Exploratory Testing Explained

v.1.3 4/16/03

James Bach
james@satisfice.com

Exploratory software testing is a powerful approach, yet widely misunderstood. In some situations, it can be orders of magnitude more productive than scripted testing. All testers practice some form of exploratory testing, unless they simply don't create tests at all. Yet few of us study this approach, and it doesn't get much respect in our field. This attitude is beginning to change as companies seek ever more agile and cost effective methods of developing software.

Among the hardest things to explain is something that everyone already knows. We all know how to listen, how to read, how to think, and how to tell anecdotes about the events in our lives. As adults, we do these things everyday. Yet the *level* of any of these skills, possessed by the average person, may not be adequate for certain special situations. Psychotherapists must be *expert* listeners and lawyers *expert* readers; research scientists must scour their thinking for errors and journalists report stories that transcend parlor anecdote.

So it is with exploratory testing (ET): simultaneous learning, test design and test execution. This is a simple concept. But the fact that it can be described in a sentence can make it seem like something not worth describing. Its highly situational structure can make it seem, to the casual observer, that it has no structure at all. That's why textbooks on software testing, with few exceptions, either don't discuss exploratory testing, or discuss it only to dismiss it as an unworthy practice.

Exploratory testing is also known as ad hoc testing. Unfortunately, *ad hoc* is too often synonymous with sloppy and careless work. So, in the early 1990s a group of test methodologists (now calling themselves the Context-Driven School) began using the term "exploratory", instead. With this new terminology, first published by Cem Kaner in his book Testing Computer Software, they sought to emphasize the dominant thought process involved in unscripted testing, and to begin to develop the practice into a teachable discipline. Indeed, exploratory testing can be as disciplined as any other intellectual activity. Microsoft practices a formalized process of exploratory testing for the purposes of certifying third-party applications for Windows compatibility (http://www.satisfice.com/tools/procedure.pdf) and session-based test management (http://www.satisfice.com/sbtm) is a method specifically designed to make exploratory testing auditable and measurable on a wider scale.

Our use of the term exploratory testing is not in any way euphemistic. We use the words in their ordinary dictionary sense, just applied to testing. Our use of "Exploratory" particularly echoes its usage in geography, as the explorers of the Royal Geographic Society, in the eighteenth and nineteenth centuries, struggled with similar issues of methodology:

> *"So, to qualify as exploration a journey had to be credible, had to involve hardship and risk, and had to include the novelty of discovery. Thereafter, like cricket it was somewhat hard to explain to the uninitiated. But one element was absolutely vital; indeed it was precisely that which distinguished the age of exploration from previous*

ages of discovery and which necessitated the adoption of the word 'exploration.' It was, quite simply, a reverence for science."

-John Keay, The Permanent Book of Exploration

Just as a star may seem dim in the spectrum of visible light, yet burn brightly in the infrared, the simple idea of exploratory testing becomes interesting and complex when viewed in the spectrum of *skill*. Consider chess. The procedures of playing chess are far less interesting than the skills. No one talks about how wonderfully Emanuel Lasker followed the procedures of chess when he defeated Steinitz in 1894 to become world champion. The procedures of chess remain constant, it's only the choices that change, and the skill of the players who choose the next move. What makes exploratory testing interesting, and in my view profoundly important, is that when a tester has the skills to *listen*, *read, think* and *report,* rigorously and effectively, without the use of pre-scripted instructions, the exploratory approach to testing can be many times as productive (in terms of revealing vital information) as the scripted variety. And when properly supervised and chartered, even testers without special skills can produce useful results that would not have been anticipated by a script. If I may again draw a historical analogy, the spectacularly successful Lewis and Clark expedition is an excellent example of the role of skill in exploration:

"Lewis was the diplomatic and commercial thinker, Clark the negotiator. Lewis, who went specially to Philadelphia for training in botany, zoology, and celestial navigation, was the scientific specialist, Clark the engineer and geographer as well as master of frontier crafts...Both men were of great intelligence, of distinguished intelligence. The entire previous history of North American exploration contains no one who could be called their intellectual equal."

-Bernard De Voto, The Journals of Lewis and Clark

Now, let me come back from the frontier, for a moment. Of course, tests may be worth reducing to a repeatable scripted form for a variety of good reasons. You may have special accountability requirements, perhaps, or maybe there are certain tests that must be executed in just the same way, every time, in order to serve as a kind of benchmark. Exploratory testing is not against the idea of scripting. In some contexts, you will achieve your testing mission better through a more scripted approach; in other contexts, your mission will benefit more from the ability to create and improve tests as you execute them. I find that most situations benefit from a mix of scripted and exploratory approaches.

Exploratory Testing Defined

I've tried various definitions of exploratory testing. The one that has emerged as the all-around favorite among my colleagues is this:

Exploratory testing is simultaneous learning, test design, and test execution.

In other words, exploratory testing is any testing to the extent that the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests.

In this view of ET, virtually all testing performed by human testers is exploratory to *some degree*. This view treats ET as a thought process infusing much of what all good testers do, and it is found on a continuum between pure scripted testing, prescribed completely in advance in every detail, and pure exploratory testing where every test idea emerges in the moment of test execution. Because of this continuum, the question "Are you doing exploratory testing?" might be better stated as "In what ways and to what degree is your testing exploratory?"

For the purposes of this article, when I say exploratory testing and don't qualify it, I mean testing that is closer to pure ET than it is to pure scripted testing. In other words, testing that substantially satisfies the definition, above. When I want to talk about testing that is pure or very close to pure exploratory testing, with no script or outside direction to speak of, I call it *freestyle* exploratory testing.

Exploratory Testing is a Profoundly Situational Practice

Have you ever solved a jigsaw puzzle? If so, you have practiced exploratory testing. Consider what happens in the process. You pick up a piece and scan the jumble of unconnected pieces for one that goes with it. Each glance at a new piece is a test case ("Does this piece connect to that piece? No? How about if I turn it around? Well, it almost fits but now the picture doesn't match..."). You may choose to perform your jigsaw testing process more rigorously, perhaps by concentrating on border pieces first, or on certain shapes, or on some attribute of the picture on the cover of the box. Still, can you imagine what it would be like to design and document all your jigsaw "test cases" before you began to assemble the puzzle, or before you knew anything about the kind of picture formed by the puzzle?

When I solve a jigsaw puzzle, I change how I work as I learn about the puzzle and see the picture form. If I notice a big blotch of color, I might decide to collect all the pieces of that approximate color into one pile. If I notice some pieces with a particularly distinctive shape, I might collect those together. If I work on one kind of testing for a while, I might switch to another kind just to keep my mind fresh. If I find I've got a big enough block of pieces assembled, I might move it into the frame of the puzzle to find where it connects with everything else. Sometimes I feel like I'm too disorganized, and when that happens, I can step back, analyze the situation, and adopt a more specific plan of attack. Notice how the process *flows*, and how it remains continuously, *each moment*, under the control of the practitioner. Isn't this very much like the way you would assemble a jigsaw, too? If so, then perhaps you would agree that it would be absurd for us to carefully document these thought processes in advance. Reducing this activity to one of following explicit instructions would only slow down our work.

This is a general lesson about puzzles: *the puzzle changes the puzzling.* The specifics of the puzzle, as they emerge through the process of solving that puzzle, affect our tactics for solving it. This truth is at the heart of any exploratory investigation, be it for testing, development, or even scientific research or detective work.

What kinds of specifics affect ET? Here are some of them:

- the mission of the test project
- the mission of this particular test session
- the role of the tester

- the tester (skills, talents, and preferences)
- available tools and facilities
- available time
- available test data and materials
- available help from other people
- accountability requirements
- what the tester's clients care about
- the current testing strategy
- the status of other testing efforts on the same product
- the product, itself
    - its user interface
    - its behavior
    - its present state of execution
    - its defects
    - its testability
    - its purpose
- what the tester knows about the product
    - what just happened in the previous test
    - known problems with it
    - past problems with it
    - strengths and weaknesses
    - risk areas and magnitude of perceived risk
    - recent changes to it
    - direct observations of it
    - rumors about it
    - the nature of its users and user behavior
    - how it's supposed to work
    - how it's put together
    - how it's similar to or different from other products
- what the tester would like to know about the product

Instead of asking what test they are instructed to run, exploratory testers ask *what's the best test I can perform, right now?* Each of the considerations, above, may influence what test is needed. These factors change continuously throughout the course of the test project, or even from moment to moment during a test session. The power of exploratory tests can be optimized throughout the test process, whereas scripts, because they don't change, tend to become less powerful over time. They fade for many reasons, but the major reason is that once you've executed a scripted test one time and not found a problem, the chance that you will find a problem on the second execution of the script is, in most circumstances, substantially lower than if you ran a new test instead.

Practicing Exploratory Testing

The external structure of ET is easy enough to describe. Over a period of *time*, a *tester* interacts with a *product* to fulfill a testing *mission*, and *reporting* results. There you have the basic external elements of ET: time, tester, product, mission, and reporting. The mission is fulfilled through a continuous cycle of aligning ourselves to the mission, conceiving questions about the product that if answered would also allow us to satisfy our mission, designing tests to answer those questions, and executing tests to get the answers. Often our tests don't fully

answer the questions, so we adjust the tests and keep trying (in other words, we explore). We must be ready to report our status and results at any time.

An exploratory test session often begins with a charter, which states the mission and perhaps some of the tactics to be used. The charter may be chosen by the tester himself, or assigned by the test lead or test manager. Sometimes charters are written down. In some organizations, test cases and procedures are documented so vaguely that they essentially serve as charters for exploratory testing.

Here are some example testing charters for DecideRight, a decision analysis product:

- Explore and analyze the product elements of DecideRight. Produce a test coverage outline.
- Identify and test all claims in the DecideRight manual. (either use checkmark/X/? notation on the printed manual, or list each tested claim in your notes)
- Define work flows through DecideRight and try each one. The flows should represent realistic scenarios of use, and they should collectively encompass each primary function of the product.
- We need to understand the performance and reliability characteristics of DecideRight as decision complexity increased. Start with a nominal scenario and scale it up in terms of number of options and factors until the application appears to hang, crash, or gracefully prevent user from enlarging any further.
- Test all fields that allow data entry (you know the drill: function, stress, and limits, please)
- Analyze the file format of a DecideRight scenario and determine the behavior of the application when its elements are programmatically manipulated. Test for error handling and performance when coping with pathological scenario files.
- Check UI against Windows interface standards.
- Is there any way to corrupt a scenario file? How would we know it's corrupted? Investigate the feasibility of writing an automatic file checker. Find out if the developers have already done so.
- Test integration with external applications, especially Microsoft Word.
- Determine the decision analysis algorithm by experimentation and reproduce it in Excel. Then, use that spreadsheet to test DecideRight with complex decision scenarios.
- Run DecideRight under AppVerifier and report any errors.

If you find any of these charters ambiguous, I'm not surprised. They are intended to communicate the mission of a test session clearly and succinctly to testers who have already been trained in the expectations, vocabulary, techniques and tools used by the organization. Remember, in ET we make maximum use of skill, rather than attempting to represent every action in written form.

In freestyle exploratory testing, the only official result that comes from a session of ET is a set of bug reports. In session-based test management, each session of ET also results in a set of written notes that are reviewed by the test lead. It may also result in updated test materials or new test data. If you think about it, most formal written test procedures were probably created through a process of some sort of exploratory testing.

The outer trappings, inputs and outputs to exploratory testing are worth looking at, but it is the inner structure of ET that matters most—the part that occurs inside the mind of the tester. That's where ET succeeds or fails; where the excellent explorer is distinguished from the amateur. This is a complex subject, but here are some of the basics:

- *Test Design:* An exploratory tester is first and foremost a test designer. Anyone can design a test accidentally, the excellent exploratory tester is able to craft tests that systematically explore the product. That requires skills such as the ability to analyze a product, evaluate risk, use tools, and think critically, among others.
- *Careful Observation:* Excellent exploratory testers are more careful observers than novices, or for that matter, experienced scripted testers. The scripted tester need only observe what the script tells him to observe. The exploratory tester must watch for *anything* unusual or mysterious. Exploratory testers must also be careful to distinguish obervation from inference, even under pressure, lest they allow preconceived assumptions to blind them to important tests or product behavior.
- *Critical Thinking:* Excellent exploratory testers are able to review and explain their logic, looking for errors in their own thinking. This is especially important when reporting the status of a session of exploratory tests, or investigating a defect.
- *Diverse Ideas:* Excellent exploratory testers produce more and better ideas than novices. They may make use of heuristics to accomplish this. Heuristics are mental devices such as guidelines, generic checklists, mnemonics, or rules of thumb. The Satisfice Heuristic Test Strategy Model (http://www.satisfice.com/tools/satisfice-tsm-4p.pdf) is an example of a set of heuristics for rapid generation of diverse ideas. James Whittaker and Alan Jorgensen's "17 attacks" is another. The diversity of tester temperaments and backgrounds on a team can also be harnessed by savvy exploratory testers through the process of group brainstorming to produce better test ideas.
- *Rich Resources:* Excellent exploratory testers build a deep inventory of tools, information sources, test data, and friends to draw upon. While testing, they remain alert for opportunities to apply those resources to the testing at hand.

Managing Exploratory Testing

In many organizations, it's important to distinguish between a test manager and a test lead. The test manager usually has hiring and firing authority and other administrative responsibilities, whereas the test lead is focused only on the test strategy and tactics. For the purposes of discussing ET test management, I'll use the term test lead, even though a test manager may be fulfilling that role.

Freestyle exploratory testing can be managed in two ways: delegation or participation. With delegation, the test lead specifies the charters. Then the testers go off on their own, design and execute the tests to fulfill the charters, and report back. In practice, a particular tester is often permanently assigned to one set of components, so that the project benefits from an uninterrupted learning curve. The test reports that come back may be written or oral. Cem Kaner suggests regular meetings with testers to discuss test progress, at least once per week. He finds it useful to open the meeting with a standard question, "What is the most interesting bug you've found recently? Show it to me." In the session-based approach, test reports are written, and testers are interviewed at least once per day.

Managing by delegation essentially treats each tester as an executive who manages the value of his own time. Just as with executives, a tester who hasn't earned credibility as a productive

and responsible tester is not given big assignments, but rather is restricted to shorter exploratory sessions and subjected to more scrutiny. Being a leader of exploratory testers means being a coach of semi-independent creative agents.

Managing by participation means that the lead tests right alongside the rest of the testers. In practice, this is best for leads who have otherwise delegated their administrative and meeting attendance responsibilities to other people. Participation allows the lead to direct the test strategy in real time, and continuously demonstrate the behaviors he expects from the team. Since the test manager is ultimately responsible for the performance of the team, participation puts him in an excellent position to fulfill that responsibility. Many concerns about the potential for confusion or inefficient testing during ET tend to disappear when a test lead is intimately involved with the testing.

Most test leads will use a test coverage guide of some kind to help them organize the test effort. This guide may take the form of a test coverage outline or matrix, a list of risks, or even an old fashioned To Do list.

Team exploratory testing can be extremely powerful. In the experience of many test leads who've tried it, the social energy of people working together, hunting for bugs on the same equipment at the same time, often leads to more and better ideas than would otherwise come out if the same people worked independently. One way to organize team ET is to put testers into pairs and have them share one computer as they test. Another way I've done it is to have one tester "drive" at the keyboard while several others watch and comment. If the driving tester discovers a problem or has a question that needs to be researched, one of the watchers can break away to attempt to investigate that issue using another test platform. That frees the driving tester to continue the main thread of testing with less distraction. This method works especially useful as a blockbusting tool to get the test effort out of a rut or to help train testers in the technology of the product or about methods of test design.

Where ET Fits

In general, ET is called for in any situation where it's not obvious what the next test should be, or when you want to go beyond the obvious tests. More specifically, freestyle exploratory testing fits in any of the following situations:

- You need to provide rapid feedback on a new product or feature.
- You need to learn the product quickly.
- You have already tested using scripts, and seek to diversify the testing.
- You want to find the single most important bug in the shortest time.
- You want to check the work of another tester by doing a brief independent investigation.
- You want to investigate and isolate a particular defect.
- You want to investigate the status of a particular risk, in order to evaluate the need for scripted tests in that area.

Freestyle exploratory testing aside, ET fits anywhere that testing is not completely dictated in advance. This includes all of the above situations, plus at least these additional ones:

- Improvising on scripted tests.
- Interpreting vague test instructions.

- Product analysis and test planning.
- Improving existing tests.
- Writing new test scripts.
- Regression testing based on old bug reports.
- Testing based on reading the user manual and checking each assertion.

ET is powerful because of how the information flows backward from executing testing to re-designing them. Whenever that feedback loop is weak, or when the loop is particularly long, slow, or expensive, ET loses that power. Then, we must fall back on carefully pre-scripted tests. Another place we might use scripted tests is in any part of our testing that will be especially controversial, or are subject to a high degree of management or customer approval. But don't settle for weak tests just because they please the auditors. Consider using a combined exploratory and scripted strategy, and get the best of both worlds.

ET in Action

I once had the mission of testing a popular photo editing program in four hours. My mission was to assess it against the specific standards of the Microsoft Windows Compatibility Certification Program. The procedure for performing such a test is laid out as a formalized exploratory testing process. My goal was to find any violations of the compatibility requirements, all of which were clearly documented for me.

With this rather clear charter in mind, I set myself to test. Applying one of the simplest heuristics of exploring, I chose to begin by walking through the menus of the application, trying each one. While doing so, I began creating an outline of the primary functions of the product. This would become the basis for reporting what I did and did not test, later on.

I noticed that the Save As… function led to a very sophisticated set of controls that allow the user to set various attributes of image quality. Since I knew nothing about the technical aspects of image quality, I felt unprepared to test those functions. Instead, I started an issues list and made a note to ask if my client was willing to extend the time allowed for testing so that I could study documentation about image quality and form a strategy for testing it. Having made my note, I proceeded walking through menus.

A basic strategy of ET is to have a general plan of attack, but allow yourself to deviate from it for short periods of time. Cem Kaner calls this the "tour bus" principle. Even people on a tour bus get to step off it occasionally and wander around. The key is not to miss the tour entirely, nor to fall asleep on the bus. My first urge to leave the tour of the menus happened when I found a dialog box in the program that allowed me to control the amount of memory used by the application. This immediately gave me an idea (sudden ideas are encouraged in exploratory testing). Since one of the requirements of the Windows Compatibility program is stability, I thought it would be useful to set the product to use the minimum amount of memory, then ask it to perform memory intensive functions. So I set the slider bar to use 5% of system memory, then visited the image properties settings and set the image size to 100 inches square. That's a big canvas. I then filled the canvas with purple dots and went over to the effects menu to try activating some of the special graphical effects.

Okay, here comes an important part: I chose a "ripple" effect from the menu and *bam*, the product immediately displayed an error message informing me that there was not enough memory for that operation. This is very interesting behavior, because it establishes a standard.

I have a new expectation from this point forward: a function should be able to prevent itself from executing if there is not enough memory to perform the operation. This is a perfect example of how, in exploratory testing, the result of one test influences the next, because I then proceeded to try other effects to see if the rest of them behaved in the same way. Verdict? None of the others I tried behaved that way. Instead, they would crank away for five minutes, doing nothing I could see other than drive the hard disk into fits. Eventually an error popped up "Error -32: Sorry this Error is Fatal." and the application crashed.

This is a nice result, but I felt that the test wouldn't be complete (exploratory testers strive to anticipate questions that their clients will ask later on) unless I set that memory usage lever all the way up, to use the most memory possible. To my surprise, instead of getting the Error -32, the entire operating system froze. Windows 2000 is not supposed to do that. This was a far more serious problem than a mere crash.

At this point in the process, I had spent about 30 minutes of a 4 hour process, and already found a problem that disqualified the application from compatibility certification. That was the good news. The bad news is that I lost my test notes when the system froze. After rebooting, I decided I had learned enough from stress testing and returned to the menu tour.

I submit that this test story is an example of disciplined, purposeful testing. I can report what I covered and what I found. I can relate the testing to the mission I was given. It was also quite repeatable, or at least as repeatable as most scripted tests, because at all times I followed a coherent idea of what I was trying to test and how I was trying to test it. The fact that those ideas occurred to me on the fly, rather than being fed to me from a document, is immaterial. I hope you see that this is a far cry from unsystematic testing. It *would* be unsystematic "ad hoc" testing if I couldn't tell the story of my tests, couldn't remember what I tested or what my test strategy was, and couldn't relate my testing to my mission.

The Productivity of ET

There are no reasonable numbers; no valid studies of testing productivity that compares ET with scripted testing. All we have are anecdotes. Here are a couple anecdotes of mine.

I have taught testing classes where exploratory testers, testing on a system that automatically logged their work, have designed and executed almost a hundred tests each. Then when challenged to write a repeatable test procedure in the same amount of time, each person managed to create only one. You can argue that the repeatable test procedure was better, in some ways, than the exploratory tests. Maybe it was or maybe it wasn't. Personally, I felt that the test scripts produced were far less powerful than were the exploratory tests.

I have led teams in the testing of applications that had already been tested by scripted means, and found dramatic problems in them. In one case, without any preparation or test materials, it took my team only ten minutes to crash a network appliance so badly that its internal hard drive had to be reformatted. True, it took longer than ten minutes to investigate and isolate the problem, but that would also have been true for a scripted test procedure, assuming it found the problem in the first place.

I helped one large company start an exploratory test team in a department surrounded by scripted testers. They found that the ET people, unencumbered by the maintenance associated

with test artifacts, were able to pursue more leads and rumors about risk. They found more bugs, and more important bugs. Three years later, the team was still in place.

A man approached me at a conference and said that he ran an exploratory test team for a large telecom company. He said his team moves from project to project, and the reason he's allowed to continue his work is because their metrics show that his team find four times as many problems in a given period of time than do scripted testers.

These vignettes don't prove anything. I include them simply to pique your interest. The truth is, productivity depends upon a lot of factors. So, work with it, and gather your own experiences.

Exploratory testing can be described as a martial art of the mind. It's how you deal with a product that jumps out from the bushes and challenges you to a duel of testing. Well, you don't become a black belt by reading books. You have to work at it. Happy practicing.