
A232
Limit of Visibility
REEF
BARRACUDA
BANK
fathoms
CHANNEL
315
641
719
814
982
612
597
270
378
308
394
225
155
215
24

Testing Without a Map

You don't always need to wait for complete specifications to start your testing effort.

BY MICHAEL BOLTON

SOMETIMES WHEN FACED WITH AN UNFAMILIAR APPLICATION and a directive to test it, it can feel as if you've been asked to chart a course through unknown waters. You're not sure what you'll find on the voyage, but you've got experience, some notions of what to look for, and an idea of where you'd like to go. If you see something surprising or unexpected, you'll take note of it. Those are exploratory skills, and sometimes they're all you need to begin.

Some testing advocates suggest that you should never test without a "complete written specification." That's unrealistic advice. First, there are *plenty* of contexts in which you may be asked to test without a formal specification: when you're evaluating a piece of commercial software to see if it suits your company's needs; when your organization's production code is so old and so heavily patched that the original specification would be meaningless—even if it could be found; or when work-

ing on Agile projects. Second, "completeness" is entirely dependent upon perspective and context. Even so-called "complete" specifications contain much that is implicit. In fact, the more explicit the document, the longer and more ponderous it is—and the less likely that someone will read it in its entirety. Finally, certain kinds of specifications might be supplied to you through some means other than a formal document—conversation, email, or your own inferences. A quick meeting with the boss, combined with your skills at identifying value, risks, and problems, might give you all the charter you need to begin supplying useful information to management.

An effective tester can always obtain valuable information by exploration, even if the sole purpose of exploring is to gather information for a more detailed test strategy.

Prepare for the Journey

When the explorers of old set sail for uncharted waters, they did not set out unequipped. They knew the sun and the stars, and they carried tools such as compasses, sextants, and clocks, not only for navigation but also for mapmaking. More importantly, they ventured out with extensive background knowledge and skills, which included deduced reckoning, celestial navigation, and horse sense. For exploratory testers, knowledge is often represented in two terms that you'll use as a testing expert: *oracles* and *heuristics*. An oracle is a principle or mechanism by which we can tell if the software is working according to someone's criteria; an oracle provides a right answer—according to somebody. A heuristic is a provisional and fallible guide by which we investigate or solve a problem; it's also a method by which learning takes place as a result of discoveries informed by exploration.

James Bach has given us a helpful set of oracle heuristics, to which I've added one, in the form of this mnemonic: HICCUPPS. The idea is that a product should be consistent with:

H*istory:* The feature's or function's current behavior should be consistent with its past behavior, assuming that there is no good reason for it to change. This heuristic is especially useful when testing a new version of an existing program.

I*mage:* The product's look and behavior should be consistent with an image that the development organization wants to project to its customers or to its internal users. A product that looks shoddy often *is* shoddy.

C*omparable products:* We may be able to use other products as a rough, de facto standard against which our own can be compared.

C*laims:* The product should behave the way some document, artifact, or person says it should. The claim might be made in a specification, a Help file, an advertisement, an email message, or a hallway conversation, and the person or agency making the claim has to carry some degree of authority to make the claim stick.

U*sers' expectations:* A feature or function should behave in a way that is consistent with our understanding of what users want, as well as with their reasonable expectations.

The **P***roduct itself:* The behavior of a given function should be consistent with the behavior of comparable functions or functional patterns within the same product unless there is a specific reason for it not to be consistent.

P*urpose:* The behavior of a feature, function, or product should be consistent with its apparent purpose.

S*tatutes:* The product should behave in compliance with legal or regulatory requirements.

Remember: Heuristics are guidelines, not edicts; they're fallible. They aren't universal—there are plenty of other ways by which we can decide whether a product is acceptable or unacceptable. There is some conceptual overlap between some of the points—but to an explorer, features of the new territory overlap, too.

Explore and Discover

Armed with these tools, let's imagine that I'm working for a small start-up, and that my company is going to be releasing its software on CDs. The company doesn't have a lot of money, and every dollar

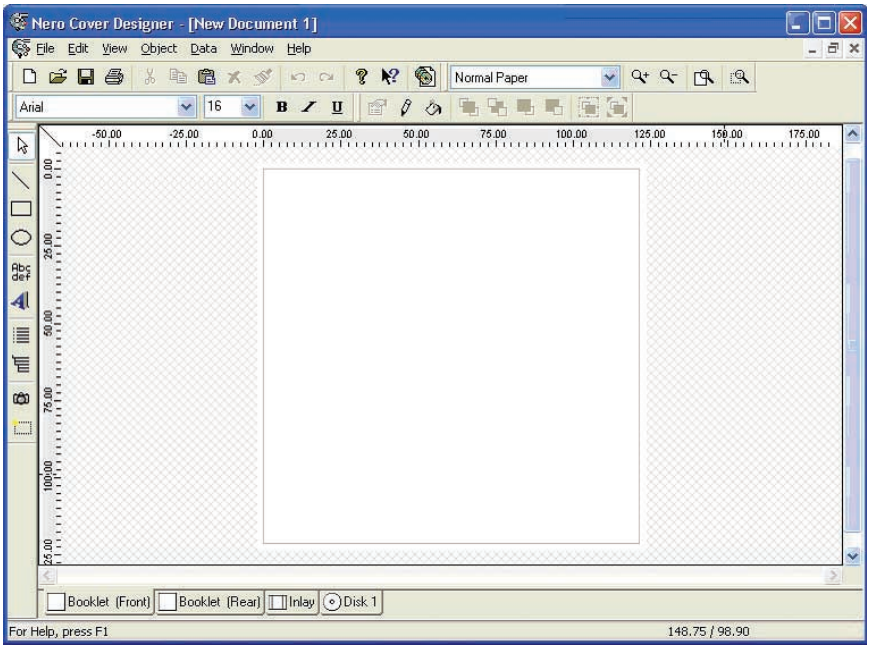


Figure 1: The main screen of Nero Cover Designer, shown immediately after startup.

counts, so my boss has asked me to evaluate the program that comes with the CD burner: the popular Nero CD recording software. Printing CD covers is a requirement, so she asks me to have a look at Nero's Cover Designer, a subset of the CD recording package, to see whether the company should use it. Instead of writing up an elaborate test plan, I'll just plunge in, quit when I have more information, and then (and only then) make some decisions about how to proceed. Figure 1 shows what I see on the main screen just after I start the program.

Yogi Berra was right: you can observe a lot just by looking. One of the first things I note is that there appears to be a default setting for the font: Arial for the face and 16 for the point size. My boss doesn't need to tell me to test fonts; I have the *consistency with purpose* heuristic in my head to tell me that, if the task is to print CD covers, graphics and text—and therefore fonts—are likely to be part of that task. Do I care about the accuracy of the point sizes and color depth of the graphics? Maybe, but I can ask about those things later, after I've run some other tests. I make a note to ask questions about accuracy and move on.

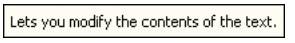
I'm going to need to put something on my CD cover, so I choose to insert a new object. I click Object, Insert, Text Box. Then I double-click the new object that appears, and the dialog shown in Figure 2 pops up.

Something already feels funny. In the new font properties area, the name of the font has disappeared and the point size now appears to be 8. In accordance with the *consistency within the product* heuristic, one would think that the font properties should be the same on both the main screen and the new dialog. Do we have our first bug? I'd say yes, but perhaps we should do some checking. I'll note it.

We don't have a specification, but Windows programs typically come with a Help file. A program should be *consistent with claims* it makes about itself, and the Help file is usually full of claims about what the program can do. So let's press the F1 key.

Why F1? Windows users have a heuristic that F1 should trigger the Help file, courtesy of the Windows User Interface Guidelines. (See this issue's StickyNotes for more information.) Cover Designer is a Windows program, and *a program's behavior should be consistent with programs like it*. If there is a compelling reason for your program to behave differently, then it might be worthwhile to depart from de facto UI standards. Otherwise, consistency with other products is a favor to the user, saving her the time and trouble of learning a different way of doing things.

When we press F1, a tooltip appears at the hot spot on the mouse pointer:



The tooltip says, "Lets you modify the contents of the text". Shouldn't that say, "Lets you modify the contents of the text *box*"? That might be a quibble, but in some contexts I'd be willing to call it a second bug. If this were my program, I might find the imprecise English a little embarrassing, which would violate the *consistency with image* heuristic: A program should be consistent with the image that a company wishes to present. And another thing: shouldn't F1 display the Help dialog instead of a tooltip? By Windows conventions, a tooltip should appear when you hover over an item with the mouse. I'll write a couple more notes about these Help issues; they might represent another bug or two.

I want to open the Help file. There's another way to do that—I can click the Help button to open it, click the Find tab, and find all the references to "text box." (See Figure 3.)

Hmmm . . . there's nothing here that looks like a reference to text. In fact, there's nothing here that seems to refer to *anything* in the Cover Designer. Let's go to the Index and look for the words "Cover Designer." All I see is Help for the Nero CD-ROM burning software. That means that either there is no Help for Cover Designer, or if there is a Help file, it's not coming up from inside Cover Designer. That's a problem based on the *consistency with user expectations* heuristic—a user could reasonably expect that a Help file summoned from within an application should be that application's Help file.

Well, it seems as though I'll have to give up on Help, and that's noteworthy. Let's return to the first presumed bug and do some more investigation. I don't know exactly what my company is going to put on the CD cover, but the specifics don't matter, so I'll put in some text that

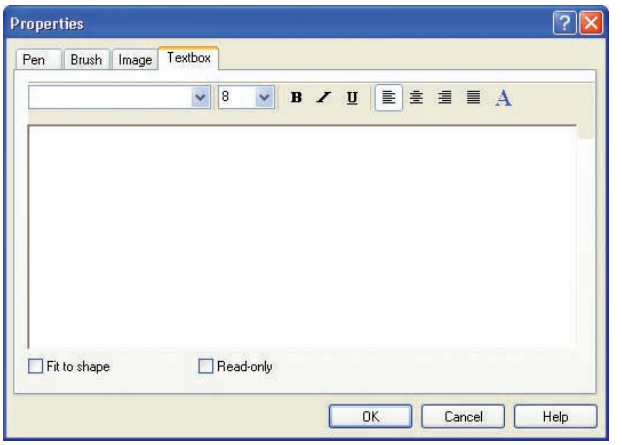


Figure 2: In the textbox properties box, the font name is missing. Could this be a bug?

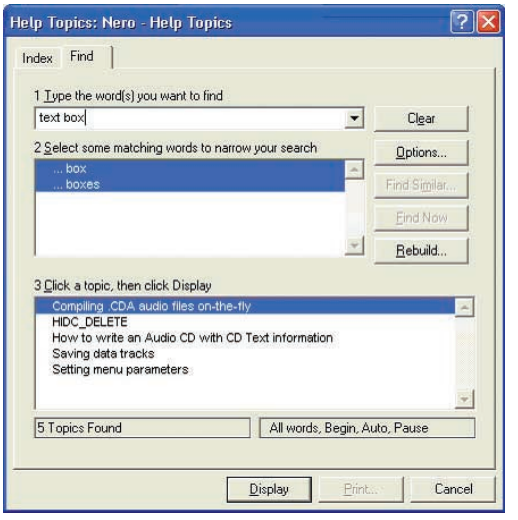


Figure 3: The Help dialog isn't helpful.

reflects the way that I might use the program. (See Figure 4.)

When I type "Beatles Compilation"—in fact, immediately after striking the B key—the font size of 8 turns to 16, and the formerly blank drop-down for the typeface is suddenly set to Arial. The *consistency with the user's reasonable expectations* heuristic suggests that typing some text should not change the selected font unless I've asked to do so. Even though this rectifies the problem I noted as the first bug, it does so in a way that gives me pause, and this is arguably yet another bug; I'll write that down. I'll highlight the text that I've entered and choose a different typeface and size; again, specifics don't matter. I'll select Comic Sans MS and 26 points. (See Figure 5.)

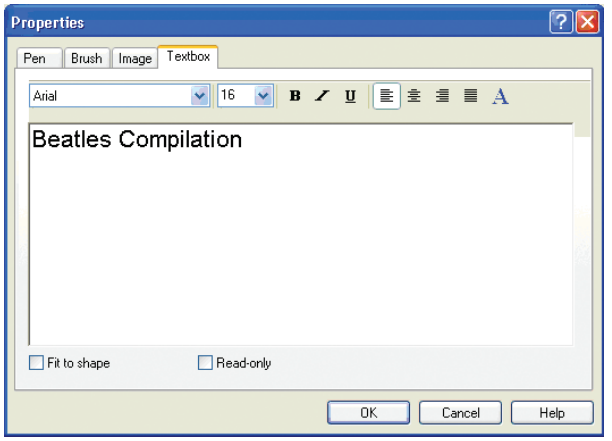


Figure 4: After typing in the text box, the font name appears.

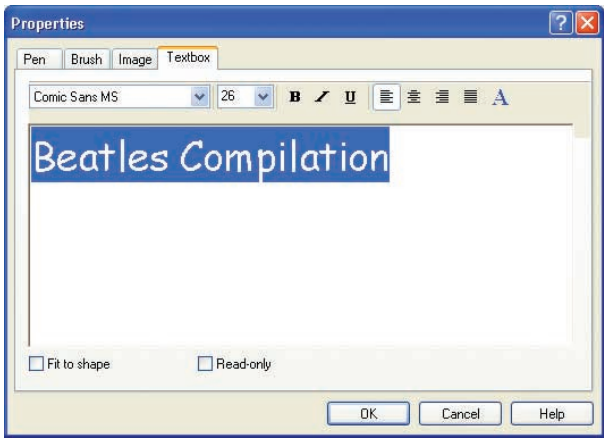


Figure 5: The text is highlighted, and the typeface and size are changed.

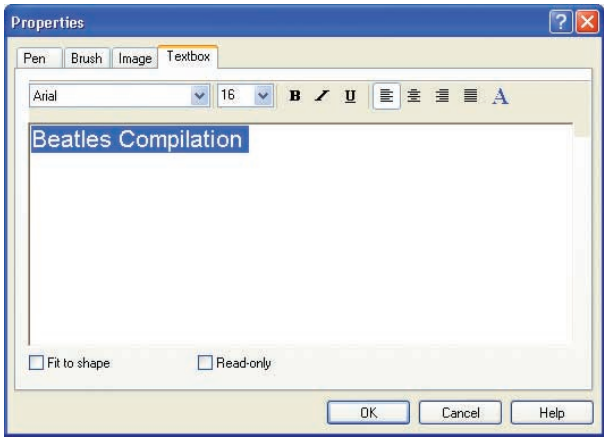


Figure 6: The typeface and size did not save with the text.

Then, I'll press OK to close the dialog. Now, I'll immediately click the text box to open the dialog again. (See Figure 6.)

Presto! The typeface is back to Arial, and the size is 16. This violates the *consistency with purpose* heuristic. Surely the purpose of pressing OK (rather than Cancel) on an object is to retain the properties that I've selected until I explicitly change them: *A feature or function should be consistent with its apparent purpose*. Another bug to note.

Note that in this dialog there are tabs for pen, brush, and image as well as text. I try this out, and I find that every time I try to reopen the text box to modify one of these attributes, the font information disappears—an inconvenience and an annoyance and, even without a specification, manifestly a bug. I'm disappointed because I seem to remember this feature working in a previous version of Nero Cover Designer. That's a violation of the *consistency with history* heuristic: A program should behave in a manner consistent with its own history or previous versions of the product.

The bugs in this program have been pretty easy to find, and this last one is so troublesome that I have some grave doubts about the rest of the program. After five minutes of testing, I'll be able to tell the boss that she should not rely on this product to produce the company's CD covers—and thank goodness I didn't waste time preparing an elaborate test plan based on some incomplete specification that some programmer apparently didn't read.

The Journey Ends

This was a particularly egregious example, but if you're still adamant that you need a written specification before you can begin testing, consider what you've just read in the context of two questions. First, did we need a written specification to provide important, credible, timely information to management? Second, would the cost of researching and preparing a specification—and waiting for it to be prepared—add significantly to the value of our report?

As you can see, in many contexts it's not only perfectly OK but also entirely desirable to test without using a specification. My background knowledge of GUIs on Windows helped me recognize several problems, and my ability to put myself in a user's shoes helped too. A few minutes of exploration, wandering through one feature of the program and looking through the spyglass of those exploratory testing heuristics, has helped me not only to find bugs but also to identify credibly *why* I think they're bugs, even though I had nothing like a complete, formal, written specification. Although I didn't have a map, I was certainly able to explore and compile one along the way. **{end}**

Michael Bolton lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries as part of James Bach's Rapid Software Testing course. Contact Michael at mb@developmentsense.com.