March/April 2010 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

MINIMIZE HANDOFFS
3 tips for agile teams

GET IT TOGETHER
The power of collaboration

Demystifying Exploratory Testing

Demystifying Exploratory Testing

by Jonathan Kohl

ISTOCKPHOTO

EARLY IN MY TESTING CAREER, I discovered exploratory testing (simultaneous test design, execution, and learning [1]) and tried to practice what experts recommended in their articles. I thought I was applying their ideas as best I could, and I knew what I was doing would be called "exploratory testing," but I was concerned. I wondered if I was missing something. I had this gnawing fear that I had misinterpreted or misunderstood a key concept. In short, I lacked confidence in my exploratory testing approach.

You can imagine my surprise when I met exploratory testing trainers who were more interested in *my* ideas and experiences than in my ability to recite theirs. Now that I mentor and train exploratory testers, I find that I, too, am more interested in other testers' unique practices, ideas, and experiences than my own. That's what's wonderful about thinking about testing from an exploratory perspective: It is human centered. Instead of dehumanizing the tester by reducing testing to a clerical task or by trying to automate away testers, we embrace their unique skills and experiences and the value they add to our projects. We use both software and thinking tools to give testers more power, not diminish it.

Recently, a test manager approached me after an exploratory testing training course. "Finally, I have words to describe what my team and I have been doing for years!" she said and thanked me for not making them feel stupid for being unconventional in their testing work. "You're the first consultant I've talked to who didn't tell me I was doing it wrong." She wasn't "doing testing wrong"—in fact, the development team was delighted with the results of her team's testing efforts. Trouble was, the consultants who had worked with her team were uncomfortable with her exploratory testing approach and the testing styles her team typically used. This is an important lesson: Just because an approach is unconventional doesn't mean it's wrong.

Exploratory Testing Styles

Because exploratory testing has less visible structure—for example, it lacks explicit steps and expected results—the test execution is different depending on who is doing the testing. There are different styles and variations that often yield similar results. What follows are some styles I've observed.

Intuitive

This is the most common style. Testers who haven't learned specific exploratory testing techniques tend to do this naturally. When you ask them what they are doing when they are testing in the absence of pre-scripted test cases, they may say, "I don't know why I did that," or that they are using their intuition. Intuition is just a fancy way of saying, "I am doing this because of the insight I have based on my experience and knowledge." It can appear to be random or chaotic, but when the tester is pressed for an explanation of what he did, a structure and purpose emerge.

Learning dominant

This is also very common. Even shops that insist that all their test cases be designed and recorded prior to formal testing use this exploratory testing style during test design. While writing test cases, the test designers will try out new software or a new feature that they need to learn about. As they learn, they invariably find bugs. This is also true of automated testing. As the automators learn about the system, they often will use an exploratory approach as they try to get the automation tool to interact correctly with the application.

The concept of *touring* [2] heuristics to learn about applications is becoming popular. Touring is a good test idea-generation activity. Testers create models of the software they are testing or use mnemonics to help them look at the application in different ways. This learning about the application leads to bug discoveries and test ideas. Some examples include creating coverage maps and following them or using different kinds of touring in a heuristic. For example, James Bach talks about touring the application to see if you can identify all the features. Or, try using the software the way you imagine different users would. Touring is a powerful tool for test idea-generation and discovery.

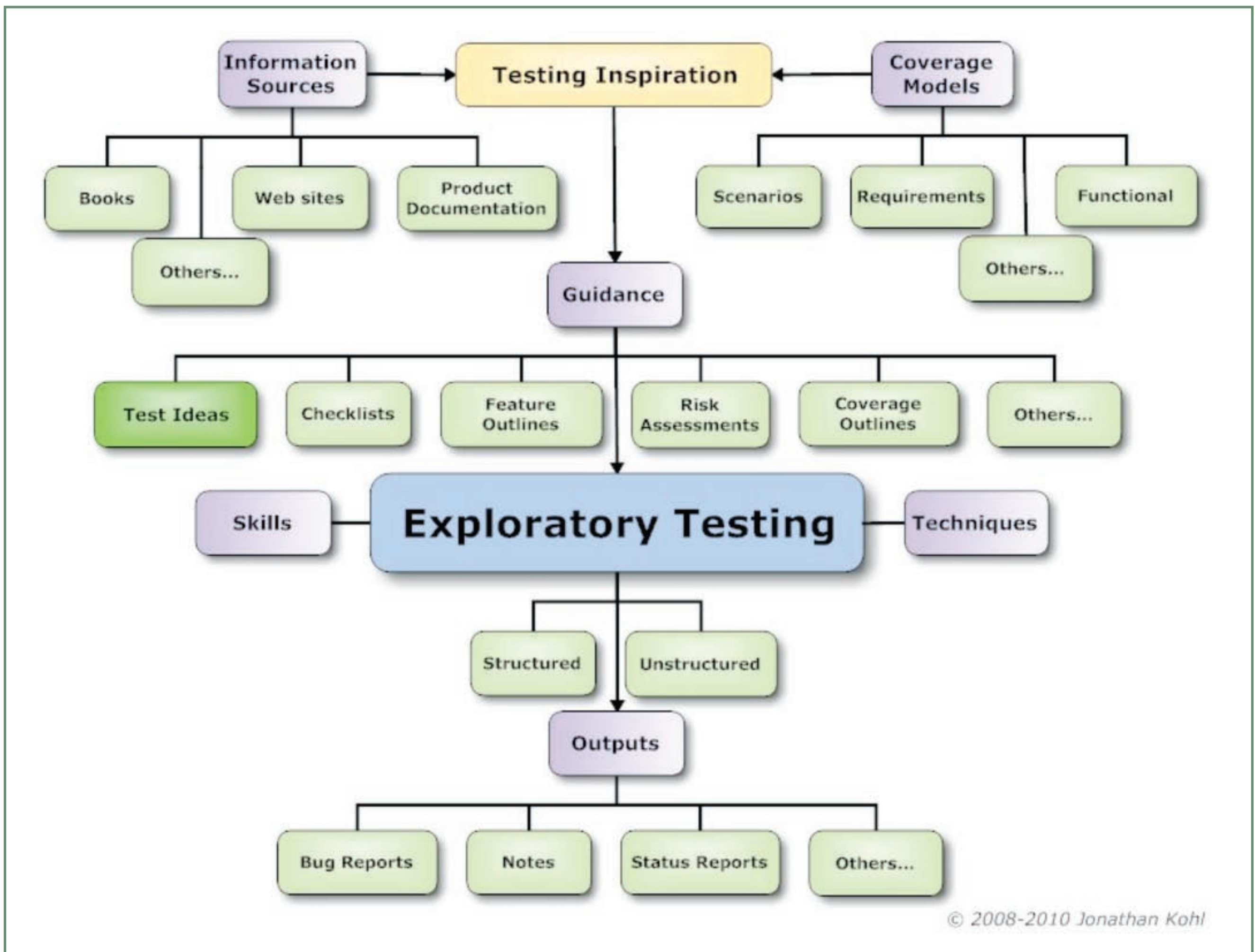


Figure 1: Exploratory testing mind map

Systematic

More advanced exploratory testing involves using a particular system, model, or strategy to guide your testing. This is done to be more thorough in looking at and testing the system, and to be more consistent in our exploratory testing work. When I started working with James Bach, one thing he noticed was inconsistency in my exploratory testing execution. I might use one set of test ideas and tools on one application but a different set on another. He taught me to use abstract models (see his Heuristic Test Strategy Model for more information) during test execution to guide my thinking and to help my exploratory testing be more repeatable and consistent.

Regression

Yes, even exploratory testers have to do regression testing. The difference between what scripted testers think of as regression testing and the exploratory testers' version is in the level of guidance used. Exploratory testers often use lightweight coverage outlines as guidance rather than test cases. The guidance of the outlines helps ensure we can easily and frequently repeat testing in desired areas. Using lightweight guidance with an exploratory approach to regression testing can be quite effective. Stakeholders have a good idea of what is being covered, such as features, functions, user scenarios, etc., while still allowing the test executors' natural variation to discover problems quickly.

Interactive automated/tool supported [3, 4]

Exploratory testers often use automation tools to help them. For example, if regression testing is becoming a burden, some aspects of testing may be automated. Task automation focuses on speeding up activities like data loads or installing new builds and can extend to test setup for exploratory testing sessions. [5] These tests are run under the supervision and direct control of the tester who can step in and intervene manually at any time.

Exploratory testers also use test automation that runs unattended, particularly if the tasks are better served by a machine than a human. The interactive part comes in when exploratory testers use the tools to help them simulate different kinds of conditions and then observe the results, change factors, and rerun the tests.

Enough Words! How about a Picture?

Most people who understand the unscripted, improvisational sense of exploratory testing but worry whether they are

doing it right aren't necessarily doing anything wrong; they merely lack confidence. One way to banish a lack of confidence is to develop your testing skills. Describing exploratory testing can be difficult, but I've found the mind map in figure 1 to be helpful. As you look through the mind map, does it help your thinking about exploratory testing? Can you identify areas that resonate with you?

I use this mind map to show that, although there may be little visible evidence, exploratory testing still can be a disciplined, structured form of testing. It's just that the tester keeps track of more of the structure and guidance in his head. Exploratory testing isn't just "playing with a product until it breaks," and it doesn't need to be a set of simple "quick tests." Often improperly characterized as "superficial bug hunting" or "quicktests" [6], exploratory testing can, in fact, take us far beyond that.

Coverage Models

When testing, we have some idea of coverage, or how much we are going to test a product. Cem Kaner defines coverage as "… the amount of testing done of a certain type." [7] We can expand that to *the amount we are testing software using different approaches*. We reveal different information if we test an application in different areas, in different ways, or with different techniques and tools.

There are many ways of defining coverage [8], which are called *coverage models*. A coverage model describes how you are going to test the software. Because testing from different perspectives reveals different kinds of information, it's better to use different kinds of coverage models than one single model. [9] For example, we could use a requirements-based approach and define a certain amount of testing to verify the software meets requirements. We could also determine coverage using different testing techniques, such as defining coverage according to security, performance, accessibility, etc. and defining an appropriate amount of testing in each area. We will usually use one model at a time when testing, but it is not uncommon to change models on the fly.

Choosing appropriate models can be determined by how we want to mitigate risk on a project or through other motivations. It depends on what our stakeholders are looking for. For example, with software startups, stakeholders often are more concerned with how testing information affects their ability to sell their product (capitalize on potential rewards) than in direct risk mitigation.

James Bach's SFDPOT "San Francisco Depot" mnemonic [10] (an abstract model for testing) spells out six possible models of coverage for exploratory testers: the *structure* of the application, its *function*, the *data* it uses, the *platforms* it runs on, the *operations* its users typically engage in, and the impact of *time* on the system.

One lightweight method of keeping track of what was tested and when is to use *coverage outlines*. Coverage outlines are physical documents that often take the form of checklists in a spreadsheet or visual diagrams that show user flows through the system. Their purpose is to direct and guide testing and to communicate what was tested and why. They are defined at a higher level than test cases, spelling out products, areas to be tested, and lists of testing ideas. A tester may run one test in a checklist based on that idea, or he may run more, depending on the task at hand, his schedule, and what he discovers while testing.

Information Sources

We use all sorts of information when testing. With experience, we grow our own testing toolbox of ideas and tools, and we sometimes find it difficult to explain where our compiled knowledge comes from. This knowledge comes from our testing experience, as well as books, Web sites, product information, subject matter experts, other team members, and conference, course, or workshop materials. It is preferable to use many rich sources of information to help guide our testing. As information providers regarding the quality of the product under test, any information that helps us do our work is useful. The richer our information sources, the more informed our testing will be.

Guidance

When we test, we usually have some sort of guide. This can be as simple as a bug report that we use for a bug fix verification, a test idea, or a test goal or charter for a testing session. Guidance can be more formal in the form of test checklists, coverage outlines, visual coverage, feature maps, or test cases. What we use for guidance is informed both by our coverage models and our information sources. For example, we may have a coverage outline for functional testing that is informed by our own knowledge and experience, as well as by sources of information from our project and from external sources. This coverage outline will guide our testing.

Guidance also comes in the form of experienced team members and outsiders who can help you and your team. In my work with testing teams, I help them develop skills and strategies so they can accelerate their learning and reduce their trial and error. Using outside expertise such as a trainer, article, concept, or coach can help. Also, look for expertise within your team. More experienced teammates are a wealth of information and are great to bounce testing ideas off of.

Structured vs. Unstructured

Structured exploratory testing has more guidance and more visible tools and documentation. For example, session-based test management [11] is a method for adding rigor to recording exploratory testing so it may be reviewed or audited. Instead of writing test cases in advance, testers take detailed notes while testing. These notes are collected, reviewed, and stored for audit purposes. At the other extreme, we may engage in unstructured exploratory testing, which is sometimes characterized as "ad hoc" or "jumping around the application to find problems." Exploratory testing can be very unstructured or very disciplined—and it usually falls somewhere between the two.

Techniques

Since exploratory testing is an approach to testing, not a

technique [6], any testing technique can be utilized if it helps. Classic analysis techniques such as boundary value analysis, equivalence class partitioning, root-cause analysis, classification trees, and others can be applied to exploratory testing. If a technique helps you generate testing ideas that lead to better testing, use it. An exploratory testing approach can be used with any kind of testing technique, such as performance, load, functional, usability, accessibility, security … the list goes on and on.

Skills

It is difficult to narrow down a short list of skills useful for software testing. I've worked with a lot of different testers with different skill sets, and the testers tend to draw on different areas of experience. I see the following skills most often:

Strategic thinking: Pick an area of focus and adjust testing to provide information in areas that yield the most useful information.

Idea generation: Exploratory testers need to be able to think of different ways to test the software in the moment. If they see something strange, they adapt their testing. This takes practice, and to do it well, testers use mnemonics to help remember different strategies and techniques.

Investigation: Analyze the product and the space in which it operates, find areas of interest, and explore those areas using tools, processes, and testing techniques to reveal useful information.

Communication: It's important to remember what you did (if you didn't record a testing session) and explain your strategy and what problems you found. If we discover important information but can't communicate it in a variety of ways, that information doesn't help the team.

I tend to talk a lot about exploratory testing as investigation (something I picked up from Cem Kaner) and related investigational skills. Technical skills that we develop on software teams—from programming to technical writing—are a natural fit. Skills related to understanding the business, our customers, and how they use our software are also very useful. I've seen wonderful testing from former forensic accountants and stock traders. Both professions attract natural investigators who are taught to assess systems rapidly and listen to their "gut feel" or intuition.

Outputs

When we test, we discover information that stakeholders on the team need to help make decisions about the software. We provide information to other stakeholders in many ways:

Bug reports: Typically distributed to the technical team, bug reports need to be well written so programmers can easily reproduce and fix the bugs.

Notes: Personal notes help us as we make our testing observations visible and help us create reports that can be shared with our teammates.

Session sheets: Descriptions of testing activities captured during session-based exploratory testing can be used in regulated environments and can be used to help review our work with colleagues to see if we have overlooked anything. With teams that use both exploratory and scripted testing approaches, session sheets and personal notes can help in test case creation.

Status reports: From low-tech testing dashboards [12] to verbal reports to stakeholders, communicating status helps the rest of the team understand what we have tested and the state of the application under test.

Test coverage reports: Stakeholders want to know how much testing of a certain type we have completed. Historically, we have counted test cases, but using multiple models of coverage and explaining what we actually worked on during testing can provide richer information than bug counts or pass/fail metrics.

Quality criteria: It's important to have measures to determine whether we are making our quality commitments with our software. For example, HP has used the FURPS+ model of quality criteria. [13] When using these sorts of models, we can report the functional applicability of the system under test and the usability, reliability, performance, and support measures that we have agreed on as a team.

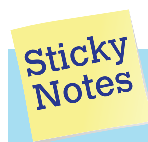
Blocking issues, problem areas: It's important to raise awareness to decision makers when we are unable to complete aspects of our work. It's also very important to provide them with a sense of our qualitative impressions of the products we are testing.

Effort and priorities: We can't test everything, and we are usually on a time budget, so we need to express what we have tested and what we think we need to test. We also need to demonstrate a sense of priority.

Our testing outputs are our only visible work products that other team members can see. We need to put effort into determining what others need, be accountable for our work, and provide stakeholders such as programmers, managers, and decision makers with the information they require. If the stakeholders are concerned with our approach, we need to adjust and provide them with what they need.

Ultimately, exploratory testing is determined by the mind of the tester who is doing the testing, and, as a result, it is interpreted in many different ways. Don't let fear, uncertainty, and doubt get in the way of your testing. Instead, use exploratory testing references to help guide your skill development and share your experiences with others. Sharing our ideas about testing, inviting scrutiny, and making our testing approach defensible are important. If your team appreciates the testing information you provide, don't worry that you're doing it wrong. If you are practicing your exploratory testing approach and your testing is adding value to your team's efforts, *you are doing it right*. **{end}**

jonathan@kohl.ca



For more on the following topic go to www.StickyMinds.com/bettersoftware.

- References