DEDICATED TO SHOWCASING NEW TECHNOLOGY AND WORLD LEADERS IN SOFTWARE TESTING

LogiGear MAGAZINE

THE EXPLORATORY TESTING ISSUE

FEATURE
Beware of the Lotus Eaters: Exploratory Testing
By Anne-Marie Charrett

RELATED
Are Testers' Ethnographic Researchers?
By John Stevenson

SPOTLIGHT INTERVIEW
Jonathan Kohl on Exploratory Testing

February 2011 | Volume V | Issue 2

EDITOR'S LETTER

EDITORIAL STAFF

Worldwide Offices

United States
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +01 650 572 1400
Fax +01 650 572 2822

Viet Nam
1A Phan Xich Long, Ward 2
Phu Nhuan District
Ho Chi Minh City
Tel +84 8 3995 4072
Fax +84 8 3995 4076

www.logigear.com
www.logigearmagazine.com
www.logigear.com/blog

Submission guidelines are located at http://www.logigear.com/logigear-magazine/submission-guidelines.html

For everyone still celebrating holidays: Happy Lunar New Year! At this time of the year many teams and companies are starting new projects, new initiatives, and hiring new staff. *LogiGear Magazine* will continue to be the resource for better testing with much less stress!

We are excited about the focus of this month's issue: Exploratory Testing. This is very timely with the rise of agile testing methodologies and the building of new testing strategies to satisfy the agile need for speed**.** Whatever the software development method: agile, waterfall or one of the many methods in-between, great exploratory testing skills and effective communication of the exploratory effort are essential to successful testing.

Anne-Marie Charrett's feature "Beware of the Lotus Eaters" contains an insightful discussion on exploratory testing and ideas about being *done***.** John Stevenson provides a unique perspective on how testers' can benefit from ethnography followed by the "Spotlight Interview" with Jonathan Kohl.

This issue launches our new layout featuring departments such as "In Brief" and "Announcements" where we showcase LogiGear events including our new training course information on exploratory testing in the US and in Poland, conference appearance in StarEast as well as this month's "Blogger of the Month."

Enjoy our special feature on Vietnam's celebration of the lunar new year also known as Tết. With articles, videos and training course information on exploratory testing, I am sure we have an issue to be referenced and passed around to test teams of all skill levels!

Gear up for our next issue with the theme focusing on the hot topic on everyone's mind: *How can we automate testing in agile development?*

Michael Hackett
Senior Vice President
Editor In Chief

IN THIS ISSUE

IN BRIEF

STAREAST 2011

Speaker: Hans Buwalda, CTO LogiGear

Topic: Full-day Tutorial: Introducing Keyword-driven Test Automation

Date: Tuesday, 3 May 2011

Time: 08:30– 16:30

Keyword-driven test automation has entered the software testing mainstream. It has proven to be a powerful approach to reach a high level of automation with the lowest possible effort. It brings the flexibility, manageability, and maintainability that cost-effective software test automation demands. Hans Buwalda introduces keyword-driven test automation and demonstrates how to make it successful.

The basis for the tutorial are his "Five Percent Challenges of Test Automation" in which he explains that no more than five percent of test cases should be executed manually and no more than five percent of total testing effort should be used to achieve this level of automation. In addition to the technical aspects of keyword automation, Hans will address the many non-technical ingredients required for success: an automation-friendly test design, managing processes, and organizing teams. Hans will also focus on topics including agile test development, testing technical software and testing graphics and multi-media empowering a tester to make keyword-driven testing work efficiently.

Not yet registered? Visit www.sqe.com/stareast. Don't miss the chance to participate in training and networking sessions! ■

Blogger of the Month

Karen N. Johnson began as a technical writer in 1985 and later switched to software testing in 1992. She maintains a blog at *TestingReflections*, a collaborative site where she is featured as a main contributor. In her latest entry, she discusses search testing with different languages. Here is an excerpt from her blog:

"I started work on a new project and needed to address search testing with a wide assortment of languages. And geez, this is a puzzle I've worked with before so I thought I would share some thoughts around the topic of search testing with multiple languages.

At the start, I looked into how many languages and what languages I'll be working with. Based on past (and current) experiences, I have certain reactions - from a testing perspective - to some languages.

Latin-based languages fall into one group. I used to think of English, Spanish and French as "different" languages but now I see these languages are rather similar from a test perspective. The same is true for any languages that use the Latin-based character set. . . .

. . . So when it was time to choose a handful of languages to test with, my reaction was to choose:

- one or more Latin-based languages
- one or more languages with a heavy use of diacriticals
- an RTL language
- a language that is more symbolic than character-based

A common problem in testing with these languages is the lack of keyboard or a means of entering characters from different languages. Cut and paste can work if you're careful. . . ."

To read more from Karen N. Johnson, visit her at www.testingreflections.com. ■

ANNOUNCEMENTS

EASTERN EUROPEAN EXPLORATORY COURSES

Victo Academy

Location: Poland

Dates: Warsaw 4-5 April/ Wroclaw 7-9 April

Warsaw & Wroclaw, Poland, April 2011 – Michael Hackett, co-founder and Senior Vice President of LogiGear, together with VictO Academy announce the first Exploratory Testing classes in Poland. The two-day classes will take place in Warsaw 4-5 April and in Wroclaw 6-7 April.

Exploratory Testing is an approach to software testing that is concisely described as simultaneous learning, test design and test execution. The term was coined in 1999 by Cem Kaner, LogiGear founder Hung Q. Nguyen and Jack Falk in their book *Testing Computer Software*. The publication launched the widely known style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the quality of his/her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run parallel throughout the project. The main advantage of exploratory testing is that less preparation is needed, important bugs are found quickly and at execution time, the approach tends to be more intellectually stimulating than execution of scripted tests.

Established in 1994, LogiGear University, the leader in training software test engineers, has developed a unique curriculum based on decades of Silicon Valley experience with trainings delivered in 15 countries. This two-day course in exploratory testing will teach test engineers how to become more productive using the method's benefits and how to avoid its pitfalls.

For more information about the classes, please visit VictO Academy: http://victo.eu/Exploratory/index.html

NEW EXPLORATORY COURSE BY LOGIGEAR UNIVERSITY

Exploratory Testing

This two-day course is designed to give test engineers a global understanding of exploratory testing from "why we do it" and its uses to "how we do it" and the value of measurement. Exploratory Testing will be examined and practiced to empower test engineers in using this method in finding better bugs earlier, focusing on customer satisfaction and making exploratory testing more manageable and easier to use as a necessary and important test method.

Visit http://www.logigear.com/course-catalog/993-exploratory-testing.html.

LOGIGEAR CHANNEL

Visit www.logigearmagazine.com to view the following videos:

Exploratory Testing with Jonathan Kohl

Introduction to Skilled Exploratory with Cem Kaner

VIET NAM SCOPE

Year of the Cat 2011

Viet Nam celebrates its lunar new year emphasizing on family, good health and a prosperous beginning

Lolita Guevarra

February third welcomes Tết Nguyên Đán, or otherwise known as Tết, welcoming the year of the cat. Tết holiday is an event of family, food and a favorable new year. The holiday varies from late January to early February officially lasting three days. More recently, China and Vietnam celebrate Tết on the same day, yet an hour apart, after years of debate regarding the designated local standard time for the lunar calendar.

Similar to western traditions, people begin preparations for the global event weeks prior in excited anticipation with warm greetings to one another and businesses are closed for an extended holiday to begin the year with rejuvenated spirits. As our Ho Chi Minh City branch celebrates the holiday, they shall indulge in the excitement of their city cloaked in the color red symbolizing happiness, luckiness and advantages. The deep hue spreads throughout the country with families in high hopes of a new lunar year's prosperous possibilities.

In an attempt to attract future wealth, prices in general rise as families laboriously clean their homes adorned with cherry blossoms, gold dragons and red envelopes filled with money. Belief holds that no sweeping is allowed in the days of Tết as it will "sweep" away the good luck that befalls the family.

For those with year-end debts, a conscience effort to clear them prior to the holiday is essential to bring one's self a clean slate for good fortune.

Those who celebrate Tết are more than familiar with the supernatural and spiritual aspects of this celebration. It is believed on the 23rd day of the twelfth month of the lunar calendar, the three kitchen gods report to the Jade Emperor of a family's events that occurred in the past year. Food is prepared with respect to the gods and their journey to the emperor. Along with paid respects to the gods, families travel to pagodas where incense is lit in memory of deceased family members.

A family altar plays a pivotal rode during the holiday as fruits and food are placed for ancestors who have passed on. Past and present members gather together in unity and respect with elaborate meals featuring Tết favorite bánh chưng, a packaged dish of tightly packed sticky rice mixed with meat or beans wrapped in banana leaves.

Side dishes prepared mainly for this holiday include the dưa hành of picked onions and cabbage and mứt, a variety of dried sweetened fruits. As this is the most important and popular holiday for the Vietnamese, many take their annual leave to spend as much time as they can with their families highlighting a mass exodus from major cities to local hometowns in the countryside.

With food and decorations thoroughly enjoyed, red envelopes are given to the squealing delights of the children. Fed and filled, children are left to play amongst themselves while the adults engage in the events of the past and those of the future. ■

FEATURE ARTICLE

BEWARE OF THE

The Greek myth inspires Anne-Marie Charrett to warn testers of premature euphoric apathy in exploratory testing

Like many professions, the lure of the "low hanging fruit" in software testing is very appealing. In this case, fruit refers to finding high impact bugs quickly.

Exploratory Testing (ET) combined with Risk based Testing is a useful strategy for finding these bugs. Of course ET can deliver a whole lot more than that, but the opportunity to demonstrate an immediate impact is often a compelling reason to use Exploratory Testing.

Testers are not the only ones who like this fruit; it is also welcomed by the rest of the team, in particular project managers. Some bugs found early in the testing phase are, in general, welcomed by most team players.

Consequently, exploratory testers often tailor strategies to find and deliver these types of bugs. This means cheap and easy tests are run first. There is nothing wrong with this. It's a good and effective strategy. There's nothing like a few serious bugs at the start of testing to lull a manager out of a false sense of security.

It becomes a problem though if it becomes the only model used to perform testing. Testers eat the lotus flower of early success and are seduced and satiated with its bewitching nectar and stop testing. Any inner doubts are squashed with arguments such as "there is no such thing as 100% coverage" or "exhaustive testing is expensive and impractical, so why try?"

This is especially true when faced with complex and hard to understand software. It can be a big problem, because

Exploratory Testing is tester centric—making the tester in control of how much testing to perform. In general, there are few team members who will object to the decision to stop testing and jump ship.

So, testing is declared as "Done Enough." To have "Done Enough" testing is the poor man's cousin to "Good Enough" Testing. "Done Enough" testing stops when the tester has had enough. Some bugs have been found and any more testing either requires imagination, technical expertise or some hard work on the tester's part. "Done Enough" Testing is bad, folks. It reflects poorly on test teams and does little to educate the wider community on the benefits of Exploratory Testing.

Exploratory testers (well, any tester for that matter) need to be aware of the lotus flower and continue to test. And while testers may not be able to perform exhaustive testing, it shouldn't preclude extensive testing.

Extensive testing goes beyond early success. Testers don't dwell on the satisfaction of finding bugs early. They test beyond that. Project managers might be reasonably content (yes, they may have been seduced by the lotus flower, too) but testers need to wake up and keep testing until testing is "Good Enough."

It's more expensive to continue testing and it is harder to achieve at a reasonable price. This is where automation or tools come in. A good toolsmith can change where tools used effectively will reduce the cost of testing to allow more testing to take place. This results in a better standard of "Good Enough."

I had a firsthand experience of the importance of tools in exploratory testing.

Eusebiu Blindu, a Czech tester, recently came up with an interesting online challenge. He created a testing exercise in finding bugs and patterns. The application drew a polygon with the number of sides determined by a number you entered into the field. You can see it here:

LOTUS EATERS

http://www.testalways.com/2010/07/05/find-bugs-and-patterns/

There was no indication of the limit of numbers this field could take. I decided to test using a variety of numbers as testing every number seemed beyond my capabilities. I took the following numbers: 0 ~ 10, then I jumped to 15, tried a few random numbers to 99, 100, 999 and then 1000 and finally I threw in a couple of negative numbers to see what would happen. I noticed some relationships in the colors, a few irregularities, but beyond that nothing else stood out.

> *"Extensive Testing goes beyond early success. Testers don't dwell on the satisfaction of finding bugs early. They test beyond that."*

So what did I do? Did I read books to find out more? Did I seek outside knowledge, ask someone, or question. Did I try and change my model? Did I examine my assumptions?

No, instead I limited my testing. I put the puzzle aside. I decided I had "Done Enough"

Until I saw the following tweet:

@jamesmarcusbach "A good exercise for tools. RT @charrett: Eusebiu sets a challenge. Anyone care to answer the call? http://ow.ly/27gmu #softwaretesting "

I was surprised; I didn't think that this puzzle merited a tool. It was a polygon puzzle with only one input. What was the benefit from automating one field? Anyhow, how could you automate the testing of something like this?

I contacted James Bach on Skype and asked him what he meant by the tweet. We had a coaching session on the topic. Incidentally, James and I do a lot of IM coaching. In fact, we're in the process of writing a book on the subject.

James explained to me he had tested 1200 inputs. I was impressed; regardless of the time it took to enter so many numbers, how could he remember the differences between each of the polygons?

It turned out his solution was to write a Perl script to automatically enter in a number, then take a snapshot of the result and save it in a directory. This was then followed by some deft blink testing. I followed his approach and very quickly, patterns and bugs became apparent.

I realised how limiting my approach to testing was in this scenario. When faced with a problem beyond my immediate capability, instead of figuring out how to fix the problem, I narrowed my model.

I allowed myself to be seduced by the deadly combination of early success and fear of complexity.

I reduced the scope because of the complexity of inputting large inputs. Rather than solve the problem of entering large amounts of inputs, I decided to test less. This was a major flaw in my testing strategy.

The lesson I learned was not to be limited by what seems to be impossible or hard. If it's not possible to achieve (or if it's too tedious to achieve) through manual means, then find another way of doing it. In this case automate it!

Testers need to be able to suspend belief of the impossible and put aside a skewed view of what can and cannot be done. They need to shine torches not only where others don't want to look but also where we fear to tread.

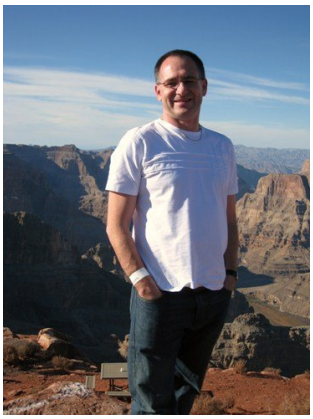
Beware the lotus eaters! ■

Anne-Marie Charrett is a test consultant and runs her own company Testing Times. An electronic engineer by trade, software testing chose her, when in 1990 she started conformance testing against European standards. She was hooked and has been testing since then. Anne-Marie blogs at http://mavericktester.com and her twitter id is @charrett . Her consulting website is http://testingtimes.com.au

RELATED ARTICLE

ARE TESTERS' ETHNO

Using ethnography, John Stevenson argues there is much testers can learn from the sociological field



People who follow me on twitter or via my blog might be aware that I have a wide range of interests in areas outside my normal testing job. I like to research and learn different things, especially psychology and see if it may benefit and improve my skills and approaches during my normal testing job. One area I have being looking at for awhile is the social science of ethnography. The approaches used when carrying out research appears to have many similarities to software testing and I feel we could benefit and maybe improve our testing skills by examining ethnography.

In my opinion, there are two areas in which we can learn from ethnography:

- To improve our understanding of users and how they differ by using ethnographic methods
- Use ethnographic methods to test software in an exploratory way.

I should start by explaining what my understanding of ethnography is according to anthropologist Brian Hoey. He defines ethnography as "equated with virtually any qualitative research project…where the intent is to provide a detailed, in-depth description of everyday life and practice." He continues to state that an ethnographer is more than documenting events and details of everyday life but that the ethnographer attempts to explain how such occurrences represents the cultural construction an individual or community lives in.

The problem with trying to describe and define ethnography is that it has wide and varied meanings. To me, it is a branch of the study of humanity (anthropology) in which the researcher actively gets involved and participates with the study group rather than just sitting back and observing. The reporting is using qualitative (words) measurements rather than rely on quantitative (numbers) measurements.

One of the key factors when approaching ethnographic research is to be aware that participation, rather than just observation, is one of the keys to the approach. Does this not sound familiar to testing, especially exploratory testing? Actively using the software under test to find out about its characteristics and behavior are similar to an ethnographic researcher living within a community and participating with that community to learn about its beliefs and characteristics. There appears to be very close parallels between ethnographic research and exploratory testing. Wikipedia states that "one of the most common methods for collecting data in an ethnographic study is direct, first-hand observation of daily participation."

How similar is that to testing software?

Another approach within ethnography is the use of grounded theory to explain the results from the participation. This is when the data is used to provide theories about the data. This is different from grand theory in which the theory is defined without the use of real life examples and therefore has a danger of not fitting the actual data gathered afterwards. Is this similar to scripted and exploratory, grand theory vs. grounded theory?

Grounded theory is a constantly evolving set of conclusions that can continue indefinitely based upon the changing data being obtained by the ethnographic researcher. One of the questions that are asked about ethnographic research is: When does this process end?

One answer is: never! Clearly, the process described above could continue indefinitely. Grounded theory doesn't have a clearly demarcated point for ending a study. Essentially, the project ends when the researcher decides to quit. (http://www.socialresearchmethods.net/kb/qualapp.php)

GRAPHIC RESEARCHERS?

How similar is this to testing? When do we stop testing? Many articles have been written on this subject and mainly we stop when we can learn nothing new, no time or ran out of money. See Michael Bolton's article where he lists twelve heuristics for test efficiency methods. I feel that ethnographic research stops because of similar reasons.

One interesting section I saw within the Wikipedia article on ethnography was about the process of ethnographic research in which to aid the researcher areas were split and the research asked questions:

- Substantive Contribution: "Does the piece contribute to our understanding of social-life?"
- Aesthetic Merit: "Does this piece succeed aesthetically?"
- Reflexivity: "How did the author come to write this text…Is there adequate self-awareness and self-exposure for the reader to make judgements about the point of view?"
- Impact: "Does this affect me? Emotionally? Intellectually?" Does it move me?
- Expresses a Reality: "Does it seem 'true' - a credible account of a cultural, social, individual or communal sense of the 'real'?"

I thought about this and started to change the context to be about software testing:

- Substantive Contribution: "Does the testing carried out contribute to our understanding of the software?"
- Aesthetic Merit: "Does the software succeed aesthetically?" Is it suitable for the end user?
- Reflexivity: "How did the author come to write this test…Is there adequate self-awareness and self-exposure for the reader to make judgements about the point of view?"
- Impact: "Does this affect me? Emotionally? Intellectually?" Does it move me?
- Expresses a Reality: "Does it seem 'true' - a credible account of a requirement'?"

By doing this I found I suddenly had a set of heuristics to measure against the software testing that has been carried out, yet again more similarities between the two crafts.

Another area in which ethnographic research can be useful to software testing is when you need to test software that has a lot of UI interactions. Using the methods of ethnography a tester could go visit the users and observe and participate in their daily routine to find out the common tasks carried out and what oddities are seen.

The oddities are the things of greatest interest since these are the things that would not normally be planned for and without active participation with the users would not normally be uncovered until it is too late.

There are many studies being carried out to determine if ethnographic research should be used when designing software system. However, my concern with this is that it appears to be stuck in the design up front way of working which is not a flexible iterative approach. In my view it is easier, quicker and cheaper to ensure that testers use ethnographic methods when testing to ensure the design is suitable for users or even better get the users involved earlier and observe them earlier.

The more I have delved into the study of ethnography the more and more I have seen similar patterns to software testing. This makes me aware that software testing is not solely a hard science but a craft that encompasses many disciplines outside of the typical number crunching and algorithm creating world of software development. Within the testing profession we need to look outside of the box and find approaches, methods, structures that can improve the discipline. To ensure our craft grows we need to ensure we do not narrow out field of vision or thought.■

To read original post please visit http://steveo1967.blogspot.com/2011/01/are-testers-ethnographic-researchers.html

Tweet John Stevenson: @steveo1967

Hoey, Brian. "What is Ethnography?" Brian A. Hooey, Ph.D. 18 January 2011. 24 January 2011 http://www.brianhoey.com/General%20Site/general_defn-ethnography.html

Bolton, Michael. "Blog: When Do We Stop A Test?" DevelopSense. 18 January 2011. 24 January 2011 http://www.developsense.com/blog/2009/09/when-do-we-stop-test/"Ethnography." Wikipedia. 2011. Wikimedia Foundation, Inc. . 24 January 2011. http://en.wikipedia.org/wiki/Ethnography

SPOTLIGHT INTERVIEW

Jonathan Kohl: Kohl

Jonathan Kohl discusses his mentors, exploratory testing methods and the diverse backgrounds of his testers

1. Can you tell us a little bit about your take on Exploratory Testing (ET)? What do you do differently than James Bach, for example?

I'm influenced by Cem Kaner and James Bach, and I have used a lot of their work as a base to build from. In my work, you'll often hear me cite their prior work, but one of the interesting things with an exploratory approach is that it is dependent on the individual. That makes it hard to compare between one or the other tester talking about ET. Sure, the ideas are similar, and you'll hear common themes from many of us, but the individual experiences and knowledge collected over time is very different.

Rather than try to compare, I will tell you what I am focusing on in my work. I'm passionate about bringing ET to the masses, by making it approachable:

- I actively work with teams to help them apply ET or complement their scripted testing with it
- I write articles addressing questions that come up the most in my public work, often providing alternate explanations for an ET approach
- I try to explain concepts in ways that people can understand and apply in their own work
- I share experiences, tell stories and provide live testing demonstrations
- I explore areas where scripted testing and ET can work together
- I explore how I can use automation tools within my exploratory test work, as described in my *Man and Machine* article
- I hate it when a tester feels stupid. I want ET to help empower them and help them feel energized and excited about their work
- I try to make my ET training as real-world and approachable as possible, and I constantly work at improving it

I've done some work as a commentator: there are a lot of approaches to ET out there. Some of them are well thought out, thorough and cite prior work. Others, not so much. However, it is fascinating to see how there are different ideas and approaches, and personalities with different ideas.

I have done some work categorizing different approaches and strategies that I see. You have people like James Bach who is constantly coming up with different ideas and approaches and is often pushing the envelope. You have that with others now too, particularly the *Thoughts from the Test Eye* blog authors. Rikard Edgren, Henrik Emilsson, Martin Jansson are a great recent example. I call a lot of these ideas "abstract strategies."

They spur testing thinkers on to get better and better and expand knowledge. At the other end of the spectrum, you have that unknown tester who just discovered how to apply a new idea to their testing, and pragmatically records it on their

> *"My manager at the time had me read* Testing Computer Software *by Kaner, Falk and Nguyen and suddenly I had words to describe what we were doing:* exploratory testing. *Cool! Now I had a term and respected people in the testing community to back up my approach."*

"cheatsheet" wiki page. "Copy this data and paste it in the app in various places" or something along those lines. That's a very *concrete* idea or strateg, and different personality types reach for different strategies at different times. There are others - I introduced some of these ideas here: http://www.kohl.ca/blog/archives/000188.html

I'm also slowly trying to develop productivity tools to help aid ET.

Concepts Explored

Why should the programmers get the benefits from cool technology advances while we still use stone-age tools? It's taking more time than I would like, but I hope I'm at least contributing ideas in that space.

I've tried to bridge the gap between the theory that we, who talk about ET often like to indulge in and the testers on the front-lines with their fingers on the keyboards. That has been surprising. For example, I have been working with others on a free open source tool to help record and structure session-based testing called Session Tester. I started out with a simple design for capturing session notes, and then I looked at the *Session-Based Testing Management* article by Jon and James Bach.

I'd tried out SBTM as described in the article several times in the past, but I had adapted it for different reasons. Some of it was too heavyweight for an Agile team, or some testers got hung up on the word "charter" and so on. I always kept the basics of SBTM sessions: a clear goal or charter, a time box, a reviewable result, a debrief with others, and the use of heuristics and other thinking tools to help with test idea generation during test sessions.

However, I tracked different things and recorded different things with different teams. When we started going through the design, we hit a crossroads: do we follow the SBTM document to the letter, or do we start basic and get feedback from real users on how they have implemented SBTM? We decided to go with the latter and let the community dictate what they wanted in the tool.

The results were fascinating. Once I got feedback on a half-baked tool pre-release, I found that there were many variations of SBTM out there. I had very few people chastise me for not supporting a "by the book" implementation of SBTM in the tool. Instead, I got a lot of encouragement to make the tool as flexible as possible to allow for variations.

I think this is great - people taking a tool, trying it, and adapting it to suit them. We have so much restrictive attitudes in software development: "do this, do that and only until you are deemed a master should you try to adapt" and it is refreshing to see what people are doing to adapt ET tools on their own without permission from a guru. Wonderful!

I find that space fascinating. There can be an amazing contrast between what people like me are talking about versus what is actually going on in the industry inside shops that we don't have visibility into. I have to admit I don't always like what I see, but if it is working for that team, then that's really all that matters. They don't have to enjoy ET at all the levels I do.

2. How do you run your sessions in ET? Are they 120 minutes long or do you have a different approach?

I follow the basics of SBTM sessions: a clear goal or charter, a time box, a reviewable result and most importantly, a debrief. How that looks, what we report on, or what terms we use depends on the team, and how I can get the concept working with that group of individuals. Often, just getting people comfortable with note-taking is the first hurdle. I've often replaced the term "charter" with "goal" or more often "mission." Like "heuristic" the word "charter" can invoke a negative reaction in some people. To me it's just a word, so let's get past the word and get on to the concept and make it work.

The time box varies in length. I like to start people out with an hour with an appropriate setup and cool down period afterwards and work up from there. It is amazing how exhausting doing full concentrated testing work with note taking for an uninterrupted period of time can be. Once people get used to that, we ramp up or down depending on energy levels and other factors. I rarely go over 120 minutes in my own work, but I have had testing sessions in pairs or trios that went for much longer because we were experiencing flow, we were discovering all sorts of interesting informa-

SPOTLIGHT INTERVIEW . . . CONTINUED

Jonathan Kohl: Kohl

tion and we fed off each other's energy.

3. How do you use heuristics in your ET process?

When I first started leading test projects, I found that my team was overwhelmed with paperwork stemming from test plans and test case management tools. I adjusted to try to meet the needs of various stakeholders in the team, and I thought: "our mission as testers is to provide important information, like bug reports to stakeholders. Hardly anyone reads our test plans, and no one else looks at our test cases. What are we in business to do? Find and report information quickly or maintain and create heaps of paper?"

I and the testers I worked with decided we should focus on testing and do as little documentation as required. We adapted quite naturally and organically, but we broke some of the rules in the QA department. People liked our results, but they were uncomfortable with how we got there. My manager at the time had me read *Testing Computer Software* by Kaner, Falk and Nguyen and suddenly I had words to describe what we were doing: exploratory testing. Cool! Now I had a term and respected people in the testing community to back up my approach. That was powerful.

As I moved on and began to lead another team, I had trouble training them. They didn't have the benefit of the experience and gradual adjustments my other team had. I needed to get them up to speed quickly, so I started creating these "test philosophy" documents, and how to guides, bug taxonomies and so on to accompany our test checklists.

I was used to using whiteboards as communication devices, so I got the biggest one I could find and stuck it up outside my office to help guide and track our testing work. This was all quite crude and rudimentary, but it worked. I started digging into Cem's work to see what other options were out there, and from Cem I discovered James Bach's work, and suddenly I had a new word to describe what we were doing: heuristic.

I was quite familiar with the term from both my undergrad work in logic and philosophy and from having my Dad as a teacher in elementary school. He was big into using heuristics to solve word problems in mathematics. When I read James' work on heuristics, I started using them to help us plan and structure our testing. It wasn't until I met James in person and we tested software together that I saw a heuristic being used in real time. A big lightbulb went off over my head.

I can use these much more in real time as well as planning and everything in-between. Wow! How did I miss that? I know about heuristics and they worked great with those word problems in real time. In fact I was already doing much of that, but using James' work as a springboard really kick-started me. Why reinvent the wheel when there is such great prior work that you can draw on?

So I use heuristics for several purposes. One is to help plan and structure our testing, and how to take a seemingly endless world of possibilities, and consciously optimize that work into something manageable and most likely to reveal important information. I also use them in real-time and I have a few of my own I like to pull out in different situations.

> *"How ludicrous is it to just specify one result when there are lots of things we can use? How can I possibly pass this test case by looking at one tiny aspect? The work Cem and James described on oracles really resonated with me."*

Heuristics are a memory aid and a possible way to solve a problem, so they help me in all sorts of situations. Anywhere from hearing a trigger when interviewing a team member about a design such as "persistence" - my heuristic is the question: "what gets persisted where?" to using something like San Francisco Depot in a live testing demonstration or during my own planning work.

Concepts Explored

4. Do you use Oracles? If yes, what was your experience in using the program?

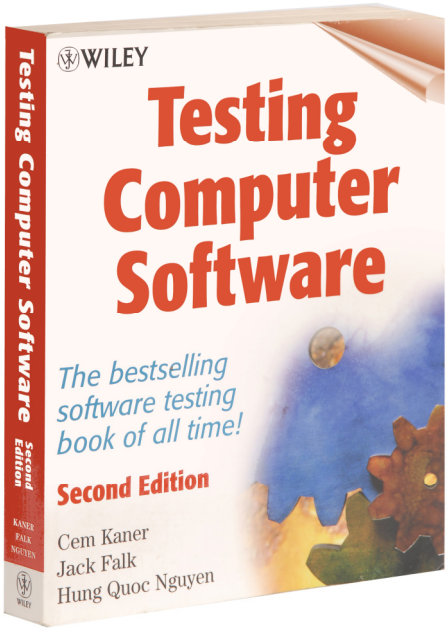
Of course, that's another word I got from James and Cem. It used to drive me crazy when I'd see a scripted test case with one expected result specified. My brain would immediately leap to different options and combinations. How ludicrous is it to just specify one result when there are lots of things we can use? How can I possibly pass this test case by looking at one tiny aspect? The work Cem and James described on oracles really resonated with me. Thankfully I wasn't alone.

The oracle problem - finding out sources of information that let you know if your tests are passing or not, can be subtly difficult. My rule of thumb is to get a second source of information to verify things by. Sometimes that's relatively simple: "There is a free government service that provides the currency exchange rates once a day that we can compare our calculations with."

Sometimes it is complex, like working with data visualization techniques when doing performance testing. Why on earth are we seeing a spike in page load times every 20 minutes? That sort of oracle determination requires a combination of experience, knowledge and a willingness to research.

Some things that have surprised me:

- Validating data sent from embedded devices in a test environment - discovering that the real thing worked far differently than a new design and tests let on. The (passing) tests were using mocked up data, and I decided to use the real thing to contrast with the data used in the tests. An engineer unlocked a storage space, pulled out some dusty devices and helped me get them going. I needed them to generate real data, so we played with a temperature probe, just to get some sort of data generated. To generate the first batch of real data, I grabbed a soldering iron and put a can of Coke in the freezer. Once the soda pop was sufficiently cold, I alternated touching the hot soldering iron and the cold pop can on the probe. Real data was generated, immediately uncovering real problems discovered in the system.
- A tester's gut feel, or intuition - faced with overwhelming evidence to the contrary, including passing tests, massive project inertia, great testers seem to be able to stand up and say: "Wait a minute. Something *feels* wrong. Give me a couple of hours to investigate." Those are always surprising because they start with a suspicion or hunch, and investigation and a good investigative structure around their testing leads to real evidence.



Countless examples of the wrong oracle in a test case document. Either the document was out of date, or just plain wrong to begin with. Using simple means to come up with an independent check on that oracle revealed the test case was just

Jonathan Kohl

wrong. Even a little creativity within test cases or using them as exploratory guidance references can reveal interesting information.

5. How do you find the best testers with the best domain knowledge? What are some of the ways you go about finding this information out?

I find out if someone has a testing aptitude by asking about their interests, and their experiences in the past. I worked with a test manager who had worked for many years in a restaurant, and I was able to map some of the lessons they had learned there to their success as a tester. When you personalize and validate prior experience that the individual tester may not associate with technology, interesting things happen.

I've had sports coaches, history researchers, accountants, short-order cooks, programmers, database administrators, systems administrators, and all sorts of roles excel in testing. Two of the best who come to mind were an executive assistant who was brought in to help test a new system part-time, and a technical support analyst. These were both people with superb analytical, reporting and investigative skills, and you'd never have known it without giving them real testing problems to solve.

6. Lastly, can you tell us about a recent tour that you've completed or done, and what you found in the way of bugs or problems using ET that you would not have found using traditional testing methods?

I feel that way about most of the past 13 years of doing exploratory testing. It's hard to pick just one. Often, this occurs when there are intermittent bugs that people are struggling to reproduce. Hold on - one is coming to mind now ...
Recently, I worked with a team that was transitioning from a highly clerical scripted testing approach to an exploratory testing approach.

They were in a regulated environment, so still required documentation. We had a transition period where we were using both systems - the emerging lightweight documentation system for ET, and the older scripted system. The test lead asked me to help get some coverage on some of the test cases that hadn't been converted over to checklists and guidance docs yet, so I happily obliged. It's fun to do something a little different once in a while. One of the test cases felt wrong. There wasn't a lot of information beyond: "go here, put in these inputs and these outputs should show up" and a note to use some tool to validate the results. I installed the tool, looked over the test case and while it was passing I was still confused. I tracked down the test case author and they were a bit taken aback.

The test didn't make much sense to them either. So I moved on and talked to a programmer, the Product Owner and a subject matter expert. They all felt that the test was out of date and not valid anymore. I had a hunch that there was something to it. Why write the test case in the first place? Also, I can't just mark a test as passed or failed without understanding it, so I decided to spend a few minutes exploring.

Now for the embarrassingly simple part of this story: I had no idea what this feature did, or what its purpose was, so I started off on a simple tour. What were all the functional aspects of this feature? I moved from area to area on the screen clicking, typing, moving the mouse around, but I still felt lost. So, in desperation, I used a "click frenzy" heuristic. The mouse started slowing down when I was in a certain region of the screen, so I honed in on that. All of a sudden: Bang! The application crashed. I went back to the programmer: "Is there any reason why clicking here over and over would cause a crash?"

That suddenly jogged his memory about that feature, and all these details from months ago rushed back. Now he knew what the test was about, and we brought in the Product Owner and Subject Matter Expert and started sharing knowledge. Of course, the feature had nothing to do with a click frenzy + crash, but that was enough of a primer to get the ideas and memories going. The scary part of all of this?

The test had been reported as passing for months, but the functionality had been broken all that time. That's what happens when testers get bored of following the same test scripts over and over - they start to game the system instead of using their powerful brains. ■

LOGIGEAR MAGAZINE
FEBRUARY 2011 | VOL V | ISSUE 2

United States
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +1 650 572 1400
Fax +1 650 572 2822

Vietnam
1A Phan Xich Long, Ward 2
Phu Nhuan District
Ho Chi Minh City, Viet Nam
Tel +84 8 3995 4072
Fax +84 8 3995 4076