Session-Based Test Management

by Jonathan Bach, jon@satisfice.com
(first published in *Software Testing and Quality Engineering* magazine, 11/00)

I specialize in exploratory testing. As I write this I am, with my brother, James, leading an exploratory test team for a demanding client. Our mission is to test whatever is needed, on short notice, without the benefit, or burden, of pre-defined test procedures. There are other test teams working on various parts of the product. Our particular team was commissioned because the product is so large and complex, and the stakes are so high. We provide extra testing support to follow up on rumors, reproduce difficult problems, or cover areas that lie between the responsibilities of the other testers.

Unlike traditional scripted testing, exploratory testing is an ad hoc process. Everything we do is optimized to find bugs fast, so we continually adjust our plans to re-focus on the most promising risk areas; we follow hunches; we minimize the time spent on documentation. That leaves us with some problems. For one thing, keeping track of each tester's progress can be like herding snakes into a burlap bag. Every day I need to know what we tested, what we found, and what our priorities are for further testing. To get that information, I need each tester on the team to be a disciplined, efficient communicator. Then, I need some way to summarize that information to management and other internal clients.

One way to collect status is to have frequent meetings. I could say: "Ok folks, what did you do today?" Some testers would give me detailed notes, some would retell exciting stories about cool bugs, some would reply with the equivalent of "I tested stuff" and fall silent without more specific prompting. And I'd be like the detective at the crime scene trying to make sense of everyone's story.

For this project, James and I wanted to do better than that. What if we could find a way for the testers to make orderly reports and organize their work *without* obstructing the flexibility and serendipity that makes exploratory testing useful? That was our motivation for developing the tool-supported approach we call *session-based test management*.

Testing in Sessions

The first thing we realized in our effort to reinvent exploratory test management was that testers do a lot of things during the day that aren't testing. If we wanted to track testing, we needed a way to distinguish testing from everything else. Thus, "sessions" were born.

In our practice of exploratory testing, a session, not a test case or bug report, is the basic testing work unit . What we call a session is an uninterrupted block of reviewable, chartered test effort. By "chartered," we mean that each session is associated with a mission—what we are testing or

what problems we are looking for. By “uninterrupted,” we mean no significant interruptions, no email, meetings, chatting or telephone calls. By “reviewable,” we mean a report, called a session sheet, is produced that can be examined by a third-party, such as the test manager, that provides information about what happened.

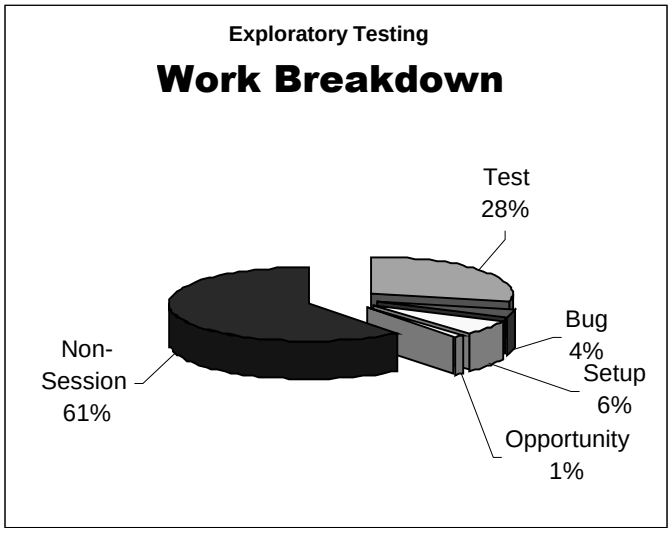
In my team, sessions last 90 minutes, give or take. We don’t time them very strictly, because we don’t want to be more obsessed with time than with good testing. If a session lasts closer to 45 minutes, we call it a *short* session. If it lasts closer to two hours, we call it a *long* session. Because of meetings, email, and other important non-testing activities, we expect each tester to complete no more than three sessions on a typical day.

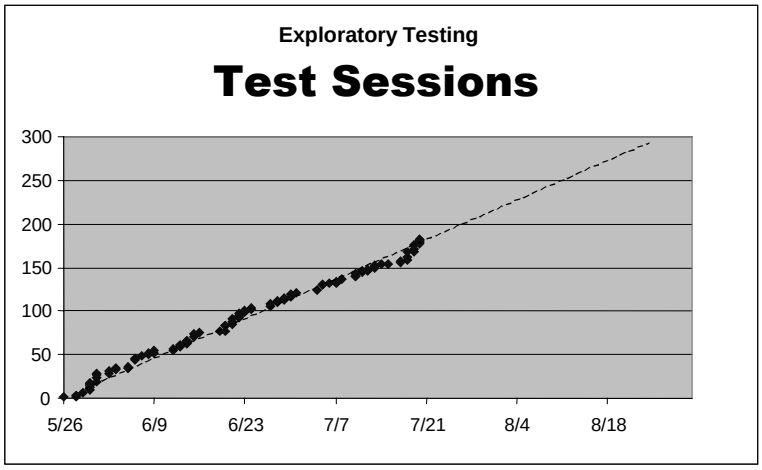
What specifically happens in each session depends on the tester and the charter of that session. For example, the tester may be directed to analyze a function, or to look for a particular problem, or to verify a set of bug fixes.

Each session is debriefed. For new testers, the debriefing occurs as soon as possible after the session. As testers gain experience and credibility in the process, these meetings take less time, and we might cover several sessions at once. As test lead, my primary objective in the debriefing is to understand and accept the session report. Another objective is to provide feedback and coaching to the tester. We find that the brief, focused nature of sessions makes the debriefing process more tractable than before we used sessions, when we were trying to cover several days of work at once.

By developing a visceral understanding through the debriefings of how much can be done in a test session, and by tracking how many sessions are actually done over a period of time, we gain the ability to estimate the amount of work involved in a test cycle and predict how long testing will take *even though we have not planned the work in detail*.

If there’s a magic ingredient in our approach to session-based test management, it’s the session sheet format: each report is provided in a tagged text format and stored in a repository with all the other reports. We then scan them with a tool we wrote that breaks them down into their basic elements, normalizes them, and summarizes them into tables and metrics. Using those metrics, we can track the progress of testing closely, and make instant reports to management, without having to call a team meeting. In fact, by putting these session sheets, tables and metrics online, our clients in the project have instant access to the information they crave. For instance, the chart in figure 1 shows that the testers are spending only about a third of their time actually testing. That corresponds to two sessions per day, on average, rather than three sessions. Since the chart represents two months of work, it suggests that there is some sort of ongoing obstacle that is preventing the testers from working at full capacity. Figure 2 allows us to get a rough sense of how many more sessions we can expect to do during the remainder of the project. The most recent week of data suggests that the rate of testing is accelerating.





Anatomy of a Test Session

From a distance, exploratory testing can look like one big amorphous task. But it's actually an aggregate of sub-tasks that appear and disappear like bubbles in a Jacuzzi. We'd like to know what tasks happen during a test session, but we don't want the reporting to be too much of a burden. Collecting data *about* testing takes energy away from *doing* testing.

Our compromise is to ask testers to report tasks very generally. We separate test sessions into three kinds of tasks: *test design and execution*, *bug investigation and reporting*, and *session setup*. We call these the "TBS" metrics. We then ask the testers to estimate the relative proportion of time they spent on each kind of task. Test design and execution means scanning the product and looking for problems. Bug investigation and reporting is what happens once the tester stumbles into behavior that looks like it might be a problem. Session setup is anything else testers do that makes the first two tasks possible, including tasks such as configuring equipment, locating materials, reading manuals, or writing a session report.

We also ask testers to report the portion of their time they spend “on charter” versus “on opportunity”. Opportunity testing is any testing that doesn’t fit the charter of the session. Since we’re in doing *exploratory* testing, we remind and encourage testers that it’s okay to divert from their charter if they stumble into an off-charter problem that looks important.

Aside from the task breakdown metrics, there are three other major parts of the session sheet: bugs, issues, and notes. Bugs are concerns about the quality of the product. Issues are questions or problems that relate to the test process or the project at large. Notes are a free-form record of anything else. Notes may consist of test case ideas, function lists, risk lists, or anything else related to the testing that occurs the session.

The entire session report consists of these sections:

- Session charter (includes a mission statement, and areas to be tested)
- Tester name(s)
- Date and time started
- Task breakdown (the TBS metrics)
- Data files
- Test notes
- Issues
- Bugs

Example Session Sheet

```
CHARTER
-----------------------------------------------
Analyze MapMaker’s View menu functionality and report on
areas of potential risk.

#AREAS
OS | Windows 2000
Menu | View
Strategy | Function Testing
Strategy | Functional Analysis

START
-----------------------------------------------
5/30/00 03:20 pm

TESTER
-----------------------------------------------
Jonathan Bach

TASK BREAKDOWN
-----------------------------------------------
```

#DURATION
short

#TEST DESIGN AND EXECUTION
65

#BUG INVESTIGATION AND REPORTING
25

#SESSION SETUP
20

#CHARTER VS. OPPORTUNITY
100/0

DATA FILES

#N/A

TEST NOTES

I touched each of the menu items, below, but focused mostly on zooming behavior with various combinations of map elements displayed.

View: Welcome Screen
- Navigator
- Locator Map
- Legend
- Map Elements
  - Highway Levels
  - Street Levels
- Airport Diagrams
- Zoom In
- Zoom Out
- Zoom Level
  - (Levels 1-14)
- Previous View

Risks:
- Incorrect display of a map element.
- Incorrect display due to interrupted repaint.
- CD may be unreadable.
- Old version of CD may used.
- Some function of the product may not work at a certain zoom level.

```
BUGS
-----------------------------------------------
#BUG 1321
Zooming in makes you put in the CD 2 when you get to a
certain level of granularity (the street names level) --
even if CD 2 is already in the drive.

#BUG 1331
Zooming in quickly results in street names not being
rendered.

#BUG <not_entered>
instability with slow CD speed or low video RAM. Still
investigating.

ISSUES
-----------------------------------------------
#ISSUE 1
How do I know what details should show up at what zoom
levels?

#ISSUE 2
I'm not sure how the locator map is supposed to work. How
is the user supposed to interact with it?
```

Tool Support

The session sheets are scanned by a tool we wrote in Perl. The tool performs about 80 syntax and consistency checks on each sheet. For instance, if a data file is referenced on a session sheet, the tool assures that the file has been placed in the appropriate data file directory. The charter section of the session sheet allows the inclusion of specific test area keywords so that each session can be associated with elements of a test matrix. In order to reduce errors, the legal values of these area keywords are stored in a separate file.

The output of the scanner is a set of text tables that help us tell the story of the test project. Each text table is in a delimited format suitable for importing to Excel for formatting and analysis. The scanner produces these tables:

- **Test Notes.** Test notes sections, by session ID.
- **Bugs.** Bug records, by bug ID and session ID.
- **Issues.** Issue records, by Issue ID and session ID.
- **Charters.** Charter statements and area keywords, by session ID.
- **Data Files.** Data file names, by session ID.
- **Session Breakdowns.** Session metrics, by session ID.
- **Coverage Breakdowns.** Session metrics, by area keyword.

- **Tester Breakdowns.** Session metrics, by tester name.
- **Day Breakdowns.** Session metrics, by day.
- **ToDo Sessions.** Incomplete session sheets.

Let me explain a bit more about the ToDo Session table. One method we use to dispatch exploratory testers is to create ToDo session sheets. A ToDo sheet is a sheet that has a charter only. All other sections are blank. When testers finish a test session, they look through the sheet folder, pick a ToDo sheet, and perform a session with that charter. The ToDo table produced by the scan tool is the list of the ToDo sheets currently in the session sheet folder. We call that list the "hopper". This arrangement provides a convenient way to charter an entire regression test pass at one time. Using another tool we wrote, called Todomaker, we can automatically generate ToDo sheets from an Excel test matrix. The hopper also helps us deal with requests for special testing by our clients. All we have to do is point at the hopper and ask them what priority we should give to their request, compared to the other sessions already in the hopper.

We also use a search utility that allows us to quickly locate, display, and print any session sheet that contains a specified string. This tool is important, because our client may ask, at any moment, to see the raw data behind our metrics and summary reports. In about five seconds, we can bring up every session sheet related to, say, the MapMaker Navigator dialog. Give us thirty or forty more seconds, and we can print all those sheets for you.

Metrics

The breakdowns tables, listed above, contain the task breakdown numbers for each session sheet. These are normalized so that the data from all session sheets can be combined. The metrics tell us what portion of on-charter work was spent on each of the TBS tasks, what portion of sessions were spent following up on opportunities, and how much time (in session units) was spent on non-session work. The latter measurement we get by assuming that a normal workday would include 5.333 normal sessions if testers did nothing but sessions. By keeping track of who was "on duty" on the test team, and on what days, we can know the theoretical maximum number of sessions that could have been executed in a given period of calendar time. We subtract the number of sessions that actually occurred, and that gives us the total amount of non-session work. This calculation is not super precise, but so far, we think it's precise enough for our purpose: to get a reasonably clear idea of how much actual testing is going on during the day.

These metrics help us make better estimates and predictions than ever before about such things as the learning curve of testers, the effects of adding a new tester to an established team, and the impact on productivity of testing a less testable versus a more testable product. For example, let's say that our team is cooking along at 3 sessions per tester day, on average, and we add a new tester. Since the testers will be interrupted by the need to provide coaching, they'll do fewer (or shorter) sessions than they otherwise would, or else they will spend more time in opportunity testing (since assisting a tester on another session is a form of off-charter test work). Meanwhile, debriefing the new tester will take more time at first. These effects will show up in the metrics.

Another simple thing we do with the metrics is to plot our total test sessions over time relative to the ship date. With such a graph, it's a simple matter to say, "At this rate, we have the bandwidth

to do about 200 more test sessions over the next four weeks. If there are 25 major areas of the product, and there are two builds per week, the deepest regression test we could do would provide only one session per product area per build. If we want some areas to be covered better than that, we need to do some triage—or squeeze in more sessions."

Although these metrics can provide better visibility and insight about what we're doing in our test process, it's important to realize that the session-based testing process and associated metrics could *easily* be distorted by a confused or biased test manager. A silver-tongued tester could bias the sheets and manipulate the debriefing in such a way as to fool the test manager about the work being done. Even if everyone is completely sober and honest, the numbers may be distorted by confusion over the reporting protocol, or the fact that some testers may be far more productive than other testers. Effective use of the session sheets and metrics requires continual awareness about the potential for these problems.

Notes From the Field

To give you a better feel for the problems with the session approach, and how we're dealing with them, here are some notes about the finer points of session-based test management:

- *What if a tester performs testing work outside of a session?*

Testers always get credit for testing. We call this the Golden Rule. If a tester does work that he later realizes should have been in a session, we have him create a session sheet to describe it, or add the work onto an existing session sheet. We do whatever is necessary to make the session sheets accurately reflect the testing that was done.

- *What if two testers work on one session?*

The scanning tool counts a session with two testers as *two* sessions. So, there's no incentive for testers not to cooperate with each other. During the debriefing, we just make sure that they really did work together.

- *What if a tester spends all day on one long session?*

This happens more often than we first anticipated. Some kinds of testing, such as "review the user manual" can drag on for hours without interruption. Instead of forcing testers to create multiple session sheets to report on what was really one session, we adjusted the syntax of the Duration keyword to allow multipliers. Hence a tester would report a four hour session as "`long * 2`" in the Duration field.

- *What if a tester can't avoid an interruption?*

We don't want test sessions to be interrupted, but sometimes it's necessary: a surprise meeting may be called, or a developer may need help reproducing a bug. In those cases we allow a session to be suspended and resumed later.

- *What if a tester can't fulfill the charter?*

If the tester can't get anywhere, the standing order is to abort the session and do a different one. If the tester makes a lot of progress, but the charter is much larger than he can complete, then the test manager should reduce the scope of the charter (in the debriefing) or create more sessions with the same charter.

- *What if the tester spends almost the whole session on "opportunity"?*

The test manager changes the charter to fit whatever it was that the tester did. As long as the charter substantially reflects what work was done, we're cool. The original charter can be re-used on a new session. In fact, it may require several sessions to fulfill any given charter. Naturally, if a tester chronically does not fulfill the charter, that's a coaching issue that is handled in the debriefings.

- *What's expected of a tester who's given an area that's already been tested?*

We expect the tester to review the test notes from earlier sessions and, where applicable, refine the earlier notes in the course of the present session. In this way, assuming that we've trained the testers in the art of test outlining, we expect that functional outlines and risk lists will get progressively better over time.

- *When testing bug fixes, how is that recorded?*

We use an area keyword titled "`strategy | bug regression`" and write a charter that instructs the tester to check each fix. We expect the tester to test in the general areas of the fixes to help assure that something else wasn't broken during the fix process.

- *How does the tester account for time writing up a session sheet? How about debriefing time?*

Session sheet completion is counted as session setup time, within the session. Debriefing is counted as non-session work.

- *How does a tester account for TBS tasks that occur simultaneously?*

What we really want to know is what things interrupt testing. So, if the tester figures out a way to test and do session setup, simultaneously, we don't count the setup time. If a test is running while the tester is investigating a bug, we don't count the bug time. Using this protocol, we can say, for the most part, that a report of 50% testing and 50% bug investigation means that the tester could have done twice as much testing, had the bugginess of the product not interrupted the test process.

- *How does the test manager conduct the debriefing?*

Apart from walking through the session sheet, we use an agenda summarized by the acronym "PROOF", which stands for:

    - **Past.** What happened during the session?

- **Results.** What was achieved during the session?
- **Obstacles.** What got in the way of good testing?
- **Outlook.** What still needs to be done?
- **Feelings.** How does the tester feel about all this?

- *What if the test manager is too overloaded to do debriefings?*

The debrief is the tester's chance to reveal their experiences, and the manager's chance to redirect or validate further effort. To the extent that the debriefings don't happen, the test manager is out of touch with the test process. The testers then begin drifting with the current of project events and their own preferences about what's fun to test, while the session sheets degrade into something more like rambling ransom notes or vaguely technical haiku. One way to help the situation could be to deputize senior testers to debrief the work of junior testers. As long as someway, somehow, the meetings happen.

"Seems like too much bureaucracy"

One colleague of mine, upon hearing me talk about this approach, expressed the concern that senior testers would balk at all the paperwork associated with the session sheets. All that structure, she felt, would just get in the way of what senior testers already know how to do. Although my first instinct was to argue with her, on second thought, she was giving me an important reality check. This approach does impose a structure that is not strictly necessary in order to achieve the mission of good testing. Segmenting complex and interwoven test tasks into distinct little sessions is not always easy or natural. Session-based test management is simply *one* way to bring more accountability to exploratory testing, for those situations where accountability is especially important.

James and I have less that a year's experience using this method, and we have a lot still to learn about the process and how it compares with traditional methods of managing testing. What have our experiences shown so far? Probably the single most important thing we've found is that this kind of test management weighs especially heavily on the test manager. We experimented with dropping the session debriefings, but that led to poor session sheets and meaningless metrics. We want good metrics, of course, but our approach produces a lot of them. Every day there's more, and sometimes we feel like we're swimming in spreadsheet tables. That also is a burden on the test manager, who must interpret and summarize the data. Session-based test management may be difficult for new test managers.

What the session stuff adds is a framework that helps clarify and track almost every element of the testing, from test planning to assigning work to coaching testers. It's that framework that intrigues us the most. If we're going to maintain the respect and support of management and developers, we must help our clients understand what it is that testers do, each and every working day.