The difference between a site going live and a site going dead can be hard to see. Even with the most sophisticated testing software, sometimes what you need most is an expert set of eyes.

Web Performance engineers have years of experience discovering the kinds of performance issues that can cause even the most sophisticated sites to crash under the smallest loads. We’ll test your entire infrastructure, looking for critical bottlenecks and configuration issues. Then you’ll receive a detailed report showing the fixes you’ll need to make so your site can go live—and stay that way.

For more information on our expert service plans and how we can improve your site’s performance dramatically, visit webperformance.com/services.
CONTENTS

10 Another Day in the Life of a Tester
   Bernie Berger, The unexpected, crazy, multi-tasking and critical thinking day

15 Ask the Tester
   Matt Heusser, Interview with James Bach

20 The Role of QA in the Brave New World of Agile Development
   Howard Deiner, How QA fits into an Agile environment

22 A Discussion About Bug Regression
   Mohammad Alam, The importance of bug regression and its outcome

26 How Children Learn to Test
   Matthew Heusser, Take your software testing learning cues from how children learn

30 Delivering Value & the Testing Quagmire
   Paul Fratellone, The business case

34 Exploratory Testing as a Creative Process
   Adam Geras, Testing with no limits

36 Performance Requirements:
   An Attempt at a Systematic View
   Alex Podelko, An examination of performance requirements

Contributors

@SoftwareTestPro    SoftwareTestProfessionals

STQA Magazine (ISSN #1548-3460) (USPS #23771) is published bi-monthly by Redwood Collaborative Media (724 S. Tejon Street, Suite C, Colorado Springs, CO 80903). STQA Magazine is a registered trademark of Redwood Collaborative Media. All contents copyrighted 2010 Redwood Collaborative Media. All rights reserved. To contact STQA Magazine subscriber services, please send an e-mail to info@softwaretestpro.com or call 719.476.0301.
As I reviewed the articles for this edition of ST&QA magazine, the article *Delivering Value & The Testing Quagmire* by Paul Fratellone struck me. I started thinking about delivering value and the impact on the organization as well as the difficulties associated with communicating value. There are many ways to deliver value but it often seems we overlook and under communicate the value we bring to the final products our organizations deliver. Paul’s article delivers insightful ways to help communicate value.

In Paul Fratellone’s article he also discusses how to create a value-add testing organization and as Paul clearly points out, value is defined by many audiences and based on many different criteria. It is important that we each understand our individual value to the team as well as the team’s value to the final product.

Bernie Berger is back with his second installment of *A Day In The Life of a Tester* and as you read through his day it is easy to see why we sometimes under estimate our value. One of the big take aways from Bernie’s series is the fact that he looks back on the day and highlights the things that did add value. Sometimes they are little things and sometimes the things we do bring significant value but we should never believe what we do is a waste of time.

It can be a lonely world being a tester and the feature article written by Michael Larsen aptly points out when you are THE test “team” it is all on you. Michael states, “You are the last line of information to the product team as to what identified dangers exist inside of the code. Sobering is a good word for this situation.” In circumstances when you are the only tester your value will be scrutinized very closely and there is nowhere to hide and no one to hide behind. He shares five ways to be a more effective lone tester.

And finally on the subject of value, Matt Heusser interviews James Bach in the regular *Ask the Tester* article. I met James in person at the March STP conference and it was very apparent that James has a mission to add as much value as one person can to the testing profession through his dedication to mentoring new, up and coming, and seasoned testers. If you want to improve your value as a tester, James is willing to help in that process. There is a lot of valuable advice for testers in this feature.

We have four more great articles in this edition. One of the goals of ST&QA is to highlight and feature the practitioners that are making it happen in the testing profession and are willing to share their experiences with you – the reader. We have been receiving articles from many new authors and it has been a pleasure making them available to you for consumption.

The STP Conference Program Board has been meeting to develop another valuable program at our fall conference in Dallas, Texas. I can’t rave enough about the quality and dedication of the people on that board! It will take place October 24-27. I hope you will save the date.

We are all valuable when we stand up and share our knowledge with others. I encourage each and every one of you to remember that what you do has a valuable impact on the organizations you work to improve. Those who are engaged at STP by submitting articles, participating on the website, newsletters, and boards are reaching tens of thousands of dedicated professionals. They add value to the membership experience. If you are interested in being involved in the association, contact me directly at rhand@softwaretestpro.com.

If your day sometimes feels wasted remember to look more closely because we have a tendency to overlook the little things that in the end can mean a lot to our careers and our organizations. This edition of ST&QA will help you find that value.
There is no question that there are great opportunities for improvement when it comes to testing teams and organizations that employ them. A discussion less frequently addressed is when individual testers are the entire test team.

Various names have been applied to this: the lone tester, the army of one, the omega tester, etc. Regardless of the name, the situation is the same. We are the sole resource for a specific project, or for a development team, and often for an entire company. As more companies move from traditional to Agile development practices (or start with Agile practices from the beginning), the idea of a dedicated test team is fading in favor of self-organizing teams that share a number of disciplines, testing being one of many needed skills. As a lone tester, my efforts take on an interesting hue. It's all on me. I really am the last line of information to the product team as to what identified dangers exist inside of the code. Sobering is a good word for this situation.

There are certainly benefits to being a lone tester. It's a very liberating environment. We often get the freedom to test in the manner that we find the most effective. Decisions around tools and processes are often tailored to our own needs and niche abilities. Our efforts are often seen as the last line of defense for the product team. We also run the risk of becoming silos if we are not careful. Knowledge that we develop is more likely to be locked to us unless we take the initiative to share the knowledge we have. Also, other testers cannot cover our deficiencies. Holes in knowledge and experience will have to be filled in by active work and learning, and needs for knowledge and experience may change over a period of months or, in some cases, days.

So how can we thrive in this environment where we may well be the only active voice for testing?

1 Focus on Communication

Communication can be difficult, because we are the only voice speaking for the bugs in the system. That's not to say that developers do not care about bugs, they certainly do, but they are often focused on their own features or their specific release. All bugs are not created equal, and the perception of which bugs are important and which are less important are addressed every day. The lone tester must be brave enough to speak up and advocate for the bugs that are not having enough light shined on them. The responsibility to stand up and speak for those bugs belongs to us.
2 Learn the Expectations of Your Development Team

When a team relies on a lone tester, there has to be a give and take as to what can be accomplished in any given time frame. Lone testers do not fail because they are by themselves; they often fail because they have made an unrealistic expectation as to what they can actually do. Assess your abilities, your strengths and your weaknesses and present them honestly. Realize that it is OK to not be an expert in everything (nobody is). State clearly areas where you perform well and areas where you will need to improve, and stay vocal about it. Many of your team mates will be happy to help you learn what you need to learn if they can; your success helps them succeed as well.

4 Learn How to Work with the Development Tools of Your Organization

It has become part of testing folklore that development and test are adversaries and that we are somehow doing battle with or providing protection from development’s mistakes. It makes for cute stories, but it’s not a very effective method for working with a development team. When you are working with a development group, you are effectively a consultant providing a service to that team. Work and behave accordingly. Be honest in your evaluation of issues, but make sure your focus is on fixing issues and getting a well-tested and solid product ready to be released.

3 Stop Thinking in Terms of Us vs. Them

Regardless of the name, the lone tester, the army of one, the omega tester, etc., the situation is the same. We are the sole resource for a specific project, or for a development team, and often for an entire company.”

Michael LARSEN
5 Plan for Pairing Sessions with Developers and Domain Experts

Pair programming is used in extreme programming and Agile. Pair testing is also an effective approach to evaluating software. While it would be great to have two testers working together, a tester and a developer also make for a great test team. Additionally, pair testing with customer support representatives, marketing team members, sales, and even executives can be very helpful. Each group in an organization has a different perspective, and each group has its own level of expertise. Pairing with the experts in my organization has given me many insights I might not think about were I to focus on testing by myself. Some of the best user experience questions and insights have come from regularly pair testing with customer support representatives; they know the customer pain points better than anyone.

6 Cultivate Relationships with Mentors

Lone testers have additional challenges when they are looking for mentorship; who can they turn to? For me, if I cannot find someone in my own company to mentor me, I look outside of the company. The testing community has many dedicated people who are happy to share their knowledge and experience with other testers. I suggest lone testers look for other lone testers to discuss ideas, get input and feedback on ideas, and work together to help build understanding and have opportunities to grow. Social media outlets like Twitter, Facebook and LinkedIn also give testers the opportunity to find and follow each other’s discussions and add their own comments, which in turn can be considered and challenged by others.

7 Develop Your Craft

Coursework like the Association for Software Testing’s Black Box Software Testing series can help testers learn new strategies and ideas for testing approaches and ideas, as well as to reach out and meet/communicate with other testers. Conferences, meet-ups, code camps and discussion forums can help introduce individual testers to other testers looking to share ideas and methods. Also, I’d be remiss if I didn’t mention Weekend Testing, an initiative I am actively involved with. Weekend Testing is a regular gathering of software testers to take on a challenge and learn about testing in a safe and fun environment.

Sessions typically take place on weekends (hence the name) but these sessions can be held any time. The key is to have a focus, and agenda, and a limited time frame to focus on testing and discuss the ideas learned. All of these are great ways to help sharpen the saw and hone our testing craft.

To my fellow lone testers, you are critical in the success of your organization’s endeavors. You have the potential to become indispensable. Many of the challenges you will face will be universal, and there will, of course, be challenges that will be unique to your organization. There is no one size fits all scenario for the lone testers out there. Although with the willingness to adapt and be open to your abilities, as well as the areas we need to improve on, we need not be solitary figures that live in the shadows. We can be seen as an essential element towards project success.

About The Author

Michael Larsen is a lone tester with SideReel.com in San Francisco, CA. He is a brown belt in the Miagi-do School of Software, an instructor with the Association for Software Testing, facilitator for Weekend Testing Americas and the producer of Software Test Professionals “This Week in Software Testing” podcast. He can be found on Twitter at @mkltesthead and blogs at http://mkl-testhead.blogspot.com.
Two Flexible Ways to Advance Your Education

Our flexible training options are designed to give software test and quality assurance professionals the skills necessary to successfully transform their complex business challenges into strategic capabilities.

Attend training in the classroom or online and take your career on the path to higher levels of success.

Visit us online to find our complete training calendar, browse by course, location and delivery format.

<table>
<thead>
<tr>
<th>Dates</th>
<th>Course</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/11/2011 – 7/13/2011</td>
<td>Agile Boot Camp</td>
<td>St. Louis, MO</td>
</tr>
<tr>
<td>8/10/2011 – 8/12/2011</td>
<td>Web Core Testing Fundamentals (ONLINE)</td>
<td>Virtual Classroom</td>
</tr>
<tr>
<td>8/15/2011 – 8/17/2011</td>
<td>Agile Boot Camp</td>
<td>Columbus, OH</td>
</tr>
<tr>
<td>8/18/2011 – 8/19/2011</td>
<td>Agile and High Speed Testing Techniques</td>
<td>Detroit, MI</td>
</tr>
</tbody>
</table>

*Class locations and dates are subject to change.*
I roll over and look at the clock. 10AM. 10AM?! I was supposed to be in the office an hour ago. What happened? I overslept? That’s so not like me.

I check my cell phone – battery is dead, no wonder, the alarm didn’t wake me up. And I didn’t hear any of the missed calls that I’m sure came in from work asking where I was.

I scramble to my laptop and frantically connect to work. Gee whiz, I completely blew off the 9:15 meeting. What poor form. I quickly log on, and send an email out to the team to let them know I’m delayed and will be in the office just as soon as I can. I quickly get ready and run out of the house.

On the train ride in, I’m thinking about what has happened. Yesterday seems so long ago. I was up late last night, online, reading my favorite blogs and catching up on Twitter. I must have lost track of time and I finally hit the sack at around 2AM. But yesterday at work, yeah, that was a good day with a high-priority project. We found lots of issues, even though it wasn’t exactly according to plan. We’ll just have to catch up today. Coming in late today is not going to be one of my shining moments.
of information about the release that ultimately is making it a stronger product in the long run. But despite all that, I think there is still enough time for us to run our planned tests before the deadline, assuming there are no further crises."

“OK good. As long as we make the deadline. Just get it done.”

I head over to the test lab to see Raj.

“Hey man, what happened to you, are you all right? Did you get stuck in all that traffic with everyone?” he asks.

“Yeah, everything’s fine. Thanks. Let’s just figure out what we’re going to be doing today. We left yesterday and the automated smoke test finally kicked off, after all those set-up problems, right?”

“Yes, it took all day to set up the environment correctly, but now we are good to go.”

Between us, we plan out our strategy for the day.

“Let’s bring up the system and get the environment set up. We’ll kick off the full set of automated smoke tests. And let’s sit together and go through these new features,” I say.

11:30

Raj and I are sitting next to each other in front of a desktop in the lab. Raj is at the keyboard, I’m next to him. I have my notes from yesterday’s meeting with Eric, the programmer.

We’re going to have a session of paired exploratory testing. Raj and I are a good team. We feed off each other when testing in a very productive way. Between us, we have a name for that feeling when we’re in the zone, when our creative energies bounce off each other creating a positive synergy. We call it being “on the jazz”. It’s a catchphrase from the TV show “The A-Team”, but it really fits. What usually happens is that I would suggest a test, and Raj tries it, then Raj notices something which triggers an idea that he tries. I’m right with him and then I suggest a third thing based off of Raj’s idea. And this cycle of trial and error continues to bounce between us until we find results that we think would be very interesting to report. We’ve done this plenty of times before. Working together, we “talk out” the tests that we want to do, which helps us discover new ideas or different variations of tests to try. Some people call it testing with “another set of eyes” but it’s much more than that. It’s the synergy that’s created between us that becomes really valuable.

We bring up the entry window, and, remembering the duplicate entries that we saw yesterday, I tell Raj to just quickly spot-check the dropdowns, just to make sure that our database is still OK. We are happy to see that it is.

There’s a new checkbox on the entry screen that the user is supposed to select to flag the entry as an “MMY entry.” “MMY” has a particular meaning to our users, but from the system perspective, it is simply a checkbox that triggers a flag in the database. And at the end of every day they need to generate a report of all the MMY entries for that day and submit it to the regulators. That’s why the project manager is so eager to get this in production already, because Compliance is down his neck for the industry-wide MMY deadline next week. Before, all entries with certain criteria were automatically considered MMY, but now the user can designate any entry no matter what criteria, as MMY, by clicking the MMY checkbox before submitting.

Simple tests seem to work fine. Clicking, unclicking, submitting. What about correcting an entry that’s already been entered? That seems to work fine too.

I tell Raj that things are looking OK but I think we’re thinking about this testing too narrowly. We’re only considering tests from the perspective of the checkbox; if it’s checked or not. We need to think out of the box.

“You mean out of the checkbox.” Raj says with a smile.

“Oh, ouch! That was horrendous,” I chuckle. “But do you know what I mean? Who cares if the checkbox gets checked or not. It’s just a stupid GUI object. I mean, no offense to the checkbox designer or anything, but do you really expect to find a bug where you click on it and it doesn’t get checked? We need to think about what it means, how it affects the downstream applications, how it would affect the different stakeholders, not only the users. Like the production support guys, for example. How would this affect them?”

“Well they are in charge of making sure that things go smoothly for everyone and if there’s a hiccup they need to smooth it out,” he says.

“OK, well, then let’s simulate a test scenario for them. Let’s see what would happen if we do all our regular checkbox tests while the system is being acted upon in a way that mimics what the production support guys would deal with on a typical day.”

“So you mean, let’s have a few workstations up. On one we’ll continuously insert entries, and on the other we’ll delete or edit them. And on the third, we’ll insert these special MMY entries.”

“Yeah, and how about this: in the middle of all these transactions, we’ll have a fourth workstation where we’ll play with the user permissions, granting and removing user entitlements for entering and deleting. Remember yesterday Eric said we didn’t have to test permissions? Let’s do it anyway.
That would be a really good simulation of real life conditions.”

“Well, not really,” interjects Raj. “What I mean to say is that it is good, but it could be better. What would be really the best is if we could get a copy of the production entry logs and replay them.”

“Ahh! Perfect! You just touched on one of the classic debates in test design – whether it’s better to test with customer data, like the production logs, or constructed data, like the specific cases we wanted to try. There are pros and cons to both sides and like everything else it depends on your context. I can tell you about it later, but for now how about we do both. I’ll go ask Maria for a copy of the production entry logs and I’ll modify the Python script I was working on last week to pump those entries in on the server side here in the QA lab. While I’m doing that, you can put together a quick GUI automation script for the continuous entry of MMY entries for the second workstation. While these are running, together we’ll manually play around with the user permissions, and we’ll see what happens.”

“Sounds like a plan.” says Raj. “But how about we get back together after lunch? I have to go to the Post Office.”

“Lunch? What time is it? 12:30 already! Wow, it feels like I just got in.”

“You kind of did,” Raj says with a smile.

So Raj goes off to the Post Office and I go back to my desk to call Maria, the DBA. I dial her extension.

12:30

“Hey Maria, how’s it going?

...Yeah, it’s busy here too. Don’t ask. How was your dentist appointment yesterday?

...I know, it’s really hard to find someone you trust. Good that you took care of it, though.

...Anyway, listen, I was wondering if you could do something for me. Can I get a copy of one day’s worth of entries?

...Yeah, from Prod.

...Doesn’t have to be. It could be from whenever. We just want to pump them into QA for a test.

...What?

...Yeah, last week is fine, it doesn’t really matter.

...Yeah, I know, I know, you have to desensitize it. We wouldn’t want us QA folks taking a look at our customer’s sensitive data, now, would we?

...Hahaha, I know, tell me about it.

...Wait, but how long does it usually take to encrypt the client sensitive data?

...Thirty minutes? That’s awesome.

...Yeah, cool.

...Oh, one more thing, is it CSV or plain text?

...No, either is fine.

...OK cool that works.

...Where? On the same shared drive as last time? Yeah that would be perfect.

...Yeah I have access to that drive; I’ll just grab it from there.

...Thanks a million, Maria, this is a great help, really.

...OK, sure, thanks.

...Yup.

...Bye.”

1:00

I’m not really hungry yet, but I better go get something from the cafeteria before they close. So I go down to get a sandwich. At the elevators I see Vlad and Eric. They seem to be arguing about something, as usual.

“Hey guys”

“Hey, what’s up”

“Not much. We were setting up a test for the MMY release.” I tell them about our test strategy.

“Sounds good. Let me know if you need any help with that.” says Eric.

“Thanks.”

1:30

Come back to the lab, Raj is there already. So we sit back down at the computer. This time, I’m driving at the keyboard and Raj is next to me.

“I updated the Python script and I have the data file from Maria”, I say. “So now I just have to set it up here, and we’ll be good to go. How did you do with the GUI automation script?”

“It’s all done. It’s really just a simple script, nothing fancy, and it doesn’t do any validation. All it does is click on the submit button once every 5 seconds, so it would send in a new entry with the same criteria continuously. You have to manually set up the entry screen with the criteria, and then kick off the script to keep hitting submit.”

“Wow, I love how crude that is. Quick and to the point. But sometimes crude is all you need.”
Why don’t you kick yours off first and we’ll just check that it would be OK to continue testing.”

“Sorry? I didn’t get you. What do you mean by ‘OK to continue testing’? Let’s set up the whole test like we said before, I’ll run my script while you run your script, and at the same time we’ll modify the permissions.”

I smile. “Raj, you just highlighted another debate in the world of testing! When testing, is it better to start with the most complex case first, or, is it better to start with the simple case and build up to the complex case?”

“I don’t know, I guess there are pros and cons to both sides here too. It depends.”

“And what does it depend on?”

“I don’t know...your context?”

“Exactly! Like you said, there are pros and cons here too. When the first thing you do is start with the simple case, you are establishing that everything is OK to proceed with testing. Each next test adds a little more complexity to the testing so that if you find an issue, it’s easier to determine which level of complexity introduced the problem. But the downside is that this sometimes takes a lot of time to get to that point, and the simple cases are not likely to cause any problems anyway. So sometimes you might want to start testing right away with the complex scenarios.”

“OK fine, so let’s start with my script.”

“It’s your choice,” I smile.

Raj sets up his test, enters the criteria in the entry window, but doesn’t select the MMY checkbox, just to check the simple case before building up to the more complex case of MMY. He kicks off the script that clicks ‘Submit’.

“Hey, something is wrong. It’s submitting the entries, but I don’t see them coming through. That’s weird.”

“Really? That IS weird. I promise I didn’t set you up for that. But why wouldn’t these entries go through? It just worked this morning. What happened?”

“That’s what we have to find out. Let me check the logs.”

Raj goes ahead to try to figure out what’s going on in the lab, while I go back to my desk to see if anyone else might have an idea. I’m going to call Eric and Vlad but first I’ll sort through my unread emails, just in case something urgent came in while I was in the lab.

Nothing too important. A reminder that I’m supposed to give a presentation at the company’s monthly QA Center of Excellence meeting tomorrow. Oh geez, I completely forgot about that. I’m not sure I’ll be able to do it anyway; this project is really taking up most of our time. I don’t even know what I’m going to talk about.

I walk over to Eric’s desk to see if he knows anything about the entries that are mysteriously not showing up.

“No, man, I have no idea. Maybe Vlad knows.”

Just then the project manager comes running in, agitated, frantic.

“IS ANYBODY SENDING TEST ENTRIES TO PRODUCTION HERE?!?” he hollers.

“To production? Of course not!” I respond, shocked.

“Well, somebody is. We just got a call from production support that one of our customers got like 50 entries in a row with test criteria in them.”

My stomach drops. Those are our missing entries.

“In Prod!? How in the world did they get to prod? We entered those in QA.”

Vlad joins us. “What was that about prod?”

“We were doing a test in the lab,” I say, “just to enter a bunch of entries, and somehow they ended up in production. But this morning we also entered the records and it worked fine, in QA.”

“Uh, I was in the lab a little while ago, no one was there. I needed to compare my new code to what is currently in production so I changed the config to prod on one of the machines.”

“You what?”

“Yeah.”

“So that explains exactly what happened. While we were at lunch, you changed the config, and then when we came back the script we ran was running over the production lines. That’s crazy. How did you change the config to prod? You know the production password?”

The project manager jumps in: “OK it doesn’t matter. We’ll talk about this later. But now we know what happened. I have to go back to prod support and let them know, so they can tell the customers what to do.”

“Sorry about that”, Vlad says. “I’ll go change the config back to QA.”

“Yes, please!” I say.
Back in the lab, talking with Raj.

“Can you believe it? It was connected to Prod! It’s a good thing that we only submitted a few entries. That was bad enough. But could you imagine what would have happened if we would have submitted everything? If we would have replayed production data into production? That would have been a nightmare.”

“Yes, I agree. But my only question is how did we have access to production? Did we have production user permissions?”

“It’s a great question. I have no idea. I would have thought that it shouldn’t have been accepted,” I say. I stop and think for a moment.

“Say, did you have the MMY flag ON in those entries?”

“No, they were the first batch, with MMY off. Why?”

“Because I bet that MMY is not backwards compatible! The production database is not set up to accept the MMY flag. We’re all extremely lucky. Listen, if the MMY flag would have been on, I’m sure we would have crashed the production system, and bad. Remember yesterday we needed to run all those database scripts to get the environment in order for MMY? Well, those scripts weren’t run in production yet. The database, as it is currently, wouldn’t know what to do with that extra flag.”

“Wow, you’re right,” says Raj. “But I still don’t understand how the user permissions allowed us to send entries to prod.”

“Yeah, user permissions...” I think for a second.

“Hang on a second! User permissions! I wonder if the users need a special permission to add MMY orders. You know, each user can be enabled or disabled for making entries in general, but they also probably need an additional permission to enter the order as an MMY. I better check.”

The project manager is out in the hallway, he just finished giving Vlad a lecture about keeping the environments in order. I join them. Nobody looks happy.

“Hey, do users need special permissions to enter MMY entries?” I ask them.

“Of course,” says the project manager. “You know that everything is permissioned.”

“Because I don’t think we have that capability.”

Project manager turns to Vlad, “Well, is MMY permissioned?”

“MMY? No, I didn’t add MMY to the permissions database. No one told me it had to be permissioned.”

Today is not Vlad’s day.

The project manager looks too tired to be upset. “Well, we need it in there. Vlad, just do it right away, come on. And we need it for this week.” And turning to me, then he says, “you’re just going to have to step it up tomorrow.”

Yeah. Tomorrow.

Packing up my stuff to go. It was another insane but good day. We had a good initial exploratory test session this morning. Raj and I bounced ideas off each other all day. And we had different examples come up today about how there’s no one right answer to testing problems, like the customer vs. constructed data and the complex vs. simple tests to try first. Also, each of us decided to build an automated script to help us with our testing. What’s cool about that is that we decided on our own that it was the next important thing for us to do, and we made that decision while we were testing. And besides the drama of pumping entries into production, that showed us a big gap in our internal systems security. Plus, not to mention that in the course of discussion, we realized that an entire piece of functionality was missing from the build!

Yes, tomorrow is another day.
He’s worked at some of the most innovative companies in Silicon Valley, co-authored a best-selling test textbook, keynoted at every international test conference, and, yes, has primary credit for popularizing the term “exploratory testing.” Did I mention he’s a high school dropout who unschools his son? (For more on unschooling, see the article “How Children Learn (to Test) on page 26.)

Of course, I’m talking about James Bach, our invited interviewee for this, the exploratory test issue of STQA Magazine.

James Bach will now answer your questions.

**QUESTION** I’ve had a few people suggest to me that exploratory testing is just a ‘fancy name’ for ad-hoc testing. How would you suggest I respond?

Peter Walen, Grand Rapids, Michigan

**JAMES** Yes, and “dance” is another way of saying “boogie down.” So what? They both refer to rhythmic movement. Yet there is a difference in connotation.

Many years ago I promoted the idea of sophisticated, professional ad hoc testing. But I could not get any traction with that. Most people don’t read dictionaries, I guess, and they kept thinking that “ad hoc” meant sloppy. It doesn’t mean sloppy. It means something prepared for a particular situation. I soon grew tired of fighting that battle and decided to embrace Cem Kaner’s term for the same kind of testing. ET had no baggage, so it was easier to get past the silly stuff and communicate the essence of the ideas behind it.

Exploratory and ad hoc are two different concepts, but they both are descriptive of non-scripted testing.

**QUESTION** In the environments I’ve used it, ET has proven very useful. However, it has been suggested that large testing organizations, both large in number of testers and large in distribution of locations, are not suitable for Exploratory Testing because of the extreme difficulty in the ability of management to effectively control test efforts. Again, I am curious – how would you respond?

Peter Walen, Grand Rapids, Michigan

**JAMES** I hear that a lot from people who mistake a particular form of exploratory testing for the whole enchilada. Look, exploratory
testing means you think while you test, and that your thinking affects the design of your test as you go. Is there anyone who truly says that a large organization wants me to stop thinking? And if so, is it possible that whoever is saying that has already, um, stopped thinking before they started speaking? No organization that seeks to organize intellectual work can do that by telling its people to stop thinking. That’s just a silly idea on its face.

Imagine if someone claimed that allowing people to drive their own cars was unrealistic in a large city because traffic would get all snarled up. Well, obviously we do drive our own cars, and yes, traffic CAN get all snarled up. But the alternative requires a great deal of investment to obtain a result that is a great deal less flexible.

All testing is exploratory to some degree, or else it isn’t testing. The reason people get confused is that they think ET is a rejection of organization and structure, which of course it isn’t. We apply whatever structures we think will help us.

**QUESTION** Can you explain the focus and de-focus technique used in Exploratory Testing? I have read about it, but have not attended a session where it was demonstrated.

*Sherry Chupka, Pittsburgh, Pennsylvania*

**JAMES** Focusing and defocusing is not really a technique, in itself. Focus is an attribute of techniques. Any given technique of testing focuses you in some ways and may also defocus you. To focus means to work within the constraints of a specific pattern of testing (hence, a test script is a focusing method); to defocus is to change your pattern. Focusing also means to restrict your attention to a smaller area more meticulously, while defocusing means covering something larger but with less fidelity. Generally speaking, focusing is for studying something specific (such as investigating a bug), and defocusing is for searching for something new to study (like finding a bug).

All of testing can be seen as a process of strategic and tactical focusing and defocusing.

A great example of the dynamic is what we do when tuning an old fashioned analog radio. We spin the dial quickly, hearing static. Suddenly, as we speed by a radio station, we hear a moment of music. Then we slow down and go back, thus acquiring the station. Going fast was defocusing; going slow was focusing. If you tried focusing all the time you tested, you would find few bugs. But if you never focused, you would not be able to claim that you had done a thorough job of testing, since you would not be able to relate testing to any specific model of coverage or risk.

**QUESTION** If I may, I have a question for “The Buccaneer Scholar”: What are the subjects/thinkers outside of testing and computer science or engineering you think are crucial for developing as a software tester?

*Curtis Stuehrenberg, Seattle, Washington*

**JAMES** I suggest that you study the social sciences. Here are some of specific social science fields that I’ve been inspired and helped by. Google these:

- Grounded Theory
- Naturalistic Inquiry
- Situated Cognition
- Philosophy of Science
- General Systems Thinking

As for people, check out the work of Richard Feynman, the physicist, or Julian Baggini, who has written the most readable books on general philosophy I have seen.

I think the scientific and philosophical illiteracy of our field explains the strange longevity of stupid ideas that don’t work, and have never worked. Our industry seems to be run by fairy tale logic.

**QUESTION** If I do a fair amount of testing in an exploratory style and I am familiar with some of the basics – but what can I do to elevate myself from getting the job done to excellence? That is to say, I am interested in ways to increase the value I provide my team and customers.

*David Hoppe, Dorr, Michigan*

**JAMES** Here are some ideas for you:

- Can you do a stand-up test report on zero notice? What if I gave you five minutes to prepare one? Will your test report include all three levels (i.e. product status, testing activities, justification of testing activities)? Practice that.
- If I were to ask you what your test methodology is, could you give me a five-minute chalk talk that looked good and also was true? You should have a mental picture of testing in your mind, and some way, under pressure, to access that model. I use guideword heuristics, for instance.
- Learn session-based and thread-based test management.
- Have you studied the ET skills and tactics list that my brother and I published? Ask yourself how you stand on each item.

You should also develop a colleague network and make use of it. For that matter, I do free coaching over Skype, as does Michael Bolton and Anne-Marie Charrett, also. Come see me for a session.
**Question**: How little can a user know about the application’s goal, purpose, and assumed workflow in order to do effective exploratory testing?

*Brian J. Noggle, Springfield, Missouri*

**James**: Effective testing requires knowledge of the product. Exploratory testing is a way of testing whereby you learn as you test. Therefore, you can do effective ET with no knowledge (since it is effective to be learning) but the quality of your testing will not be at its best until after you’ve learned enough about the product to know how to observe it, control it, and recognize important problems in it.

Scripted testing is the same way. You can’t write great test scripts unless and until you know the product. That’s why most people create scripted tests by doing exploratory testing first.

**Question**: What are similarities and differences between exploratory testing and something like hacking, white hat penetration testing or security testing? Would it benefit these groups to interact more and learn from each other?

*Daw Cannan, Raleigh, North Carolina*

**James**: Penetration testing is inherently exploratory. Hackers are inherently exploratory. I’ve dabbled in this quite a bit, myself. I would recommend that testers read 2600 magazine and books about hacking.

**I think the scientific and philosophical illiteracy of our field explains the strange longevity of stupid ideas that don’t work, and have never worked. Our industry seems to be run by fairy tale logic.**

*James Bach*

**Question**: Can you provide one or more specific examples of what it looks like to successfully integrate exploratory testing with other test approaches on a medium to large size test project?

*Sean Stolberg, Seattle, Washington*

**James**: First, it’s already happening on every project that has ever been done. Exploratory testing is not some exotic weird thing. You do it whenever you investigate a bug, for instance. Geez. This is going on naturally. Have you ever been exploring a product and then decided to make an outline or test matrix to describe specific test conditions? If so, then you already know the answer to this question.

Mostly, when I’m asked this, the questioner turns out really to have been asking a different question, so I’ll answer that different question:

**Q**: Can you provide one or more specific examples of what it looks like to do great testing in a way that doesn’t worry and confuse busy-body managers who don’t understand testing or trust testers and therefore wish to apply silly management theories to testing such as “you should write down each test in a procedural scripted form?”

**A**: Yes, that’s why we created session-based test management. In this form of test management, unscripted testing is done in sessions. Sessions are time-boxes within which the testing occurs. Each session has a charter (a little mission) and results in a session report. We can create metrics from these which are reasonable and tend to give managers warm and fuzzy feelings inside. Managers get something to count, and testers remain relatively free to do their jobs. See more about it on my website.

**Question**: In your Rapid Software Testing training class and in your book Buccaneer Scholar, you start off by saying, apparently proudly, that you’re a high school dropout. I think the readers would be interested to hear about how that has helped your professional success?

*Bernie Berger, New York City, USA*

**James**: It helps me and it hurts me. But the reason I tell people is that I like how it defines me. (And people with degrees can say exactly the same thing.)

The kind of people I want to work with are those who understand that a vigorous habit of self-education is absolutely necessary in order to stay on top of a technical craft. By telling my institutional educational status, I hope to frighten away the kind of clients or colleagues for whom education is about symbols of power rather than skills and good ideas. I also want to establish myself as a maverick. That I’m a dropout communicates that I have a long history of doing things that I think are right, even when they are unpopular.
As for how being a dropout helps me, specifically: well, since I have no ceremonies or rituals to rely on for my success, I rely instead upon the simple idea that, for me, success comes through actually knowing a lot about my craft and being really good at it. I’ve known all along that I need to struggle to succeed. Nothing is handed to me because of my social status. I think my educational history has made me perpetually hungry, in certain ways.

I’m not against universities, though. I admire many people who are highly educated by institutions. Dr. Cem Kaner, for instance, is my hero, and he has helped me elevate my standards of scholarship.

**QUESTION** As a programmer, what can I do to maximize the value of both my (unit) testing effort, and my tester’s ET effort? How can I avoid wasteful overlap without leaving holes?

**Sean McMillan, Kalamazoo, Michigan**

**JAMES** When testers know testing and love testing there is rarely wasteful overlap. However, a simple thing you can do is encourage your tester to perform scenario testing, rather than merely simple functional testing. Scenario testing means creating complex and realistic situations with the product. Long flows of actions.

I also strongly suggest putting function-level logging into your system, so that you can automatically see what functions and features have been touched during the testing. This makes testing more playful, and yet still reliable.

**QUESTION** I feel as though as a profession, testers have failed to communicate the value of testing in a way that businesses can understand. This is part of the reason that so much misunderstanding still exists about what software testers do. It is also why so many otherwise capable business people cannot tell the difference between checking and testing, or understand the importance. Do you believe it is important to advocate for good software testing even today? What is your approach? Do you expect the testers in the trenches to get further someday than the first wave of thoughtful exploratory testers did? Or do you still fight the good fight and hope that business people will someday appreciate what makes testing good and understand how to identify context driven testing from other schools of thought? Have you given up on business people at this point? When you teach and create content, who is it for? Are there any areas you’ve given up on?

**Lanette Creamer, Seattle, Washington**

**JAMES** These are a lot of questions. I can address a lot of it in this way: Yes, those of us who study testing must vigorously advocate it to people who don’t study testing. It will always be that way, because testing is an intangible craft.

“Those of us who study testing must vigorously advocate it to people who don’t study testing. It will always be that way, because testing is an intangible craft.”

**JAMES** SDETs are a fetish of Microsoft and Google, primarily. You are in Seattle, so maybe that’s what you are seeing. Still, test automation is important, in certain parts of the industry. Some technologies require heavy duty tool support.

The challenge is the same for SDETs as for any other tester – can you get them to study their craft? I hear noises from Microsoft that they care about being good testers, but that’s not the impression I get from hearing the specifics of how the “Engineering Excellence” program works over at Microsoft. I’d like to see them really commit to testing excellence, and I believe they eventually will. When they do, they will recruit a variety of people into their test groups, because it takes a variety to build a powerful test team.
Software testing is an ever-changing occupation. Testers who care about what they do for a living constantly study/build skills. But, the test managers may not necessarily be doing so. They may be spending way too much time in meetings with other managers and not enough time in the trenches to see that the light at the end of the tunnel is a train...

How would you approach a test manager about this? How would you bridge the gap between what were the testing “standards” and how changing them is essential? How would you suggest that management gets on board with the evolution of testing?

Michele Smith, Maine

But regardless of change, I agree that we have a big problem with middle and upper managers who don’t comprehend testing. My general recommendation is that companies urgently grow test leads who can act as “master sergeants” or “centurions.” These test leads would not be tied up with endless meetings, but would bridge the gap between testing supervision and communication to management. A big problem I see is that the connection between management and actual testing breaks down at the tactical leadership level first, but then really collapses at the next level up. Too many testers seem frightened to push their frustrations and ideas up the management chain. They also lack the skill of explaining testing to management even when they have the nerve to do it. Into this vacuum, management then pours their weird ideas about controlling and measuring testing.

That’s why test case metrics are super popular, despite being about as meaningless to test progress as, say, a pile of chicken bones. If we had better bridges to management, we would not have such struggles.

STP is now accepting nominations for the 2nd Annual Software Test Luminary Award

A luminary is someone who is recognized for their lifetime achievement and excellence in software testing. The Luminary award will honor any software testing and quality assurance professional who is determined, persistent, and committed to improving a process or methodology.

Software Test Luminary award will be presented at the Software Test Professionals Fall 2011 Conference, October 24-27 in Dallas, Texas.

SUBMIT YOUR NOMINEE TODAY!

Learn more about the award including criteria and deadlines by visiting us online at: www.softwaretestpro.com/stpluminary
Maybe it's time to step back, take a breath, and embrace the changes that need to occur when you and your organization stop trying to test quality into code and transition into baking quality in from the start.

Are you in a quandary over how QA is supposed to work in an Agile Delivery Team? Were things better for you when you could count on having complete requirements to start from and completed systems to test to? Are you finding yourself over worked and stressed out with the constant churn of new requirements that Agile encourages and the constant set of new working software every two weeks?

Maybe it's time to step back, take a breath, and embrace the changes that need to occur when you and your organization stop trying to test quality into code and transition into baking quality in from the start. So, what's your new role, and what can you expect to do in this capacity?

Traditional waterfall software development methodology follows a time-honored progression of

- gathering all of a project's requirements up front
- analyzing requirements to come up with a solution design
- implementing design into code
- testing the code (by QA)
- deploying the code to production once QA says that it is "ready to ship."

Then some silly Agilist (yes, I'm one of them) questions what happens when you miss a requirement, new requirements comes along, or someone changes their mind. The normal reaction is that you have a very long cycle of events that start a lot of the process over, and contribute to a lot of waste from unnecessary rework. Since QA is near the end of this long process, when things change, and the amount of time left to stay on schedule gets tight, several effects are normally seen:

- People start saying that they need less time to test. I wonder if that might affect quality.
- People start trying to stop change from occurring by having things such as Change Review Boards. This really makes me hit my head with the palm of my hand. Now that the customer knows what they need, we try to inhibit that from changing anything. I wonder if that's a good outcome for the business?
- People blame QA for not wanting to test early. But QA has a point here. If you test one piece now, and another piece later, who is to say that the stuff you did later didn't have side effects that broke the stuff you finished before?

Does any of this sound familiar?

The short story is that the tried and true waterfall methodology is just too costly for standard software projects (let's not get into avionic control systems or radiation therapy machine software for the moment). It's costly because we assume that we can gather all of the requirements at the start of a project, that none of those requirements will ever change, and that we can perfectly plan for every development event in advance. My claim is...
that the waterfall methodology makes you comfortable in thinking that you can plan everything up front and then just execute on the plan and get it right the first time through. But it just doesn’t work like that in the real world! There will be mistakes, and the mistakes will cause costly rework. This rework will be more costly than it has to be because we find the defects late in the game.

As you know, the Agile software development process is different. For purposes of this article, the process is different in two very important ways:

- We limit planning details for things that are too far into the future, because things will change, and we want to wait until the last responsible moment when we have the most information to make the best decisions (Poppendieck).
- We work in self-organized teams of generalized specialists with roles that blur: Furthermore, everyone on these delivery teams will be needed to make the software: developers, business analysts, user experience people, architects, technical publications people, QA, and anyone else needed! The team’s job is to understand the problem, and figure out their solution. They are not told what to do. They are self-governed, and will strive to plan every day to figure out the optimal way to make the best use of all their resources to solve the problems being worked on.

So, in Agile, we start off without a complete understanding of everything. We have to discover not only the problem, but also the solution. And there is no one to tell us how to make the solution — that’s something else we have to figure out. Oh, and don’t forget to expect changes to occur. Is this why you’re sort of at odds with the process? It doesn’t seem fully baked. Well, I want to tell you that once you get into this mindset, there is no going back.

The problem that most QA people have with the process isn’t with the process itself. It’s with what their management thinks that they should be doing, and what they are used to doing to please their management. Most non-QA people confuse Quality Assurance personnel with the traditional QC personnel that come from the manufacturing world. In the manufacturing world, it is standard to fully design the process from beginning to end then start the assembly line in motion. QC people monitor goods coming off of the assembly line. Bad products are tossed in one pile, and good ones get placed in another. Wash, rinse, and repeat until the whistle sounds to the end of the shift.

As you know, the QA role is actually much different. Quality Assurance is all about ensuring that the goods delivered from the process are right from the start. The idea is to have zero defects coming off of the line. In manufacturing, using QC to toss the bad ones out is costly enough. But when we use QC in the software development world, we are asking people to start the process over when a defect is found (back to requirements, analysis, design, implementation, and testing). The really horrible part is that we have to start the QA testing all over again once the defect is remediated, because we can’t trust that something else is not messed up as a side effect of fixing our original defect. Talk about costly. This can be deadly to a company!

So, the trick is to embrace the real meaning of QA. Get back to designing systems that allow us to become part of the solution in making code work the first time and all the time. Stop being turned into QC people who are mere gatekeepers between the end of development and the beginning of deployment. After all, when you were hired, the title on the job description said QA, not QC, didn’t it? Your management would rather pay you to work smarter, not harder. Doing a QC sort of job is hard in the software world because you have to fight the tedium of performing the same regression tests over and over.

So, what can I suggest?

First, start by actually being part of the delivery team. QA people need the same understanding of the domain to test in as developers need to code from. In fact, the difference between requirements (in Agile, we call these features, and represent them in stories), and tests is almost non-existent. That is, if the requirement is to “produce a system that can add 12 to 13 and get 25”, the test for that is “Enter 12 into the system. Then enter 13 into the system. Add. Check that 25 is returned.” Do you find a lot of difference there? Again, working arm and arm with the rest of the delivery team and designing tests as the Agile stories are developed is the first step.

Now, what do you do when there are just no tests to develop for any particular day? Nothing? Watch YouTube for an entire day? No. You are a full member of an Agile team. As a team, you have to decide how to make best use all of your resources. That may mean that in a morning Scrum, you and your team decide that since there are no tests to develop you can do something else such as a coding, writing user documentation, or do some GUI work. It all depends on what you’re capable of and how much you are willing to stretch. Agile teams work together which is good for you, the team, and for your organization simultaneously.

Let’s talk about exactly how the testing part goes. When you design tests, can you give me a good reason that you don’t automate them as you develop them? Didn’t think so. You will get numerous benefits. The labor and tedium of regression testing goes away. You get a computer to do the grunt work and it never complains. The automated tests become a barometer of doneness. As more and more tests start working, you have hard metrics of how close you are to being done.

The final benefit from automated testing is that you get to apply your brain to challenge and test concepts which is why you were hired into a QA role to begin with. You are more than a QC machine who rejects code with a fail stamp. QA needs to be recognized as a valued player from the start of the process that bakes quality into the code from the beginning. QA people need to take an active role from the start and full-fledged membership in the Agile delivery team for your next project is where you can make your stand.

About The Author

Howie Deiner is an independent software consultant who specializes in Agile process and practices. He has a varied background spanning 25 years in the industry, with extensive domain knowledge in commercial software, aerospace, and financial services. He has played many of the roles in the development arena, such as developer, analyst, team lead, architect, and project manager.
I have yet to see a process where bug regression is also part of bug verification. It is not uncommon to sometimes see that new bugs are created as a result of a bug fix. Sometimes the change in the code inadvertently causes the existing functionality to break. As we have found a few cases where the existing functionality was no longer working, we felt the need to do bug regression. I will discuss more about bug regression in the following paragraphs, but I will limit the discussion specific to our process.

Test & Fix

In order to keep this discussion brief and simple, I will not go into the details of the current development environment, but will simply state that it is a Quasi-Agile environment. The application is built by working on multiple components at the same time over a series of repetitive processes. There is fairly good documentation of the user stories and design specs, but they do change throughout the course of development. As code is written, reviewed and unit tested by the developers, testers are creating and reviewing test scenarios, writing and reviewing (time permitting) test cases. In the Test & Fix phase, testing will primarily focus on the positive test scenarios (happy path). Here is an example,
The subsequent test scenarios will probably not be tested from the metric for simple reasons such as, the code for the Program Editor may not be in yet or, we don’t know the rules of interactions between the DC console and the Program Editor or the DC console and the change of page layouts. And so, it makes sense to have a shorter test cycle for Test & Fix. The next phase is ready to begin, as all the different components are built and integrated.

**Integration**

Since the bulk of the changes are out of the way, there is almost a static target to take aim at. It is very difficult to shoot down a moving target when user stories are continuously defined and redefined and the code is constantly changing to reflect the user stories. In some cases, new user stories are added and some existing ones are removed. At this phase of testing expectations are such that changes are mostly done, but it is not always the case. The hope is that there will not be new user stories and very few new definitions of any existing one at this point. De-scoping of a user story is fine as it doesn’t require any new work and rework from our part. As the target is fairly immobile at this stage, we are now poised to attack. For the same user story (DC4.1.4.1.1), the following test scenarios will be tested.

DC console will not disappear when:
- Page layouts are changed
- Applications are added and deleted within each page layout
- Applications are swapped within each page layout
- A program or function is opened
- A program or function is viewed
- A program or function is edited
- A program or function is executed
- Pages are dragged and dropped in tree view
- Pages are copied and pasted in tree view
- Pages are copied and pasted in page sorter view DC console will disappear when:
  - A new problem is added
  - Existing problem is deleted

---

"As code is written, reviewed and unit tested by the developers, testers are creating and reviewing test scenarios, writing and reviewing (time permitting) test cases.”  
Mohammad ALAM
DC console will disappear when:
- A new problem is added
- Existing problem is deleted

Since a significantly large number of test cases are executed during integration, this is perhaps the longest of all test cycles. As a result, a large number of fairly complex and difficult bugs are generated during this test cycle. As integration nears the end, the bug reporting frenzy flattens out gradually. The fact that fewer and fewer bugs are found near the end of the test cycle tells me a couple of things. Either most of the bugs have already been found or, the testers have simply run out of ways to find more bugs. The thought of running out of ideas for finding more bugs worries me the most. It is often a tester’s desire to improve on the depth and breadth of test coverage. But, which way does one go? Right or left to add the breadth or up or down to add the depth? How far left or right and up or down does a tester need to go? There is not an easy answer to these very difficult questions. However, the answer to some of these questions just might be in bug regression. Here is how it works.

Bug Regression

Once a new bug is logged and triaged, to make sure that it is indeed a bug, a test case is written (if we don’t have one already) to address the bug. But more needs to be done such as:

1) Regress Around the Bug:
A set of test scenarios, and a number of test cases based on those scenarios, needs to be created around the bug. So, next time when the bug is verified, the test cases associated with the bug will also be executed. Let’s take this bug (a true bug) for instance.

Bug Summary: DC console disappears when opening a program or function in the Program Editor.

Test Scenario: DC console will not disappear when a program or function is opened.

In this case, there is already a test case based on the test scenarios from the metric. As a matter of fact, the bug is found by executing the very test case that illustrates the test scenario. The cause of the bug is known from the bug tracking tool. It turns out that, the function GoToWidgetOnPage() does not handle the moving console between pages very well. The solution to this bug was to add code to this function to better handle the moving console between pages. In order to regress this bug, test cases need to run with the following test scenarios (see Test Coverage metric).

DC console will not disappear when:
- Page layouts are changed
- Applications are added and deleted within each page layout
- Applications are swapped within each page layout
- A program or function is opened
- A program or function is viewed
- A program or function is being executed
- Pages are dragged and dropped in tree view
- Pages are copied and pasted in tree view
- Pages are copied and pasted in page sorter view

Since test cases already exist, there is not a need to create any new test scenarios or write new test cases. The current test case provides good coverage for this bug.

2) Follow the Code: Since changes are made to function GoToWidgetOnPage(), it is important to know if the same function is being used in any other part of the application. The premise of going after other areas of the application has to do with the notion that they are guilty by association. This information has to come from the developers and will be tracked through a bug tracking tool. Bug tracking tools can be configured such that the developers can document the affected areas (GUI and functional) in the tool. It can be a simple text field where the developers will list all the areas affected by the change. Once this information is obtained a tester will either use the existing test cases, (if there are any), create new ones or do both to test the areas that use this function. As a result, next time this bug is verified as part of the regression, a tester will also test the parts of the application guilty of using the same code thus making sure that fixing this bug does not cause any unpleasant occurrences in any other areas of the application. It is possible that some bug fixes are strictly isolated and have no affect on any other areas. But for the bugs with shared code (most are, since codes are reusable), this approach of bug regression will significantly increase the depth and width of our test coverage in all directions.

Exploratory: The exploratory sessions usually start after the integration. This form of unscripted, unstructured and anything goes type of testing is actually very refreshing. The only scripted part is a small document with a few lines of text which explains the areas of the application which are to be tested. Exploratory sessions are like Easter egg hunting; a race to find as many bugs as possible with as many participants as possible. Sometimes very strange bugs come out of these exploratory sessions. Here is a true bug:
Bug Summary: Cursor position changes

Bug description:

- Open the application
- Create a program in Program Editor
- Enter a line of text
- Go to the end of the line by pressing the right arrow on the keyboard
- Do a mouse click on the current cursor position

Result: Cursor jumps to a different position.

This is probably a very minor issue and chances are the user will always use the mouse instead of the keyboard to change the cursor position. Since this is a bug, we may want to create a couple of more test scenarios around it such as:

- Going to the beginning of the line by pressing the left arrow on the keyboard
- Going to the middle of the line by pressing the right arrow on the keyboard

It is comforting to know that we usually don’t find too many show stoppers from the exploratory sessions. At this stage of the test cycle we don’t anticipate any critical issues but, we do find one or two every now and then.

In Conclusion

It is important to point out some of the difficulties of practicing bug regression. They are mainly two-fold:

- “Follow the code” part of bug regression requires configuring the bug tracking tool. It also requires participation from development and thus needs a coherent effort from both testers and developers for putting this in place.
- Adding more on to a tester’s plate can also be a tough sell.

Fortunately, we were able to see some reduction on bug-recurrences by regressing around the bug. Over time when all the pieces are in place, it will benefit both QA and development and the outcome will be a better product.

About The Author

Mohammad Alam started as a mainframe programmer and gradually migrated to c/s environment in the mid 90’s with the intention to become a software developer. Because of his programming experience, he was asked to be a part of test automation which gave him an opportunity to write code. As Mohammad started to learn about the world of test automation and QA, he realized that this is what he wants to do. He has been doing software QA ever since in various capacities, such as tester, test lead, automation engineer, automation lead and coordinator for offshore test teams.
HOW CHILDREN LEARN TO TEST
In the 1960s, during the height of social revolution, a gentleman named John Holt wrote a pocket-sized book, “How Children Learn”.

Holt’s argument in the book was simple: that children have an amazing capacity to learn. He suggested that by the age of three, children have learned a staggering amount. By that age, children have learned to drink from a cup, eat with utensils, walk, even mastered a foreign language. Given the opportunity, they may also be able to swim and read – all without stepping foot in a classroom, seeing a chalkboard, or following a lesson plan.

Holt went on to suggest meaning behind the observation that children learn through exploration and play. This means that the school-as-a-factory model was broken; that forcing children to sit at a desk and do a hundred math problems was both an ineffective way to each and would dull their curiosity. Instead of just preaching, Holt did an amazing thing: He tested his idea by going into schools and working with children.

I know. Crazy.

The rest of the book is about those experiments. The children made a number line, each number exactly one inch apart, on the floor and through the halls. The children asked to be involved, and worked on the line during their play time – and, through this line, they gained a perspective on numbers, their relationship to distance, counting and math skills. Holt showed the children the basic format of addition, stepped back, and watched them figure out how to add any numbers without carry-over up to three digits – all by themselves. Or there was the time when he brought pendulums of different sizes into the classroom, and the children derived the rules for the time it would take to swing as a proportion of the pendulum size.

About The Author

Matthew Heusser is a Consulting Software Tester, Software Process Naturalist, as well as a contributing editor for STQA Magazine. You can read more about Matt on his personal blog, "Creative Chaos", at http://xndev.blogspot.com, read his contributions to the STP Community blog at http://www.softwaretestpro.com/blog, or follow him on twitter at @mheusser.
Exploratory Methods

If you read testing books from the 1980s, you’ll likely find a dominant paradigm about test cases. The idea here is that we need to separate the coming up with the test idea from the actual work, enshrining the tests in documents, and the documents in a version control system. Testing is then done when the tests don’t run.

The assumptions here are staggering: that testing the same way (repeatability) is preferable to variation in tests, that humans who are running test scripts are somehow more effective than humans exploring the software, and that time spent documenting couldn’t be better spent testing something else.

Sure, there is a middle ground; I tend to document high-level test ideas in a spreadsheet, coloring rows green when the test is good, and red or yellow if there is a problem. I’m not talking about that, I’m talking about trying to exhaustively document what can and should happen for the entire interaction, at the click-type-click-inspect level.

That approach might have some value, but I’d rather have a computer do it than a human; I also hope you can agree that this is the complete opposite of exploratory methods.

The good news is that the 1980’s were a long time ago; the test community has come a long way since Cem Kaner first published the term ‘Exploratory Methods’ in his first edition of “Testing Computer Software” in 1988.

Today, exploratory testing is widely recognized as a valid way of thinking about testing – as a sort of open-ended investigation of a product or service. Likewise, we tend to use test cases more in the “quick light” method I described earlier, more as guidelines than pseudo-code to be executed. I am also pleased to see more and more recognition that “and nothing else strange happen” is implied at the end of every test case, and that while passing the acceptance tests may be necessary to accept the software, that does not mean acceptance testing alone will always be sufficient.

Those are all good things … but have you attended a test conference lately?

The typical software test conference, even today, has PowerPoint. Lots and lots of PowerPoint. And lectures. And desks.

What you probably won’t see at a software test conference is, well … people testing.

A conference or training event could be the chance to do more than just testing. For instance, you could experiment; observe someone else test; or explore new techniques without the risk of botching a production system. A conference or training event can be the time to sit back and ask yourself “how’s this working for me?” and try something else.

It’s a huge opportunity, and we’re missing it. Instead of testing software, we sit back and listen to PowerPoint presentations.

Sure, there’s a time for that. Some of the training is on leadership or conflict skills, but still, I wonder: where’s the testing? For that matter, couldn’t the sessions be more like workshops, with some exercises, experiments, and peer sharing, and less like the university lecture model invented in the Middle Ages?

We’re getting there. Last fall, Software Test Professionals Conference had an entire hands-on track, in which people could be butt-in-seat, hands-on-keyboard. The conference also had a tutorial that was really a hands-on performance workshop. Some tutorials and classes are already hands on; Rapid Software Testing, by James Bach or Michael Bolton, comes to mind as the most prominent. Several other conferences also have a test lab or competitive testing event, and I see more interactive training offered every year.

As much as we are getting there, we still have a ways to go.

If you care about the state of software testing in the world today, you probably care about how people are trained and the way they are taught to think. I hope you’ll join me in asking for more interactive training; no, I hope you’ll join me in demanding it.

You may ask, if I really believe this, shouldn’t I change the way I work and live?

Well, basically, yes. John Holt, along with a few other renegades such as Maria Montessori and John Gato, were big influences on the decision my wife and I made to home-school our children; they have also reinforced the ideas I already had about testing. Test training? Well, we’ve still got some work to do.

See you at the Software Test Professionals Conference in the fall?
That’s all you get

COMING SOON!
The next STP Professional Development Series II:

SOFTWARE TEST PROFESSIONALS TEST BOOK

By Kim H. Pries and Jon M. Quigley
Book length 60 pages | Download price $10.00


Copyright © 2011 STP LLC
DELIVERING VALUE & THE Testing Quagmire

How to create a value-add testing organization

by Paul FRATELONE
Testing Quagmire

Success be it an interim deliverable or great tool to raise the performance in the marketplace is a challenge for companies. Business goals driven by the nature of the end product/application will affect how quality is defined. Product managers, business owners, and end users will all be involved in defining quality and value. Each will have their own perspective if they have been satisfied. In this article, the discussion will surround how to create a value-add testing organization. The scope will cover business objectives and goal alignment, quality dimensions in support of the business goals and how investments in testing have a quantifiable return.

**Performance Pay and the Alignment of the Delivery Team**

Companies that have disparate cross-functional goals have more operational issues, system outages, customer emergencies, and obstacles in achieving their objectives than those with aligned goals. Goals will be different from company to company but generally, revenue targets, customer satisfaction rating, market share, solving business problems and challenges are some of the most common. Aligning goals to the success criteria of the end result attained in the marketplace is a great tool to raise the performance of the team. Instead of making some success be it an interim deliverable date defined by some project plan, the team measures performance on whether or not the end result attained the business value and objectives. When product/business owners, developers and testers are tied (at the wallet) to the end result, you will see an immediate increase in collaboration and teamwork. Similarly, when applied to an organizational goal to increase efficiency and lower cycle times. The delivery team examines how each could do their part in attaining this goal and looked for ways to assist their teammates in removing waste and indentifying efficiency enablers.

**Dimensions of Quality**

Aspects of quality or dimensions generally fall into the following categories: reliability, usability, maintainability, conformability and performance. The challenge here is that it is not normal to think of a test plan or strategy in terms of quality dimensions nor are business requirements categorized in such a manner. As testers, one needs to be able to articulate the tests in terms of these dimensions along with the effort to deliver it. A value-add testing team will communicate and quantify the effort to achieve the quality dimension(s) yielding the commensurate business value. In the case where application reliability and accuracy are paramount to achieving business objectives, the testing organization needs to quantify and communicate the test effort required to deliver reliability. This robust test coverage will have an increased cost as compared to an application where reliability had less of a business impact. To achieve this level, the number of tests will increase ensuring the team has met a highly reliable application (i.e. range, boundary, min, max and error testing etc.). Business owners can now make a connection between the goals, quality and the test effort required. In the case where the testing phase gets crunched from eight weeks down to four, the team can quantify the effort that is necessary to attain that stated level of quality/dimension (reliability, accuracy, performance) to meet the business goal(s). The time reduced in testing might be so great that the product will not meet its goal to increase application reliability because the time to test is no longer sufficient to certify the products functionality. There is increased risk to the business goals; there will be untested code deployed into production that could negatively impact the business. Testing teams unable to articulate quality dimensions/objectives in terms of the test effort to support the business most often have been labeled a low value add team.

**Keeping Your Eye On the Prize**

When you hear the battle cry of your business and technology leadership saying we need faster, better, cheaper delivery of our software products and we must be innovative while doing so, it can be daunting. It truly is a very tough task to accomplish all of these goals at the same time. It can be done, but not all at once. Ask yourself do you understand how this request aligns with the company’s goals and objectives? Just as important is to understand and help your team understand the connection from quality to testing. If sales forecasts require that the product is available in the market place by a stated date in order to maximize the window of opportunity for revenue, then time to market is the most important criteria. Those activities that will accelerate delivery time will be sought out by the team. Finding that balance where increased speed does not create a quality issue is how the team will proceed and make trade-offs accordingly. At some point the team will need to assess when does speed to market cause lower quality that could impact this objective and possibly other objectives? Balance across objectives needs to be considered before a team proceeds with course corrections. Making sure the business owner is clear about the test effort in relation to the quality objectives will lower the chances of surprises when the effort (time, duration, cost) is needed for the level of quality required.

Through my many years in QA, I have experienced many epiphanies about quality, testing and what these terms actually mean to business owners and end users. After 20+ years working for both large multi-national and small companies across several industries, I have become intensely aware of how business goals and objectives drive the many dimensions of measuring business value and quality. The nature of the end product/application will dramatically affect how quality is defined and therefore as testers, one’s approach and understanding of how to relate and articulate this will be instrumental in ensuring results have accurately validated attainment. Product managers, business owners and consumers/end users will all be involved in defining quality and value. Each will have their own perspective if they have been satisfied. In this article, the discussion will surround how to create a value-add testing organization. The scope will cover business objectives and goal alignment, quality dimensions in support of the business goals and how investments in testing have a quantifiable return.

Performance Pay and the Alignment of the Delivery Team
Companies that have disparate cross-functional goals have more operational issues, system outages, customer emergencies and obstacles in achieving their objectives than those with aligned goals. Goals will be different from company to company but generally, revenue targets, customer satisfaction rating, market share, solving business problems and challenges are some of the most common. Aligning goals to the success criteria of the end result attained in the marketplace is a great tool to raise the performance of the team. Instead of making some success be it an interim deliverable date defined by some project plan, the team measures performance on whether or not the end result attained the business value and objectives. When product/business owners, developers and testers are tied (at the wallet) to the end result, you will see an immediate increase in collaboration and teamwork. Similarly, when applied to an organizational goal to increase efficiency and lower cycle times. The delivery team examines how each could do their part in attaining this goal and looked for ways to assist their teammates in removing waste and indentifying efficiency enablers.

Dimensions of Quality
Aspects of quality or dimensions generally fall into the following categories: reliability, usability, maintainability, conformability and performance. The challenge here is that it is not normal to think of a test plan or strategy in terms of quality dimensions nor are business requirements categorized in such a manner. As testers, one needs to be able to articulate the tests in terms of these dimensions along with the effort to deliver it. A value-add testing team will communicate and quantify the effort to achieve the quality dimension(s) yielding the commensurate business value. In the case where application reliability and accuracy are paramount to achieving business objectives, the testing organization needs to quantify and communicate the test effort required to deliver reliability. This robust test coverage will have an increased cost as compared to an application where reliability had less of a business impact. To achieve this level, the number of tests will increase ensuring the team has met a highly reliable application (i.e. range, boundary, min, max and error testing etc.). Business owners can now make a connection between the goals, quality and the test effort required. In the case where the testing phase gets crunched from eight weeks down to four, the team can quantify the effort that is necessary to attain that stated level of quality/dimension (reliability, accuracy, performance) to meet the business goal(s). The time reduced in testing might be so great that the product will not meet its goal to increase application reliability because the time to test is no longer sufficient to certify the products functionality. There is increased risk to the business goals; there will be untested code deployed into production that could negatively impact the business. Testing teams unable to articulate quality dimensions/objectives in terms of the test effort to support the business most often have been labeled a low value add team.
Justifying your Investments in Testing

Investments require the establishment of sufficient business rationale in order to determine the value that this investment is going to bring to the business and thus justify required expenditures and resource allocations. Call it a business case or business justification; we anchor our investments in testing to meet quality objectives in support of business goals. We measure success (meeting specifications) through testing/validation. The ability to articulate the expected benefits and associated cost of quality through investments in testing will be decided by the merits of a business case/rationale. These goals can be qualified (reliable, scalable) and/or quantified (5% increase in sales revenue/ market share) but regardless, one needs to identify the value and benefits the investments (testing) will and have contributed to the business. A qualitative goal such as, “the product has to be more intuitive to the customer”, is much harder to measure; but in the end, even this intuitive goal can be measured quantitatively. One can quantify/measure website traffic, usage, duration on site, increased sales/products sold, features purchased and increased advertising revenues. The marketing department has likely done some studies on qualitative goals and it would be a good source for the data one will need. Some general guidelines to follow:

- Understand the business rationale, quality goals, & success criteria – discuss this with your business owners and stakeholders but do not forget about marketing and finance to ensure a full understanding
- Know what you want from this initiative and how it will contribute to fulfill your success criteria
- Quantify as many of your objectives/goals as possible
- Identify all the processes in support of the goals and business rational
- Set thresholds and targets
- Use different ROI models for different purposes for the same project
- Numbers can say whatever you want; don’t use ROI calculations to confirm a bad investment

Why Metrics

Software testing exposes human failure and naturally people do not like to admit mistakes. A high performing aligned team will rally around these failures and hold themselves all accountable to the resolution bypassing the individual aspect. Some will go to surprising lengths to hide mistakes or take steps to remedy them before they are discovered. Establishing a measurement program is an investment in success as it enables proactive behavior driving early detection of risks to the business. Metrics provide the ability to identify, resolve, and/or mitigate risk issues before they surface. The act of measurement is not a goal in itself and needs to be integrated into the culture, the delivery methodology or the team. Metrics are used to compare the current state of your process/end results with past performance and/or previous estimates. The results will identify trends (negative/positive) which enables the team to continue to adjust the course corrections. To be effective and drive improvements, metrics must not only be collected they must be used to understand if changes to delivery are yielding the intended efficiencies. Effective components include the following examples:

- Clearly defined software delivery issues affecting the business objectives and the data elements needed to provide insight
- Standardized processing of collected data
- Results need to be published on a regular basis otherwise progress will remain uncertain.
- Results must drive process improvements

“Software testing exposes human failure and naturally people do not like to admit mistakes.”

In summary, if an organization’s and/or department’s goals are not in line or in support of business goals it is important to share business goals with your peers and understand how the synergized goals can be more powerful to the organization. Spend the time and find out some information about the product, its’ users, the market and what quality dimensions are important to each and how to measure them. Protect brand image and good will as bad news travels very quickly in the social networking era. When customers delay their purchases, there is an immediate impact to operating revenue. They might start to lose confidence in your software; this translates to lost opportunities and negatively impacts future revenue streams. Keep the team aligned; assess the impact of change to meeting your goals and your chances for success increase dramatically. Justify and quantify the test effort. If the business owner understands the relationship and you have quantified the effort, your ability to explain the cost, risk, benefit equation of quality & testing will go a long way to ensuring your team is adding value to the business.

About The Author

Paul Fratellone has a 25-year career in information technology centralized in Testing & Quality Management. Performing every activity/task of the QA professional from QA Analyst to Director, he has experienced many challenges and opportunities to deliver quality through testing. Through the years and challenges, Paul has gained a keen insight in knowing how to tailor the process to fit the maturity level while still delivering incremental efficiencies.
The Business Value of Performance Testing program is designed for software test and quality assurance professionals responsible for performance testing projects. Professionals who are driven to make their performance testing more effective, efficient and valuable to their organization and the end-user will benefit from registering for the STP Online Summit. Learn new concepts and techniques to enhance your skills.

Register today by visiting us online or calling: 877.257.9531

SAVE $50 BY REGISTERING BEFORE JULY 12

<table>
<thead>
<tr>
<th>SESSIONS</th>
<th>Speakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Testing’s Value Proposition to the Business</td>
<td>Dan Downing</td>
</tr>
<tr>
<td></td>
<td>Principal Consultant, Mentora Group LLC</td>
</tr>
<tr>
<td>System Performance Cycle in The Enterprise</td>
<td>Scott Barber</td>
</tr>
<tr>
<td></td>
<td>Chief Technologist, PerfTestPlus, Inc.</td>
</tr>
<tr>
<td>Identifying Performance Testing Objectives &amp; System Performance Goals</td>
<td>Anne-Marie Charrett</td>
</tr>
<tr>
<td></td>
<td>Software Test Consultant, Testing Times</td>
</tr>
<tr>
<td>Interfacing with Developers/Designers</td>
<td>Mike Dwyer</td>
</tr>
<tr>
<td></td>
<td>Principal Agile Consultant, BigVisible</td>
</tr>
<tr>
<td>Strategies for Performance Testing Integrated Sub-Systems</td>
<td>Eric Proelger</td>
</tr>
<tr>
<td></td>
<td>Performance Lead, Hyland Software</td>
</tr>
<tr>
<td>Performance Testing Strategies for 3rd Party Interfaces (Including “SOA”)</td>
<td>Dan Bartow</td>
</tr>
<tr>
<td></td>
<td>Vice President and CloudTest Evangelist, SOASTA</td>
</tr>
<tr>
<td>Dealing with Non-Production-Like Test Environments (Including “Reverse Validation”)</td>
<td>Mark Tomlinson</td>
</tr>
<tr>
<td></td>
<td>Shunra Software</td>
</tr>
<tr>
<td>Interfacing with IT/Capacity Planning</td>
<td>Wilson Mar</td>
</tr>
<tr>
<td></td>
<td>Performance Architect, Mercury Software (now HP)</td>
</tr>
<tr>
<td>Release-over-Release Enhancements</td>
<td>Scott Barber</td>
</tr>
<tr>
<td></td>
<td>Chief Technologist, PerfTestPlus, Inc.</td>
</tr>
</tbody>
</table>

WWW.SOFTWARETESTPRO.COM/EVENT/1132
“Just back a little... bit... more...” Jenny struggled with her words, so much of her energy being consumed in processing the image in her viewfinder. Her assistant Martin drew the disc-shaped diffuser back an amount so small that the model could hardly notice. Back lights, diffused and reflected, flashed as Jenny took the picture. “OK, let’s see how that looks.” Jenny and Martin moved almost apprehensively to the computer screen to look at the photo that had been beamed wirelessly to the nearby laptop (yeah, the technology is cool and amazing). “Nope, not there yet. His face is still too dark, we’re not getting it.” Martin lunges to the reflector stand, turns it left, then right, Jenny again focuses on the image in the viewfinder. “Yes! That’s it! That’s got to be it.” Together with the art director, they again crowd around the laptop to see the image. “Ok, we got it; now let’s work on his expression.” As the model changes expression, freezes, lights flash, bits get beamed, images scanned for any possible flaw, sighs of relief, expressions of joy and excitement, when finally the ‘right’ frame flashes into view on the screen. And the whole process is repeated until the scene is ‘complete’. I’ll try not to romance photo shoots in this article since I’m sure it’s not all glamorous, but there are many wondrous things about the process that are worth highlighting in the context of exploratory testing. Let’s start with the end in mind; what does the end of shooting a scene really mean? What does the end of an exploratory session mean? I like the thought that both are creative processes without a definitive ‘end’. In both cases, it requires judgement and skill, sometimes of more than one person, to decide to move to the next target. The skills are multi-disciplinary, too. In testing, we investigate technical, functional, usability and aesthetic concerns simultaneously and independently, mixing and mashing them as we go until we believe we’ve captured the essence of the test target, that is, the feature we are investigating.

With this mindset, you can see how important it is to show your work and communicate your observations. Communicating only conclusions such as “pass or fail,” subverts your audience’s participation; it gives them a shortcut to drawing a conclusion but it also leaves them in the dark. That hardly describes making an informed decision. The idea that exploratory testing shines light on both good and bad characteristics of a feature might be new to testers that are used to going bug hunting. Observing what works needs to be in your repertoire just as much as observing what doesn’t. In some situations, the ages-old consulting adage of starting off with the bad news and ending with the good news - the solution, the way through - is a useful style when talking to other developers. It demonstrates to them that you have taken the time to understand their mental model of the way the feature works, and that gives your observations more credibility. In my personal practice I’ve taken this one step further. My goal is to eliminate bug reports in favour of feature reports, a reporting style that I’ve started to refer to as “feature advocacy”. My investigation results in a report, and that report might contain observations that I found delightful, and it may contain observations that didn’t match my expectations, or what I believed to be the expectations of those stakeholders I am investigating on behalf of. It’s the feature that goes backwards in the workflow if the observations cause the decision-maker to reject it; it’s the feature that will be referenced in the user and support manuals; it’s the feature that will actually be used. So for now, it makes sense to me to enhance and share information using the same basis. Calling it a feature report also gives me the chance to lavish praise on those that build delightful features; in practice it is possible with a bug report but at the same time, incongruent.

Let’s move on to the exploration itself and more about making decisions. As investigators, we certainly make decisions. We decide when to end the investigation as above, but along the way we also choose the way; some may be drastic changes of direction, others
may be slight course adjustments, depending on what we’ve found so far. I mentioned the “right” frame when describing the photo shoot. In testing, sometimes “right” is just as artful, that is, your characterization of the feature requires judgement and all the observational skill you can muster. In photography the automatic focus doesn’t take the place of the critical assessment of the resulting image. In fact, sometimes the automatic focus has to be abandoned in place of manual focus, depending on the lighting conditions and the requirements of the scene. You still go for the shot. In testing, automated test results are not our decision, they are input to it. They inform the test process, but do not define it. True liberation for the tester is to have a portfolio of styles, techniques, and heuristics to use as needed. The team might not permit automation, but you still go for the shot; that is, you still test, and you still decide your next step based on those results. The team might also have used unit-level and feature-level test automation to build the right thing the right way, but again you still have to evaluate and decide your next step based on those results. Ideally your next step is enhanced by that level of automation so that you can spend your energy on the really interesting points of investigation. As the photographer relies on her powers of observation, so does the exploratory tester. With sharpness taken care of, thoughts of composition can take over; similarly with the right thing built the right way, thoughts of stakeholders, personas, value, and delight can take over.

So as we explore, we take notes and we describe what we see in the “right” way and others can judge whether or not we know enough about that feature, or if we need to investigate further. As the photographer brings together elements of photography such as lighting and composition to create that perfect shot, an exploratory tester brings together elements of technology such as functionality, performance, and usability to characterize features. Composition in photography is arguably about five things: patterns, symmetry, texture, depth of field, and lines. To compose is to manipulate those elements to highlight the subject. What’s “composition” in testing? Are there only five? Are there even only five categories? Here are some examples to see the analogy through, but know that my message is for you to build your own set of heuristics, that is, your own way of highlighting what is good or bad about features.

- **Workiness.** You’ve heard of truthiness. This is like that, but different. Seriously, whether you can or cannot complete the feature’s happy-path scenario repeatedly is part of your feature report.
- **Patterns.** We use patterns as we observe things; they also help us communicate. On one project we extracted all the text from all the bug reports regardless of current status and created a word frequency chart using Wordle.net; we used this to highlight a hot spot in the application that wasn’t apparent before. See the attached image for an example of a word frequency chart created from bugs reported for Adobe Flash Player (generated from the first 1000 results from the public Adobe Bug and Issue Management System). Your feature report would identify if a feature is part of a pattern or not.
- **Consistency.** Highlight a feature that stands out from others, in good or bad ways. The feature may be knock-your-socks-off delightful, or it might be so different that it, uh, defies logic. I’ll never forget the first application one of our clients out-sourced; the application did exactly what they said it needed to do, but there was no File menu, no Edit menu, and no tab order on any of the screens. You get the picture. We expected consistency with other applications that might be found on a typical desktop even though that wasn’t in the requirements our client had sent to the supplier. This consistency problem was external consistency, but there is easily a form of internal consistency too that also might be an element of a feature report.
- **Utility.** How smooth might the user’s adoption of that feature be? Clearly related to workiness, yes, but there is more context that you can apply. I remember in our test lab deciding that a solution was ready for on-site user acceptance testing. The day testing started was a cold winter day and the temperature in the room where the test machine was located was about 5C (40F) so our business tester wore gloves. Writing on paper wasn’t a problem with gloves. Typing and using a mouse was a problem with gloves. We eventually did launch, but not with that feature in the workflow and not with the machines installed in that room.
- **Depth of Impact.** Direct, or downstream workflows affected, or both? Positively or negatively? You don’t have to work on a business intelligence team to know that data quality is a big deal for a lot of workgroups. Wouldn’t it be terrific to be able to report that a feature saves time downstream?
- **Pathways.** Was the experience the same no matter how you accessed the feature? Notice that I’ve not made my list about “composing” test cases since I generally don’t do that anymore. For my exploration, composing is about describing the feature and how I can continually improve at that. Shifting from focusing on bugs to focusing on features is a first step to guided exploratory testing, eloquently framed by Rob Lambert (http://pac-testing.blogspot.com/2009/03/normal-0-false-false-false-en-gb-x-none.html) and an idea that I’ve been promoting using checklists. And it’s easy thinking about elements to describe features in the negative sense but I encourage you, as part of your exploration, to practice thinking about features in the positive sense so that you can characterize them in both negative and positive ways; this builds credibility with those writing the software since it proves you have taken the time to understand what they’ve done.

In the end, it’s about helping the team and stakeholders to make informed decisions. Since they make decisions about features, it feels appropriate that I align my compositions with what they have already identified, described, discussed, demo’d and delivered. The enjoyable aspect of testing using an investigative, exploratory style is that nothing limits the way you do it. It’s your exploration and therefore your art. You just have to poke that box and do it.

---

**About The Author**

Adam Gera is a researcher, coach, and speaker for those that test software. He specializes in systems delivery methods, particularly methods that enhance communications within teams. Most recently Adam has been adapting Agile principles and methods to large-scale test management services for enterprise programs and projects. Adam has worked in IT for over 20 years initially as a developer and architect and over the last 8 years, quality assistant, test manager and coach to large projects.
Performance Requirements: An Attempt at a Systematic View
Performance Requirements: What is the Problem?

At first glance, the subject of performance requirements looks simple enough. Almost every book about performance has a few pages about performance requirements. Quite often a performance requirements section can be found in project documentation. But the more you examine the area of performance requirements, the more questions and issues arise.

Performance requirements are supposed to be tracked from the system inception through the whole system lifecycle including design, development, testing, operations, and maintenance. However different groups of people are involved in each stage using their own vision, terminology, metrics, and tools that makes the subject confusing when going into details.

For instance, business analysts use business terms. The architects’ community uses its own languages and tools (mostly created for documenting functionality so performance doesn’t fit them well).

Developers often think about performance through the profiler view. The virtual user notion is central for performance testers. Capacity planners use some mathematical terminology when they come up with queuing models. Production people have their own tools and metrics; and executives are more interested in high-level, aggregated metrics. These views are looking into the same subject – system performance – but through different lenses and quite often these views are not synchronized and differ noticeably. All of these views should be synchronized to allow tracing performance through all lifecycle stages and easy information exchange between stakeholders. Many existing approaches to describing performance requirements try to put these multi-dimensional and cross-dependent performance views into a set of simple flat templates designed for functional requirements.

IEEE Software Engineering Book of Knowledge (SWEBOK, http://www.computer.org/portal/web/swebo/) defines four stages for the requirements process:

- **Elicitation**: gathering requirements
- **Analysis**: elaboration and negotiation requirements
- **Specification**: documenting requirements
- **Validation**: making sure that requirements are correct

Before diving into specific stages of performance requirements process, let’s discuss the most important performance metrics (sometimes referred as Key Performance Indicators, KPIs). It is a challenge to get all stakeholders to agree on specific metrics and ensure that they can be measured in a compatible way at every stage of the lifecycle (which may require specific monitoring tools and application instrumentation).

Let’s take a high-level view of a system (Fig.1). On one side we have users who use the system to satisfy their needs. On another side we have the system, a combination of hardware and software, created (or to be created) to satisfy user’s needs.

### Business Performance Requirements

Users are not interested in what is inside the system and how it functions as soon as their requests get processed in a timely manner (leaving aside personal curiosity and subjective opinions). So business requirements should state how many requests of each kind go through the system (throughput) and how quickly they need to be processed (response times). Both parts are vital: good throughput with long response times usually is as unacceptable as are good response times with low throughput.

Throughput is a business requirement whereas response times have two components which include usability requirements as well as business requirements. Throughput is the rate at which incoming requests are completed. Throughput defines the load on the system and is measured in operations per time period. It may be the number of transactions per second or the number of processed orders per hour. In most cases we are interested in a steady mode when the number of incoming requests would be equal to the number of processed requests.

Defining throughput may be pretty straightforward for a system doing the same type of business operations all the time, like processing orders or printing reports when they are homogenous. Clustering requests into a few groups, such as small, medium and large reports, may be needed if requests differ significantly. It may be more difficult for systems with complex workloads because the ratio of different types of requests can change with the time and season.
**PERFORMANCE | REQUIREMENTS**

Throughput usually varies with time. For example, throughput can be defined for a typical hour, peak hour, and non-peak hour for each particular kind of load. In environments with fixed hardware configuration the system should be able to handle peak load, but in virtualized or cloud environments it may be helpful to further detail what the load is hour-by-hour to ensure better hardware utilization.

"**Response times**

*(in the case of interactive work)*

or processing times

*(in the case of batch jobs or scheduled activities)* define how fast requests should be processed."

Alex PODELKO

Homogenous throughput with randomly arriving requests (sometimes assumed in modeling and requirements analysis) is a simplification in most cases. In addition to different kinds of requests, most systems use a kind of session; some system resources are associated with the user (source of requests). So the number of parallel users (sessions) would be an important requirement further qualifying throughput. In a more generic way this metric may be named concurrency: the number of simultaneous users or threads. It is important, because connected but inactive users still hold some resources.

Quite often, however, the load on the system is characterized by the number of users. Partially it is coming from the business (in many cases the number of users is easier to find out). Partially it is coming from performance tests. Unfortunately, quite often performance requirements get defined during performance testing and the number of users is the main lever to manage load in load generation tools.

But the number of users doesn't, by itself, define throughput. Without defining what each user is doing and how intensely (i.e. throughput for one user), the number of users doesn't make much sense as a measure of load. For example, if 500 users are each running one short query each minute, we have throughput of 30,000 queries per hour. If the same 500 users are running the same queries, but only one query per hour, the throughput is 500 queries per hour. So there may be the same 500 users, but a 60X difference between loads (and at least the same difference in hardware requirements for the application – probably more, considering that not all systems achieve linear scalability).

The number of online users (the number of parallel sessions) looks like the best metric for concurrency (complementing throughput and response time requirements). However terminology is somewhat vague here, sometimes “the number of users” may have a completely different meaning:

- Total or named users (all registered or potential users): This is a metric of data the system works with. It also indicates the upper potential limit of concurrency. In some cases it may be used as a way to find out concurrency as a percentage of total user population, but definitely is not a concurrency metric.

- "Really concurrent" users: the number of users running requests at the same time: In most cases it is matching the number of requests in the system. While that metric looks appealing, it is not a load metric: the number of "really concurrent" requests depends on the processing time for this request. The shorter the processing time, the fewer concurrent requests we have in the system. For example, let’s assume that we got a requirement to support up to 20 “concurrent” users. If one request takes 10 sec, 20 “concurrent” requests mean throughput of 120 requests per minute. But here we get an absurd situation that if we improve processing time from 10 to one second and keep the same throughput; we miss our requirement because we have only two “concurrent” users. To support 20 “concurrent” users with a one-second response time, we really need to increase throughput 10 times to 1,200 requests per minute.

It is important to understand what users we are discussing. The difference between each of these three “number of users” metrics may be drastic.

Response times (in the case of interactive work) or processing times (in the case of batch jobs or scheduled activities) define how fast requests should be processed. Acceptable response times should be defined in each particular case. A time of 30 minutes could be excellent for a big batch job, but absolutely unacceptable for accessing a web page in a customer portal. Response times depend on workload, so it is necessary to define conditions under which specific response times should be achieved; for example, a single user, average load or peak load.

Response time is the time in the system (the sum of queuing and processing time). Usually there is always some queuing time because the server is a complex object with sophisticated collaboration, multiple components including processor, memory, disk system, and other connecting parts. That means that response time is larger than service time (to use in modeling) in most cases.

Significant research has been done to define what the response time should be for interactive systems, mainly from two points of view: what response time is necessary to achieve optimal user’s performance (for tasks like entering orders) and what response time is necessary to avoid website abandonment (for the Internet). Most researchers agreed that for most interactive applications there is no point in making the response time faster than one to two seconds, and it is helpful to provide an indicator (like a progress bar) if it takes more than eight to 10 seconds.

Response times for each individual transaction vary, so we need to use some aggregate values when specifying performance requirements, such as averages or percentiles (for example, 90 percent of response times are less than X). Apdex standard (http://www.apdex.org) uses a single number to measure user satisfaction.

For batch jobs, it is important to specify all schedule-related information, including frequency
(how often the job will be run), time window, dependency on other jobs and dependent jobs (and their respective time windows) to see how changes in one job may impact others).

It is very difficult to consider performance (and, therefore, processing requirements) without full context. It depends, for example, on the volume of data involved, hardware resources provided, and functionality included in the system. So if any of that information is known, it should be specified in the requirements. Not everything may be specified at the same point. While the volume of data is usually determined by the business and should be documented at the beginning, the hardware configuration is usually determined during the design stage.

**Technological Performance Requirements**

The performance metrics of the system (the right side of the fig.1) are not important from the business (or user) point of view, but are very important for IT (people who create and operate the system). Performance (technological) requirements are derived from business and usability requirements during design and development and are very important for the later stages of the system lifecycle. Traditionally such metrics were mainly used for monitoring and capacity management because they are easier to measure and only recently tools measuring end-user performance get some traction.

The most wide-spread metric, especially in capacity management and production monitoring, is resource utilization. The main groups of resources are CPU, I/O, memory, and network. However, the available hardware resources are usually a variable in the beginning. It is one of the goals of the design process to specify hardware needed for the system from the business requirements and other inputs like company policies, available expertise, and required interfaces.

When resource requirements are measured as resource utilization, they are related to a particular hardware configuration. They are meaningful metrics when the hardware configuration is known. But these metrics do not make any sense as requirements until the hardware configuration would be decided upon; how can we talk, for example, about processor utilization if we don’t know yet how many processors we would have? And such requirements are not useful as requirements for software if it gets deployed to different hardware configurations, and, especially, for Commercial Off-the-Shelf (COTS) software.

Only way we can speak about resource utilization on early phases of the system lifecycle is as a generic policy. For example, corporate policy may be that CPU utilization should be below 70 percent.

When required resources are specified in absolute values, like the number of instructions to execute or the number of I/O operations per transaction (as sometimes used, for example, for modeling), it may be considered as a performance metric of the software itself, without binding it to a particular hardware configuration. In the mainframe world, MIPS was often used as such metric for CPU consumption, but there is no such widely used metric in the distributed systems world.

The importance of resource-related requirements is increasing again with the trends of virtualization, cloud computing, and service-oriented architectures. When we depart from the “server(s) per application” model, it becomes difficult to specify requirements as resource utilization, as each application will add only incrementally to resource utilization. There are attempts to introduce such metrics. For example, the ‘CPU usage in MHz’ or ‘usagemhz’ metric used in the VMware world or the ‘Megacycles’ metric sometimes used by Microsoft (for example, see Exchange mailbox sizing [http://technet.microsoft.com/en-us/library/ee712771.aspx]). Another related metric sometimes (but rarely) used is efficiency when it is defined as throughput divided by resources (however the term is often used differently).

In the ideal case (for example, when the system is CPU bound and we can scale the system linearly just adding processors) we can easily find needed hardware configuration if we have an absolute metric of resources required. For example, if software needs X units of hardware power per request and a processor has Y units of hardware power, we can calculate the number of such processors N needed for processing Z requests as $N = \frac{Z \times X}{Y}$. The reality, of course, is more sophisticated. First of all, we have different kinds of hardware resources: processors, memory, I/O, and network. Usually we concentrate on the most critical one keeping in mind others as restrictions.

Scalability is a system’s ability to meet the performance requirements as the demand increases (usually by adding hardware). Scalability requirements may include demand projections such as an increasing of the number of users, transaction volumes, data sizes, or adding new workloads. How response times will increase with increasing load or data is important too (load or data sensitivity).

From a performance requirements perspective, scalability means that you should specify performance requirements not only for one configuration point, but as a function of load or data. For example, the requirement may be to support throughput increase from five to 10 transactions per second over the next two years with response time degradation not more than 10 percent.

Scalability is also a technological (internal IT) requirement. Or perhaps even a “best practice” of systems design. From the business point of view, it is not important how the system is maintained to support growing demand. If we have growth projections, we probably need to keep the future load in mind during the system design and have a plan for adding hardware as needed.

**Software Requirements Process**

In the next part we plan to discuss all stages of the performance requirements process, which include elicitation, analysis, specification, and validation, according to the IEEE Software Engineering Book of Knowledge (SWEBOK). The article will consider each stage and their connection with other software life cycle processes.

**About The Author**

Alex Podelko has specialized in performance engineering for the last fourteen years. Currently he is Consulting Member of Technical Staff at Oracle, responsible for performance testing and tuning of Hyperion products. Alex has more than 20 years of overall IT experience and holds a PhD in Computer Science from Gubkin University and an MBA from Bellevue University. Alex serves as a board director for Computer Measurement Group (CMG). His collection of performance-related links and documents can be found at [http://www.alexanderpodelko.com](http://www.alexanderpodelko.com).
The Key to a Stronger Test Team – Join the STP Community

Visit us online to deepen your expertise and elevate your career through relevant information, comprehensive education and targeted networking.

Premier Conferences | In-depth Training
Community Forums | Digital Publications
Articles | Informative Blog | Podcasts | Webinars
Professional Development Series (Free to Pro Members)

www.softwaretestpro.com
877.257.9531