Exploratory Testing: Finding the Music of Software Investigation

Articles

Posted by:

Posted on : 2008/3/18 1:03:16

My friend Steve is an exceptional classical guitarist. Watching him perform is inspiring – he has a rare mastery over the instrument and has spent years developing his craft. Steve can also explain the techniques he is using while he is playing, to teach and demonstrate how a student can learn and improve their own skills. Steve can make a guitar sing, and says that music is about tension and resolution. If music is all tension, you get uncomfortable as a listener. If it only resolves, it is boring, tedious repetition. Steve extends this concept to the actual physical actions that a guitarist employs to create certain sounds. For example, if you play with a lot of tension, you will limit your ability to do certain tasks. To make music, you need to find a balance between tension and resolution, and to find this balance, you need a mix of knowledge, skill and creativity.

Like Steve, my friend James Bach is also exceptionally skilled. James isn’t a guitarist, he is a software tester. James is also inspiring to watch while he practices his craft. He is a master of skilled exploratory testing: simultaneous test design, execution and learning [1]. James can also explain the testing techniques he uses while he is testing, to instruct testing students. The first time I saw him test software, I was reminded of my friend Steve. This time the tension and resolution wasn’t related to music composition or the execution of techniques on a musical instrument. Instead, the tension and resolution revolved around ideas. James would simultaneously design and execute tests based on his curiosity about the application. He would get feedback on a test, learn from it and design a new test. The tension was generated by the questioning nature of his tests, and the resolution emerged from the results of those tests. There was something almost musical in this interplay between the mind of the tester and the application being tested. This shouldn’t be surprising; as a software tester, James has a well-developed mix of knowledge, skill and creativity.

Author: Jonathan Kohl, Kohl Concepts Inc., www.kohl.ca

As a software testing consultant and musician, I meet a lot of skilled testers who do amazing work. Through experience and a lot of trial and error, they have developed skills they can’t easily explain. Unfortunately, with software testing, there aren’t as many obvious avenues for skill development as there are for musicians. Many software testers don’t realize that there are learnable exploratory testing skills they can develop to help them become even more valuable to software development teams.

When I work in a new organization, it doesn’t take long for me to get introduced to the testers who are highly valued. Often, when I ask testers about the times they did their best work, they apologize for breaking the rules: “I didn’t follow any pre-scripted test cases, and I didn’t really follow the regular testing process.” As they describe how they came through for the team on a crucial bug discover, they outline activities that I identify with skilled exploratory testing. Little do they know that this is often precisely why they are so effective as testers. They have developed analysis and investigative skills that give them the confidence to use the most powerful testing tool we have at our disposal: the human mind. Exploratory testing is something that

testers naturally engage in, but because it is the opposite of scripted testing, it is misunderstood and sometimes discouraged. In an industry that encourages pre-scripted testing processes, many testers aren’t aware that there are other ways of testing other than writing and following test scripts. Both software testing and music can be interpreted and performed in a variety of ways.

Western classical music is often highly scripted in the form of sheet music. Compositions are written in a language that performers can interpret with their voices or instruments. Despite a detailed well-disseminated shared “language” for printed music, it is difficult to perform music exactly the way the composer intended, particularly with musical pieces that have been around for centuries, because we don’t have the composer around anymore to consult. The opposite of playing from sheet music is improvisation, creating unrehearsed, unscripted music. A continuum exists between these two styles, because a composition is open to at least some interpretation by the performer, and some performers embellish more than others. Software that plays music is very precise in repeating what is input from sheet music, but is rarely as pleasant to listen to as a real performer. Music can be boring and tedious when played by a computer program, and full of life when played by a musician. At the other end of the spectrum, successful improvisation requires skill, and top performers study to develop a large breadth and depth of musical theory and technical proficiency on their instruments in order to successfully and creatively improvise.

In testing, test scripts that are written down are also open to interpretation by the test executor. Automating these tests is the only way to guarantee that they will be repeated exactly the same way, but like automating music, the lack of interpretation in execution can limit the results. A computer can only find the problems we predict and program it to find. Repeating scripted tests over and over can get boring, tedious, and may only feel like idea resolution, without the vital tension created by curiosity. At the other end of the spectrum, there is improvisational testing: exploratory testing. Pure exploratory testing means that my next test is completely shaped by my current ideas, without any preconceptions. Pure scripted testing and pure exploratory testing are on opposite ends of a continuum.

This analogy of music and software testing isn’t perfect however. Music is performed for entertainment purposes or as practice for musicians who are developing their skills. The end goal is entertainment for listeners, skill development, and the enjoyment of the musician. Software testing on the other hand isn’t generally done for entertainment, instead it is used to discover information. As Cem Kaner says, software testing is an investigative activity to provide quality-related information about software [2]. To gather different kinds of information, we want to be open to different interpretations, and to be able to look at a problem in many different ways. In music, improvisation can have negative effects when used at an inappropriate time or in an inappropriate manner. (When a musician plays a wrong note, we really notice it.) In software testing, exploring and improvisation, even when done wrong, can often lead to wonderful sources of new information. Inappropriate interpretations can be a hazard in musical performances, but on software projects, accidents, or “playing the wrong notes”, can lead to important discoveries. Furthermore, software projects are faced with risk, and exploratory testing allows for us to instantaneously adjust to new risks.

What does skilled exploratory testing look like? Here is scripted testing and exploratory testing in action. In one test effort, I came across a manual test script and its automated counterpart which had been written several releases ago. They were for an application I was unfamiliar with, using

technology I was barely acquainted with. I had never run these tests before, so I ran the automated test first to try to learn more about what was being tested. It passed, but the test execution and results logging didn’t provide much information other than “test passed.” To me, this is the equivalent of the emails I get that say: “Congratulations! You may already be a winner!” Statements like that on their own, without some sort of corroboration mean very little.

I didn’t learn much from my initial effort: running the automated test didn’t reveal more information about the application or the technology. Since learning is an important part of testing work, I delved more deeply. I moved on to the manual test script, and followed each step. When I got to the end, I checked for the expected results, and sure enough, the actual result I observed matched what was predicted in the script. Time to pass the test and move on, right? I still didn’t understand exactly what was going on with the test and I couldn’t take responsibility for those test results completely on blind faith. That violates my purpose as a tester; if I believed everything worked as advertised, why test at all? Furthermore, experience has taught me that tests can be wrong, particularly as they get out of date. Re-running the scripted tests provided no new information, so it was time leave the scripted tests behind.

One potential landmine in providing quality-related software information is tunnel vision. Scripted tests have a side effect of creating blinders - narrowing your observation space. To widen my observation possibilities, I began to transition from scripted testing to exploratory testing. I began creating new tests by adding variability to the existing manual test, and I was able to get a better idea of what worked and what caused failures. I didn’t want to write these tests down because I wanted to adjust them on the fly so I could quickly learn more. Writing them down would interrupt the flow of discovery, and I wasn’t sure what tests I wanted to repeat later.

I ran another test, and without the scripted tests to limit my observation, noticed something that raised my suspicions: the application became sluggish. Knowing that letting time pass in a system can cause some problems to intensify, I decided to try a different kind of test. I would follow the last part of the manual test script, wait a few minutes, and then thoroughly inspect the system. I ran this new test, and the system felt even more sluggish than before. The application messaging showed me the system was working properly, but the sluggish behaviour was a symptom of a larger problem not exposed by the original tests I had performed.

Investigating behind the user interface, I found that the application was silently failing; while it was recording database transactions as completing successfully, it was actually deleting the data. We had actually been losing data ever since I ran the first tests. Even though the tests appeared to pass, the application was failing in a serious manner. If I had relied only on the scripted manual and automated tests, this would have gone undetected, resulting in a catastrophic failure in production. Furthermore, if I had taken the time to write down the tests first, and then execute them, I would most likely have missed this window of opportunity that allowed me to find the source of the problem. Merely running the scripted tests only felt like repeating an idea resolution, and didn’t lead to any interesting discoveries. On the other hand, the interplay between the tension and resolution of exploratory testing ideas quickly led to a very important discovery. Due to results like this, I don’t tend to use many procedural, pre-scripted manual test cases in my own personal work.

So how did I find a problem that was waiting like a time-bomb for a customer to stumble upon? I treated the test scripts for what they were: imperfect sources of information that could severely limit

my abilities to observe useful information about the application. Before, during and after test execution, I designed and re-designed tests based on my observations. I also had a bad feeling when I ran the test. I’ve learned to investigate those unsettled feelings rather than suppress them because feelings of tension don’t fit into a script or process; often, this rapid investigation leads to important discoveries. I didn’t let the scripts dictate to me what to test, or what success meant. I had confidence that skilled exploratory testing would confirm or deny the answers supplied by the scripted tests.

Testers who have learned to use their creativity and intelligence when testing come up with ways to manage their testing thought processes. Skilled exploratory testers use mental tricks to help keep their thinking sharp and consistent. Two tricks testers use to kick start their brains are heuristics (problem-solving approaches) and mnemonics (memory aids) [3].

Musicians use similar techniques, and may recognize “the circle of fifths” as a heuristic to follow if they get lost in an improvised performance. (This isn’t a guarantee though, a heuristic may or may not work for you. When a heuristic is inappropriate, you simply try another.) Musicians tend to have large toolboxes of heuristics, and also use mnemonics as well. One example is “Every Good Boy Does Fine” which is used to remember the notes “EGBDF” on the lines of a staff. Skilled testers use similar tools to remember testing ideas and techniques.

I’m sometimes called in as an outside tester to test an application that is nearing completion. If the software product is new, a technique I might use is the “First Time User” heuristic. With very little application information, and using only the information available to the first time user, I begin testing. It’s important for me to know as little as possible about the application at this time, because once I know too much, I can’t really test the way a first-time user would.

To start testing in these situations, I often use a mnemonic I developed called “MUTII”. (A composer named Nicolo Mutii helps me remember it.) This mnemonic helps me maintain consistency in the way I think about testing. Expanding the mnemonic:
* Market—The targeted constituency of users this software is intended for. For example, “the finance department”, or “medium sized accounting firms”.
* Users—The actual users who will use the software. Who are the users? What do they do? What are their motivations for using our software?
* Tasks—What are the tasks that the users will use this software for? What are some typical tasks in their work?
* Information—What does the product tell me about the tasks it automates, and how I can perform them?
* Implementation—Is the software easy to use as a first time user? Is it reliable? Can I easily implement the tasks given the information and design of the product?

Before I start testing the application, I gather information about the market and the users from the business. This helps frame the kinds of tests I will develop when I use the software for the first time. If I’m not familiar with the market and users, I will also ask for typical tasks engaged in by the users.

When I start testing, I open my notebook to take notes of my observations and thoughts, and any bugs I find. (See Figure 1.) I begin designing a test in my mind, execute it with the software, and observe the results. I keep repeating this process, changing tests, and referring back to my MUTII mnemonic. Not only does each letter of the mnemonic help me frame my testing, but the acronym helps me quickly design and execute many tests under each section as I go. I may also use other heuristics and mnemonics as I test, if I find areas to explore differently or more deeply.

As I work through the application, I may find mismatches between the application and the market it is intended for, and the information supplied to users. If I have trouble figuring out the software’s purpose and how to use it, so will end users. This is important usability information that I write down in my notes. If the software isn’t usable, it isn’t going to sell. I invariably find bugs in this first testing session. I explore them, take notes so I can report them later, and design new tests around those bugs. After the session is over, I will have bugs to report and usability questions to ask. I will now have a model developed in my mind for testing this software. My brain will work on this model constantly, even when I’m not testing, and other testing activities will help build this and other models of the software.

Using heuristics and mnemonics helps me be consistent when testing, but I don’t let them rule my testing actions. If I observe something suspicious, I explore it. If something feels wrong, I investigate, and confirm or deny that feeling with defensible facts. It’s common to switch from heuristics and mnemonics to pure free-form improvisation and back again, or to improvise around highly structured tests. Exploratory testing—like improvising—helps me adapt my thinking and my actions based on what the software is telling me. This is a powerful concept. You can seize upon opportunities as soon as you observe them. Furthermore, you can adapt quickly to project risks, and discover and explore new ones. By developing skills to manage your thinking about testing, you no longer have to wait for spontaneous discoveries to appear out of thin air, and not be able to explain why you found a particular problem, or repeat it.

Developing exploratory testing skill puts you in charge of your testing approach. Skilled software testing, like skilled musicianship, is often referred to as “magic”, simply because it is misunderstood. Music follows a set of patterns, heuristics and techniques. If you know a handful, you can make music quite readily. Getting there by trial and error takes a lot longer, and you’ll have a hard time explaining how you got there once you arrive. Testing using pure observation and trial and error can be effective, but can be effective much more quickly if there is a system to frame it.

Skilled exploratory testing can be a powerful way of thinking about testing. However, it is often misunderstood, feared and discouraged. When we dictate that all tests must be scripted, we discourage the wonderful tension and resolution in testing, driven by a curious thinker. We limit the possibilities of our software testing leading to new, important discoveries. We also hamper our ability to identify and adapt to emerging project risks. In environments that are dominated by technology, it shouldn’t be surprising that we constantly look to tools and processes for solutions. But tools and processes on their own are stupid things. They still require human intelligence behind them. In the right hands, software tools and processes, much like musical instruments, enable us to realize the results we seek. There are many ways to perform music, and there are many ways to test software. Skilled exploratory testing is another effective thinking tool to add to the testing repertoire.

References

1. Bach, James. (2003) Exploratory Testing Explained
http://www.satisfice.com/articles/et-article.pdf
2. Kaner, Cem. (2004). The Ongoing Revolution in Software Testing. Presented at Software Test & Performance Conference, December, 2004, Baltimore, MD
http://www.kaner.com/pdfs/TheOngoingRevolution.pdf
3. Bach, James. (1999, November) Heuristic Risk-Based Testing. Software Testing and Quality Engineering Magazine http://www.satisfice.com/articles/hrbt.pdf

Jonathan Kohl, Kohl Concepts Inc. © 2007 First published by Kohl Concepts Inc. in July 2007

Originally published in the Winter 2007 issue of Methods & Tools