteria?
   - A. The audit checklists are documented
   - B. The audit interviews are complete
   - C. The audit report has been approved
   - D. The opening meeting has been conducted

3. In a Client/Server architecture, the client typically:

I. manages the network resources.

II. centralizes storage of shared data.

III. runs on a decentralized computer.

A. II only

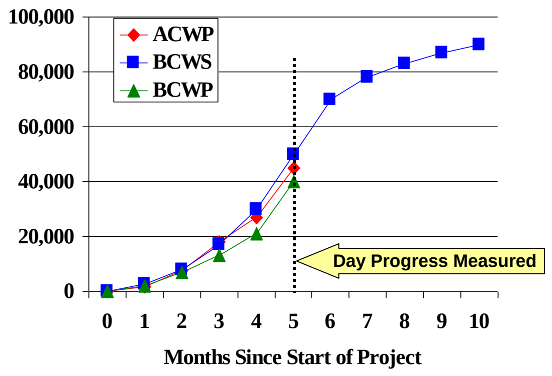
B. III only

C. I and II only

D. I and III only

4. The graph above reports the earned value for project ABC, which has been in progress for 5 months. Based on the information in this graph, at month 5 project ABC is currently:

A. over budget and behind schedule.



B. over budget and ahead of schedule.

C. under budget and ahead of schedule.

D. under budget and behind schedule.

5. Utilizing Cyclomatic complexity as a metric to measure the size of a software module is:

A. neither reliable nor valid.

B. reliable but not valid.

C. valid but not reliable.

D. both valid and reliable.

6. Which of the following is an example of a dynamic analysis verification and validation technique?

A. Running a spell checker on the user's manual

B. Compiling a software module

C. Integration testing a software/hardware interface

D. Peer reviewing a detailed design document

7. Which of the following is NOT typically a function of configuration identification?

A. Assigning unique identifiers to each product and its components

B. Identifying component, data and product acquisition points and criteria

C. Establishing product and component baselines

D. Tracking the content and status of each build

1. **Answer B is correct.** If a requirements defect is detected during the requirements phase and it costs one unit to fix (e.g., 3 engineering hours, $500) that same defect will typically exponentially increase in cost to fix the later it is found in the life cycle. For example, if it is not found until the design phase, costs increase because both the requirements specification and the design that was based on that requirement have to be fixed. If the defect is not found until coding, the requirements and design specifications and all of the code based on the defective requirement have to be fixed. By the coding phase, test cases and user documentation may also have been written based on the defective requirement that will need to be corrected. If the defect isn't found until the testing phases, all of these work products would have to be corrected, retested and regression tested. **CSQE Body of Knowledge Area: I.A.1**

2. **Answer A is correct.** Entry criteria are conditions that must be true before a process can be started. The audit checklists should be documented as part of the audit planning and should be complete before the on-site visit starts. The opening audit meeting and the audit interviews would be tasks that are defined as part of the audit on-site visit process. The audit

report is written after the completion of the on-site visit. **CSQE Body of Knowledge Area: II.A.4**

3. **Answer B is correct.** The client is typically an application that runs on a decentralized computer or workstation and allows the localization of processing. The client typically downloads or accesses centralized data that is stored on the server. The server also typically manages the network resources. **CSQE Body of Knowledge Area: III.A.2**

4. **Answer A is correct.** At month 5 project ABC is over budget and behind schedule. The cost variance = BCWP – ACWP. A negative cost variance indicates that we are in an over budget condition (i.e., the actual cost of the work performed (ACWP) is more than what was budgeted for that work (BCWP)). The schedule variance = BCWP – BCWS. A negative schedule variance indicates that we are behind schedule (i.e., the budgeted cost of work scheduled (BCWS) is more than the budgeted cost (amount) of work we have actually performed (BCWP). **CSQE Body of Knowledge Area: IV.B.3**

5. **Answer B is correct.** Reliability is a function of consistency. Cyclomatic complexity is a reliable metric because the Cyclomatic complexity of a module can be measured by different people (or the same person multiple times) with the same result. A metric is valid if it measures what we expect it to measure. Cyclomatic complexity is a measure of the control flow complexity of a module and not its size and therefore is NOT a valid measure of the size of a module. **CSQE Body of Knowledge Area: V.A.1**

6. **Answer C is correct.** Dynamic analysis involves the actual execution of the software product and evaluating the actual results against the expected results. Testing is a form of dynamic analysis. Running a spell checker, compiling a source code module and performing a peer review are all static analysis verification and validation techniques. **CSQE Body of Knowledge Area: VI.A.1**

7. **Answer D is correct.** Tracking the content and status of each build is a configuration status accounting function. **CSQE Body of Knowledge Area: VII.B.1**

Back to top

Medical Device Software

New IEC Guidance on the Application of ISO 14971 to Medical Device Software

By David Walker

Most medical device R&D organizations engineering software intensive devices are currently piecing together ISO 14971 “Medical devices — Application of risk management to medical devices” and AAMI (Association for the Advancement of Medical Instrumentation) TIR 32 “Medical device software risk management” to ensure an effective and FDA compliant safety risk management strategy for software.
The AAMI Medical Device Software Committee recently reviewed a committee draft of a new IEC TR 80002 (Medical device software – Guidance on the application of ISO 14971 to medical device software) that provides guidance on the application of ISO 14971 to medical device software. This report is effectively a globalization of the AAMI TIR 32 that established consensus in the US on software risk management philosophy and strategy for medical devices. This short article will summarize the contents and key points of this new guidance that is expected to reach global consensus in 2009.

Details:
IEC TR 80002
Title: Medical device software – Guidance on the application of ISO 14971 to medical device software
Ballot Committee: IEC/SC 62A, Common Aspects of Electrical Equipment Used in Medical Practice
Committee Author: IEC/SC 62A/JWG 03, Joint IEC/SC 62A-ISO/TC 210 WG: Medical device software (ISO/TC 210/JWG 02)
Tag: AAMI/SW, Medical Device Software Committee
The Committee Draft (CD) was prepared by the IEC Sub Committee 62A Joint Working Group (JWG) 3 between IEC Sub Committee 62A and ISO Technical Committee 210.

Firstly, the document structure of IEC 80002 matches that of ISO 14971. This makes it very easy for those familiar with ISO 14971 to find things. Even the section numbers and subsection titles match. An overall perspective would be that this technical report clarifies the application to software for each section of ISO 14971. One might also say the this technical report takes the information provided in AAMI TIR 32 and folds it into the outline of ISO 14971. The draft is currently 66 pages long and contains much detail with regard to software aspects of safety risk management.

In section 3, General requirements for risk management, warns against segregation of software risk management from overall system or device risk management. Often, there are opportunities to mitigate system safety risks with software safety features, and to mitigate software risks with system safety features. The involvement of software engineering personnel in hazard analysis and overall risk management process is critical to software intensive medical device safety.
Other points within this section include the iterative nature of software risk management, the difficulty in estimating software risk probability, proactive safety management, and some great detail in software engineering aspects of safety.

In section 4, Risk analysis, various methods are discussed for risk assessment such as Fault Tree Analysis (FTA, see IEC 61025), Failure Mode Effects Analysis (FMEA, see IEC 60812), and Hazard and Operability Study (HAZOP, see IEC 61882) and the value of

such approaches in various situations. It is stressed that due to the difficulty in estimating probability for software failure, the probability should be set to 1, and the rigor in mitigating the risk should be commensurate with the severity of the hazard.
Some work has started at the SEI (Software Engineering Institute) to establish structure for “Goal Structured Assurance Cases” as a method of software assurance. A goal-structured assurance case specifies a claim regarding a property of interest, evidence that supports that claim, and a detailed argument explaining how the evidence supports the claim. This is ground breaking work to provide a replacement for reliability studies in building confidence in software safety. Follow this link to section 5 of the report: http://www.sei.cmu.edu/pub/documents/08.reports/08tr025.pdf

In section 5, Risk Evaluation, early consideration of risk exposure is stressed to provide adequate mitigation. Hazards are traced to software components so that action can be taken to mitigate the risk of software components contributing to hazards.

In section 6, Risk control, the concept of last point of control, is discussed. Where a hazardous condition could result in a chain of events, it is often effective to consider risk control measures for the last event. This increases the chances of trapping loosely coupled causes which have unpredictable effects and may have been missed in risk analysis.
Risk management considerations for Software Of Unknown Provenance (SOUP) is also covered.

Section 7, Evaluation of overall residual risk acceptability, and 8, Risk management report, do not have much elaboration for software.

In section 9, Production and post-production information, the maintenance cycle is discussed. Considerations are made for updates to developed software or SOUP and risks associated with these changes.

There are four annexes:
Annex A (informative) Discussion of Definitions
Annex B (informative) Direct causes example
Annex C (Informative) LOOSELY COUPLED CAUSE/RISK CONTROL MEASURE
An excellent source for developing coding and design standards
Annex D (Informative) POTENTIAL PITFALLS
Annex E (Informative) Life cycle/RISK MANAGEMENT grid

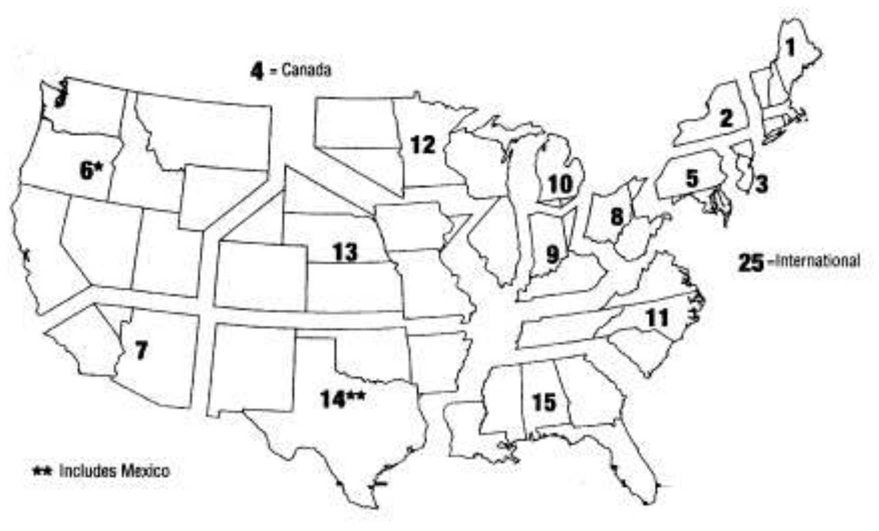
The ASQ Software Division newsletter will announce when the new IEC 80002 technical report is released and available for purchase. Watch for the spring edition of this newsletter in which a summary will be provided for another important IEC standard currently undergoing domestic review by the AAMI Medical Device Software Standards Committee, **IEC 80001 Application of risk management for IT-networks incorporating medical devices.**

David Walker is the immediate past chair of the ASQ Software Division. He represents ASQ’s interests in medical device software standards development through membership on the AAMI Medical Device Software Standards Committee.

Back to top

Back to top

FROM THE REGIONS



The following links will provide you with a snapshot of the latest activities in the regions.

Region 4 (Canada): Nguyen Hien

Having just joined the Regional Council since October 15, I am really looking forward to working with all the members in the local organizations. I am hoping 2009 will be a great year for us to work together to promote sharing software knowledge and best practices across Region #4.

Being a new kid on the block, I hope to get to know the software local chapters better in the next quarter. I welcome your input, comments on how we can work better together to achieve the above goal, especially your participation in the ASQ software division activities. Please feel free to contact me at mqhien@rogers.com.
My best wishes to you all for a festive Christmas holiday.

National
For the software members at large, do visit the Software Division website to access the discussion groups, and for updated information on relevant association meetings, conferences and other events in Canada: http://www.asq.org/software/.
If you have information that you would like to share with fellow **ASQ Software Division** members, or questions that you would like to have an opinion on, you can participate in the SW discussion board, you can post them on the Software discussion board: http://www.asq.org/discussionBoards/forum.jspa?forumID=11

Western Canada
The **Software Quality Assurance Vancouver User Group (VanQ)** held the session "Capturing Wild Requirements for Testing" in October, presented by Trevor Atkins. In November, Robert Goatham presented a model of a project to evaluate the principle of "Agile development". The schedule of events for 2009 will be available from their website: www.vanq.org.

The Calgary-based **IEEE/ASQ Discussion Group for Software Quality** meets the third Tuesday of each month during the lunch hour. In November, Paul Rogers presented "You can't automate that, can you?" December session focused on "Pitfalls and Perils of Data Testing in a warehousing Environment", presented by Mike Heinrich.
The 2009 January to April events is on various aspects of testing. The details are posted on the website (www.software-quality.ab.ca).

Eastern Canada
The **Toronto Association of Systems and Software Quality** (TASSQ) has posted its 2009 January and February 2009 events on TASSQ website: www.tassq.org. The January session will offer tips to testers and software engineers through "I am a bug" presentation by Rob Sabourin. The February session will focus on R& D tax credit, by Jason Schwandt.

The **Toronto Software Process Improvement Network** (SPIN) has been quite active in Q4.
In September, 2008 TSPIN had two presentations on estimation "Estimating software –intensive projects" presented by Emmanuel Gonnet, and "Estimating Your Way To Success " by Carolyn Swadron.
In October**,** TSPIN hosted a networking event. The November theme was measurements with again two presentations on "Measurement Reports" by Erika Vintan, and "We have data, now what?" by Norocel Popa. Full details can be found at www.torontospin.com.

The **Ottawa Software Quality Association (OSQA)** had a meeting in November. Matt Brayley-Berger from Borland presented "Agile Testing: Practical Approaches to Quality that Get Results", which was well attended with a lot of lively discussions.
OSQA will host a session in January 2009 on "SAP Automating Testing". More information on the January event will be posted on OSQA website: www.osqa.org.

The **Montreal Software Process Improvement Network** (SPIN) Web page is under construction. Hopefully, more information will be forthcoming in January of 2009 on its website: http://www.spin-montreal.org/.

Back to top

Region 6 Pacific Northwest – Tom Gilchrist

Version…1Q2009-1

ASQ Software Division 6 Report
February 2009
By Tom Gilchrist

If you are in the Seattle area on the third Thursday of every month (except December), The Seattle Area Software Quality Assurance Group (SASQAG) holds monthly public meetings in the Seattle area. SASQAG also supports certification and study groups. If you are in the area and want to attend, please look at www.sasqag.org for upcoming events, directions, and meeting time.

The Americas Aerospace Quality Group (AAQG) is meeting in Portland March 10-12. The AAQG Project #60 will convene at this meeting to work on draft AS9115, the new international software quality program standard. AS9115 expands on the scope of the Americas standard AS9006 and contains more robust guidance on airborne complex and simple hardware devices, more explicit guidance on configuration management and is tuned to harmonize with the newly released version of AS9100. Questions can be directed to the project lead, Mike Kress, michael.p.kress@boeing.com.

Also in October, the annual Pacific Northwest Software Quality Conference will be held (http://www.pnsqc.org/) in Portland, Oregon. In April, they will be calling for papers and presentations for the conference. If you have a paper or tutorial you'd like to give this fall, get an outline started now!

The Seattle SPIN (Software Process Improvement Network) is holding meetings on the first Tuesday, five months a year.. The organization is driven with a single, clear-cut goal in mind: change an organization in a way that improves that organization's ability to develop software. If you are interested in more information on SeaSpin, you can go to http://www.seaspin.org

If you have information on local software quality and testing events in your area of Region 6, please send them to me for our events calendar   Visit http://www.tomgtomg.com/asq6 for information on events around Region 6.

Tom Gilchrist, Region 6 ASQ Software Division
tomg@tomgtomg.com

Back to top

Region 8 Ohio Valley – Greg Zimmerman

Region 8 - Greg Zimmerman

The ASQ Pittsburgh section is offering certification prep classes via webinar. Visit http://www.asqpgh.org/ed.html for more information.

Anyone interested in learning more about the Software Engineering Institute at Carnegie Mellon University (SEI) should visit their website - http://www.sei.cmu.edu.

The ASQ Columbus section's summer/fall course schedule is about halfway done, but there are several courses not starting until October, including the CSQE refresher. See the schedule under the Education link at http://www.asq-columbus.org/.

The Central Ohio Quality Assurance Association (COQAA) is the most active software quality-specific group in the Columbus area. COQAA is a federation chapter of the Quality Assurance Institute (QAI). They meet on the third Thursday of September, November, January, March, and May. Visit their site - http://www.coqaa.org/ - for locations and topics.

ASQ Cleveland is looking for a volunteer to take on the Program Chair responsibilities of recruiting and scheduling speakers. If interested, please contact Kurt Costantino (user600818@aol.com).

Please feel free to contact me with any information for Region 8 members, requests for software quality related referrals, or if you would like to learn more about getting involved with the Software Division. You can reach me at: gregz@appliedqualitysolutions.com.

Back to top

Region 10 Michigan, Northeast Indiana & Northwest Ohio – Louise Tamres

Region 10: Louise Tamres
1Q2009

The 3rd Great Lakes Software Excellence Conference was held this past November in Grand Rapids, Michigan. To find out about the r
conference (dates, call for papers), visit the conference web site periodically at glsec.org.

The Great Lakes SPIN meets the second Thursday of the month. Locations vary between Oakland University in Rochester, University
Michigan-Dearborn, and Schoolcraft College in Livonia. The GL-SPIN frequently sponsors CMMI and SPICE training programs. Information about programs and events is available at gl-spin.org.

The Southeastern Michigan Software Quality Assurance Association (SEMISQAA) is a local chapter of the Quality Assurance Institute
(QAI). The group meets monthly on the second Tuesdays, and it also sponsors QAI training sessions. Event locations vary throughout
year. More information at semisqaa.org.

Ann Arbor Software Quality Professionals (AASQP) has suspended its monthly meetings due to low participation. The mailing list rema
active as a source of communication and information for software quality professionals in southeastern Michigan. Access is through th
Yahoo group tech.groups.yahoo.com/group/aa-sqp .

ASQ chapters in southeastern Michigan provide programs primarily related to manufacturing. Any programs focusing on software and software quality will definitely be highlighted when available.

I am surveying a sample of software quality professionals in the greater-Detroit area to gauge whether enough interest exists to provid CSQE training. In addition, some local software quality professionals have volunteered to offer a set of training courses in southeaster Michigan. If either of these educational opportunities interest you (whether as an attendee or as an instructor), do let me know at l(dot)tamres(at)computer(dot)org .

Back to top

Region 14 Southwest & Mexico – David Peercy

ASQ Software Division Region 14 Report for 1Q2008
by David Peercy

This column provides members with information on relevant association meetings, conferences and other events in Region 14 – Texas, New Mexico, Oklahoma, Arkansas and to some extent Mexico. If you have information that you would like to share with fellow **ASQ Software Division (ASD)** members, or you would like to get involved with the Division, contact: depeerc@sandia.gov or 505-844-7965. As always – if you or your section have something you'd like to include in this short quarterly report, please let me know. If you have a website that I can promote – please let me know. The 2QFY09 information is due by end-April 2009. Please check the ASQ site below for much information on multi-region activities and contacts.

http://asqgroups.asq.org/Divisions/Soft/Newsletter/

TOPICS FOR DISCUSSION

It has been awhile since this report was written, and there are several topics of interest.

(1) Region 14 Split Into Region 14A and Region 14B

a. Regional Directors for Split Regions Announced
Effective January 1, 2009, regions 6, 11, 14, and 15 will each split, and will be designated as A and B.

SAC Chair David B. Levy has appointed Regional Directors to complete the terms for the B regions. Regional Directors for the 8 regions are as follows: 6A: Tim Koester; 6B: Neal Kuhn; 11A: Diane Byrd; 11B: Clay Hodges; 14A: Belinda Chavez; 14B: John Breckline; 15A: Joni Judd; and 15B: Gene King.

b. Region 14A & 14B Sections:

Region 14A		Region 14B	
1400	Albuquerque NM	1402	Dallas TX
1401	El Paso TX	1407	Central Arkansas (Little Rock)
1403	Mexico City MX	1408	Oklahoma City OK
1404	San Antonio TX	1409	Tulsa OK
1405	Greater Houston TX	1413	Ozark (Fayetteville AR)
1406	Central Texas (Temple)	1415	Northeast Arkansas (Jonesboro)
1410	Shreveport-Bossier LA	1416	Fort Worth TX
1412	West Texas (Amarillo)	1419	Arko (Ft. Smith, AR)
1414	Austin TX	1421	South Arkansas (Camden)
1418	Brazosport/Freeport TX	1424	Texoma (Sherman TX)
1420	Beaumont TX	1426	North Central Texas (Stephensville)
1422	Bay Area Texas (Clear Lake)	1428	North Central Arkansas (Batesville)
1423	Southwest Louisiana		
1425	Lower Rio Grande (McAllen TX)		
1429	Juarez MX		

c. For now, I will be the Region 14A and 14B Software Regional Councilor, but in the near future we could use a replacement for the new Region 14B – any takers?

(2) Conference Planning and Significant Actions

a. WCSQ/Institute for Software Excellence 5/18-20/2009, Minneapolis, Minnesota

   i. http://wcqi.asq.org/

b. **International Conference on Software Quality 2009 (ICSQ)**
Theme: Controlling Software Before Software Controls You!
Conference Dates: November 10-11, 2009 (with pre-conference tutorials on November 9)
Conference Location: Northbrook (Chicago), Illinois

Technical papers and panels should be practitioner-oriented. They may be based on research of interest to practitioners or on experiences that relate to any aspect of software quality. Tutorial/Workshop Sessions will also be presented. These are either half-day or full-day sessions that provide practical knowledge to participants. Workshops that promote the active participation of learners through problem solving, case studies, or other interactive learning methods are encouraged.

Important dates:
March 1, 2009: Abstract Submission Deadline
March 20, 2009: Presenters Notified of Acceptance
April 1, 2009: Technical Program Available Online

(3) Regional Information: Past and Upcoming (websites explain this – here are a few)

a. **ASQ Dallas Section 1402: See** *www.**asqdallas**.org/*

b. **ASQ Austin Section 1414: See** *www.**asqaustin**.org/*

c. **Fort Worth ASQ Section 1416: See** *www.**asqfortworth**.org/*

d. **Albuquerque ASQ Section 1400: See** *www.**asq**1400-abqnm.org*

(4) Some Thoughts and Discussion Points

a. Year ago – WCSQ Software Quality Principles

b. Near Future – I'll provide an executive summary approach to what I call a better understanding of "Quality" – including "Software Quality" as a sub-topic. Hopefully I'll be able to get this approach into the ASD Newsletter and initiate more discussion.

(5) Region 14 Membership Question

a. Our Region 14 software division membership was 280 as of April 2007. What do you think it is now? I'll report on the membership numbers in the next Region 14 report – but – just to see if anyone is really interested – how about a guess or two? Is it lower or higher? Give me a number?

(6) Next Region 14 Report – an executive summary on what I think "Quality" means

a. I've heard many discussions on just what quality is – from "You know it when you see it", to "Meeting customer requirements" to "Meeting customer expectations" to… plus descriptions of the characteristics and principles and factors that determine quality. I'd like to venture in a slightly different direction – yet a direction that I believe integrates all these concepts – in a conceptually simple way. ASD leaders Bill Trest and David Walker have both been interested in my thoughts on this – and, given a little time, I'd like to share those thoughts next time. Stay tuned.

Back to top