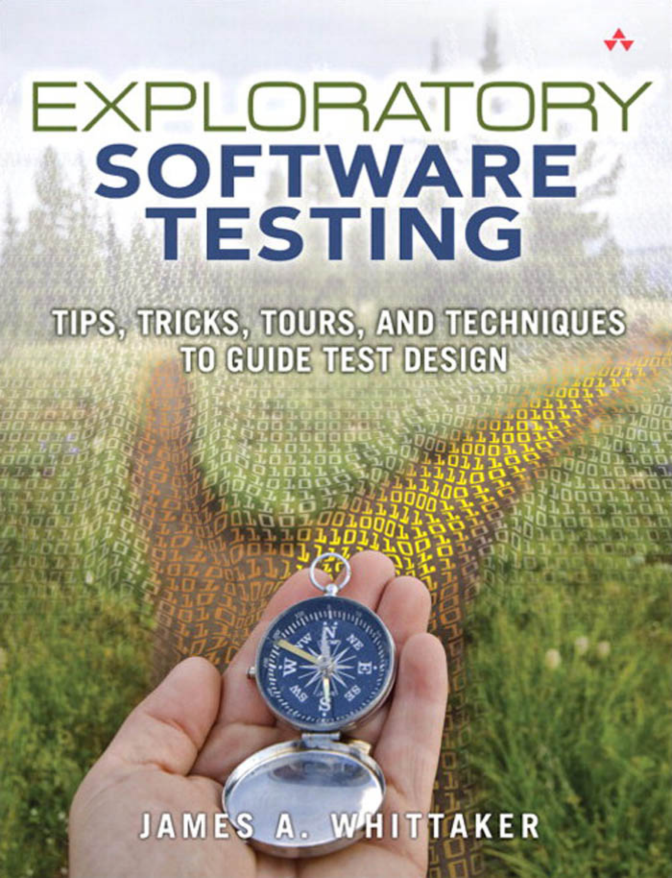
EXPLORATORY SOFTWARE TESTING

TIPS, TRICKS, TOURS, AND TECHNIQUES TO GUIDE TEST DESIGN



JAMES A. WHITTAKER

"Great book! Whittaker delivers ideas that are innovative, smart, and memorable. He really knows how to inspire engineers to think differently about testing."

—Patrick Copeland, Director of Test Engineering, Google

"James has perfected a fantastic manual testing methodology. The touring concept not only works, but it works so well that we've started sharing the tour concepts in the internal testing courses taught to all of our testers. If you want to bring your manual testing processes into the 21st century then read this book."

—Alan Page, Director, Test Excellence, Microsoft

"I began working with James at IBM in 1990. Even then, he was inspiring testers and developers to think outside the box. With this book he's taken his passion for software quality to a whole new level. Read it and watch yourself become a better tester. James is the real deal and this book should be read by every tester and software developer on the planet who cares about software quality or just wants to have more fun doing what they do."

—Kaushal K. Agrawal, Sr. Director of Engineering, Cisco Systems

"James Whitaker is a true visionary in the world of testing. uTest and our global community of QA professionals regularly look to James for inspiration, interpretation of trends, and overarching testing wisdom. Now he's finally written it down for everyone else and our industry will be smarter because of it."

—Doron Reuveni, CEO and Co-Founder, uTest

"Only someone like James Whittaker would think of combining the idea of tourism with software testing in such a novel way—and only James could pull it off. The tours approach provides a memorable and extremely effective mental model that combines right degree of structure and organization with plenty of room for exploration and creativity. Bugs beware!"

—Alberto Savoia, Google

"James is one of the best speakers around on software testing and reading his book is much like hearing him speak. If you want to increase your knowledge of testing and make yourself a better tester, this is the book for you."

—Stewart Noakes, Chairman and Co-Founder, TCL Group Ltd.

Most software has features that serve these purposes. For example, the business district for a word processor is the set of features to construct the document, write text, and insert graphics, tables, and artwork. The entertainment district, on the other hand, is the filler features for laying out pages, formatting text, and modifying backgrounds and templates. In other words, the work is in making the document and the "fun" part is making it look nice and represents an intellectual break from the actual work.

Tours through the entertainment district will visit supporting rather than mainline features and ensure that the two are intertwined in useful and meaningful ways.

The Supporting Actor Tour

I'm glad I started this chapter using London as my analogy because London is full of interesting sites and really cool buildings. On one of the many guided walking tours I've taken through nearly all parts of the city, I couldn't help but to be more attracted to the buildings that the guide was *not* pointing out than the ones he was telling us about. As he described a famous church that had historical significance, I found myself drawn to a short row house of buildings with rounded doors barely more than 5 feet high. It was like the hobbit portion of the city. On another stop, he was telling the story of pelicans that had been granted tenure in one of the city's parks. I found the pelican uninteresting, but a small island in the pond had a willow tree with a root structure that looked like dragon's teeth. I was enthralled.

Whenever salespeople demo a product or marketing touts some feature of our application, users are liable to be tempted by features nearby those in the spotlight. The *Supporting Actor tour* encourages testers to focus on those features that share the screen with the features we expect most users to exercise. Simply their proximity to the main event increases their visibility, and we must not make the mistake of giving those features less attention than they deserve.

Examples include that little link for similar products that most people skip in favor of clicking on the product they searched for. If a menu of items is presented and the second item is the most popular, choose the third. If the purchasing scenarios are the bread and butter, then select the product review feature. Wherever the other testers are looking, turn your attention a few degrees left or right and make sure that the supporting actor gets the attention it deserves.

In Chapter 6, Nicole Haugen shows how she used the Supporting Actor tour on the Dynamics AX client software.

The Back Alley Tour

In many peoples' eye, a good tour is one in which you visit popular places. The opposite of these tours would be one in which you visited places no one else was likely to go. A tour of public toilets comes to mind, or a swing through the industrial section of town. There are also so-called "behind the scenes" tours at places like Disney World or film studios where one can see how things work and go where tourists ordinarily don't tread. In exploratory testing terms, these are the least likely features to be used and the ones that are the least attractive to users.[9]

If your organization tracks feature usage, this tour will direct you to test the ones at the bottom of the list. If your organization tracks code coverage, this tour implores you to find ways to test the code yet to be covered.

An interesting variation on this theme is the *Mixed-Destination tour.* Try visiting a combination of the most popular features with the least popular. You can think of this as the Landmark tour with both large and small landmarks intermixed. It may just be that you find features that interact in ways you didn't expect because developers didn't anticipate them being mixed together in a single scenario.

Feature Interaction

It's a frustrating fact of testing life that you can test a feature to death and not find bugs only to then see it fail when it interacts with another feature. In reality, one would have to test every feature of an application with every other feature in pairs, triplets, and so on to determine whether they interact in ways that will make the software fail. Clearly, such an exhaustive strategy is impossible, and for the most part it is not necessary. Instead, there are ways to determine if two features need to be tested together.

I like to frame this problem as a series of questions. Simply select two candidate features and ask yourself the following:

- **The input question:** Is there an input that gets processed by both features in question?
- **The output question:** Do the features operate on the same portion of the visible UI? Do they generate or update the same output?
- **The data question:** Do the features operate on shared internal data? Do they use or modify the same internally stored information?

If the answer to any of these questions is "yes," then the features interact and need to be tested together.

In Chapter 6, Nicole Haugen, David Gorena Elizondo, and Geoff Staneff all use the Back Alley tour for a variety of testing tasks.

[9] It would be fair to ask whether we should test these features at all, but I feel that it is important that we do so. If the feature has made it into the product, it is important to someone somewhere. At companies like Microsoft and Google, the user base is so large that even the less-popular features can be used millions of times in a single day. In such cases, there really is no such thing as an unimportant feature. However, it is wise to proportion a testing budget in accordance with usage frequency as much as it is possible to do so.

The All-Nighter Tour

Also known as the *Clubbing tour,* this one is for those folks who stay out late and hit the nightspots. The key here is *all night.* The tour must never stop; there is always one more club and one last drink. Such tours, some believe, are tests of the constitution. Can you last? Can you survive the all-nighter?

For exploratory testers, the question is the same: Can the app last? How long can it run and process data before it just collapses? This is a real challenge for software. Because of the buildup of data in memory and the constant reading and writing (and rereading and rewriting) of variable values, bad things can happen if given time to do so. Memory leaks, data corruption, race conditions…there are many reasons to give time to your testing. And because closing and opening an application resets the clock and clears out memory, the logic behind this tour is to *never close the app.* This also extends to using features continuously and keeping files open continuously.

Exploratory testers on the *All-Nighter tour* will keep their application running without closing it. They will open files and not close them. Often, they don't even bother saving them so as to avoid any potential resetting effect that might occur at save time. They connect to remote resources and never disconnect. And while all these resources are in constant use, they may even run tests using other tours to keep the software working and moving data around. If they do this long enough, they may find bugs that other testers will not find because the software is denied that clean reset that occurs when it is restarted.

Many groups use dedicated machines that never get turned off and run automation in a loop. It is even more important to do this for mobile devices that often do stay on for days at a time as a normal course of usage. Of course, if there are different stages of reset, such as a sleep mode or hibernation mode, these can be used at varying rates as long as the software itself retains its state information.

Tours Through the Tourist District

Every city that focuses on tourism has a section of town where tourists congregate. It's full of souvenir shops, restaurants, and other things to maximize spending and ensure the profits of the local merchants. There are collectibles for sale, services to be bought, and pampering to be had.

Tours through the tourist district take several flavors. There are short trips to buy souvenirs, which are analogous to brief, special-purpose test cases. There are longer trips to visit a checklist of destinations. These tours are not about making the software work, they are about visiting its functionality quickly…just to say you've been there.

The Collector's Tour

My parents possess a map of the United States with every state shaded a different color. Those states all started out white, but as they visited each state on vacation, that state was then colored in on the map. It was their goal to visit all 50 states, and they went out of their way to add to their collection. One might say they were collecting states.

Sometimes a tour involves freebies that can be collected. Maybe it's a wine tasting or a fair with booths for children to work on some craft. Whatever it may be, there is always someone who wants to do everything, collect everything. Perhaps it's the guy who has to take a picture of every single statue in the museum, a lady who wants her kid to meet every overstuffed character at Disney World, or the guy at the supermarket who must consume every free sample on offer. Well, this is just the kind of greed that will come in handy to the exploratory tester.

For exploratory testers also collect things and strive for completeness. The *Collector's tour* suggests collecting *outputs* from the software; and the more one collects, the better. The idea behind this tour is to go everywhere you can and document (as my parents did on their state map) all the outputs you see. Make sure you see the software generate every output it can generate. For a word processor, you would make sure it can print, spell check, format text, and so on. You may create a document with every possible structure, table, chart, or graphic. For an online shopping site, you need to see a purchase from every department, credit card transactions that succeed and ones that fail, and so on and so forth. Every possible outcome needs to be pursued until you can claim that you have been everywhere, seen everything, and completed your collection.

This is such a large tour that it is often good to take it as a group activity. Divvy up the features among members of the group or assign certain outputs to specific individuals for collection. And when a new version of the application is ready for testing, one needs to throw away all the outputs for the features that have changed and restart the collection.

The Collector's tour is demonstrated by Nicole Haugen in Chapter 6.

The Lonely Businessman Tour

I have a friend (whom I won't name given the rather derogatory title of this tour) who travels a great deal on business. He has visited many of the world's great cities, but mostly sees the airport, the hotel, and the office. To remedy this situation, he has adopted the strategy of booking a hotel as far away from the office he's visiting as possible. He then walks, bikes, or takes a taxi to the office, forcing him to see some of the sites and get a flavor of the city.

Exploratory testers can perform a variety of this tour that can be very effective. The idea is to visit (and, of course, test) the feature that is furthest away from the application's starting point as possible. Which feature takes the most clicks to get to? Select that one, click your way to it, and test it. Which feature requires the most screens to be navigated before it does anything useful? Select it and test it. The idea is to travel as far through the application as possible before reaching your destination. Choose long paths over short paths. Choose the page that is the buried deepest within the application as your target.

You may even decide to execute the Garbage Collector's tour both on the way to such a destination and once you get where you are going.

The Supermodel Tour

For this tour, I want you to think superficially. Whatever you do, don't go beyond skin deep. This tour is not about function or substance; it's about looks and first impressions. Think of this as the cool tour that all the beautiful people take; not because the tour is meaningful or a great learning experience. On the contrary, you take this tour just to be *seen.*

Get the idea? During the *Supermodel tour,* the focus is not on functionality or real interaction. It's only on the interface. Take the tour and watch the interface elements. Do they look good? Do they render properly, and is the performance good? As you make changes, does the GUI refresh properly? Does it do so correctly or are there unsightly artifacts left on the screen? If the software is using color in a way to convey some meaning, is this done consistently? Are the GUI panels internally consistent with buttons and controls where you would expect them to be? Does the interface violate any conventions or standards?

Software that passes this test may still be buggy in many other ways, but just like the supermodel…it is going to look really good standing at your side.

The Supermodel tour is used extensively in Chapter 6. Along with the Landmark tour and the Intellectual's tour, it was used on every pilot project at Microsoft.

The TOGOF Tour

This tour is a play on the acronym for Buy One Get One Free—BOGOF—that is popular among shoppers. The term is more common in the United Kingdom than the United States, but in either case it is not just for groceries and cheap shoes anymore. The idea isn't for the exploratory tester to buy anything, but instead to *Test One Get One Free.*

The *TOGOF tour* is a simple tour designed only to test for multiple copies of the same application running simultaneously. Start the tour by running your application, then starting another copy, and then another. Now put them through their paces by using features that cause each application to do something in memory and something on the disk. Try using all the different copies to open the same file or have them all transmit data over the network simultaneously. Perhaps they will stumble over each other in some way or do something incorrect when they all try to read from and write to the same file.

Why is it a TOGOF? Well, if you find a bug in one copy, you've found a bug in all of them! David Gorena Elizondo demonstrates how he applied this tour to Visual Studio in Chapter 6.

The Scottish Pub Tour

My friend Adam Shostack (the author of *The New School of Information Security*) was visiting Amsterdam when he had a chance meeting with a group of Scottish tourists. (The kilts betrayed their nationality as much as their accents.) They were members of a pub-crawling troupe with international tastes. He joined them on a pub tour of the city that he readily concedes consisted of venues he would never have found without their guidance. The pubs ranged from small, seedy joints to community gathering places buried in neighborhoods more than a little off the beaten path.

How many such places exist in my own town, I wonder? There are many places that you can find only by word of mouth and meeting the right guide.

This tour applies specifically to large and complicated applications. Microsoft Office products fit this category. So do sites such as eBay, Amazon, and MSDN. There are places in those applications that you have to know about to find.

This isn't to say they receive hardly any usage, they are just hard to find. Adam tells stories of many of the pubs on his Scottish tour fairly heaving with people. The trick is in finding them.

But testers can't depend on chance meetings at hotels with kilt-wearing guides. We have to meet the guides where they are. This means finding and talking to user groups, reading industry blogs, and spending a great deal of time touring the depths of your application.

Tours Through the Hotel District

The hotel is a place of sanctuary for the tourist. It is a place to get away from the hustle and bustle of the vacation hotspots for a little rest and relaxation. It is also a place for the software tester to get away from the primary functionality and popular features and test some of the secondary and supporting functions that are often ignored or under-represented in a test plan.

The Rained-Out Tour

Once again, my selection of London as my tourist base pays off because sometimes even the best tours get rained out. If you've taken a pub tour in London between the autumn and spring months, it can be a wet, rainy affair, and you may well find yourself tempted to cut the tour short at the second stop. For the tourist, I do not recommend this tactic. You are already wet, and that's just going to ensure that the next pub feels even better than the last once you manage to actually get there. But for the exploratory tester, I highly recommend use of that cancel button.

The idea behind the *Rained-Out tour* is to start operations and stop them. We may enter information to search for flights on a travel website only to cancel them after the search begins. We may print a document only to cancel it before the document is complete. We will do the same thing for any feature that provides a cancel option or that takes longer than a few seconds to complete.

Exploratory testers must seek out the time-consuming operations that their application possesses to use this attack to its fullest. Search capabilities are the obvious example, and using terms that make for a longer search is a tactic that will make this tour a little easier. Also, every time a cancel button appears, click it. If there is no cancel button, try the Esc key or even the back