

Bug Report

INFO TO GO

- Bug reports should recommend a particular expected result.
- If at first you don't succeed in getting an application's behavior changed, use the documentation as a way of keeping the bug report alive—and maybe changing the developers' minds.
- Concentrate on the wording of error messages and documentation. Inconsistencies there may provide clues to design flaws.

But It's a Feature!

by Danny R. Faught

"WHEN I CLICK FILE→OPEN, THE APPLICATION crashes, erases my hard drive, and kicks my cat."

That's a compelling bug report. You probably don't need to refer to the manual that explains what the File→Open function does to make the point that harassing your pets is *not* one of the software's intended features. However, the everyday run-of-the-mill, low-severity bugs we usually deal with aren't this clear-cut. And the action the organization should take as a result of the bug reports we file isn't as clear-cut either.

Becoming a Bug Advocate

When you file a bug report, you become a bug advocate—it's your job to follow up to see that the bug is addressed one way or another. The way you write your report influences whether the application's behavior is changed, the documentation is changed to match the current behavior, or the bug is ignored. Many bug reports don't advocate a particular course of action and, as such, are easier to ignore. Consider this brief bug report:

"When I start the application from the command line with no options, I get the error 'bad filename' and the application aborts."

Does the tester who wrote that report want there to be a default action when no command-line options are specified? Or does the tester want the documentation to better explain why at least one command-line option must be given? Or does the tester just want a better error message? Maybe she doesn't care what the fix looks like, as long as the results are clear to the user and the behavior matches the documentation? Let's look at four ways the bug report above could be focused to improve its chances of being acted upon.

Please Change the Behavior. *"When I start the application from the command line with no options, I get the error 'bad filename' and the application aborts. Page 6 of the User Guide states that the application will use the default filename 'foo.dat,' so I expected the command to succeed with no options."*

Please Change the Documentation. *"When I start the application from the command line with no options, I get the error 'bad filename' and the*

application aborts. The User Guide does not clearly explain the command-line options: namely that the filename must be specified."

Please Change the Error Message. *"When I start the application from the command line with no options, I get the error 'bad filename' and the application aborts. An error message is expected, but it should be improved to better explain the situation, e.g., 'no filename specified.'"*

Please Change Either the Documentation or the Behavior. *"When I start the application from the command line with no options, I get the error 'bad filename' and the application aborts. This scenario is not documented in the User Guide, so it could be frustrating to the user."*

Leveraging Documentation

So what happens if they choose to ignore a bug that you feel needs to be fixed? I have a *trick*... er...I mean *technique* to convince developers to reconsider a bug. The idea is to focus on the documentation and the error messages.

For example, I recently filed a bug describing an error message from two different tools on an operating system that I'm testing for one of my clients: ps and sweep (see the sidebar "Getting Your Fix" for more details). The error didn't seem to indicate any malfunction that would affect the operation of the system. It also didn't affect mainstream users of the system, who wouldn't have access to the errant programs, though it would cause confusion for both internal and third-party developers who were using the system. There wasn't much of an opportunity to bring the documentation into play when I first reported the bug (but it did come into play later). So I reported it as a minor severity bug.

The developer, who worked for an outsourced vendor in a different location, replied with an explanation of why I was seeing the error, ending his response with "We don't consider it a problem."

I didn't want to understand the cause of the error message—I just wanted it fixed! So I pulled out one of my favorite bug advocate tools. I replied to the developer, "Please document this as expected behavior in the documentation for ps and sweep."

Two days later, the response from the developer was not a documentation change, but a code

change that removed the error. Victory! Perhaps the developer didn't like the idea of immortalizing this ugly behavior in print.

What Just Happened?

Of course, I didn't actually want a docu-

mentation change. If the developer had called my bluff and changed the documentation rather than the code, I would have marked that bug as fixed, since that admittedly would have been an improvement. Then, I would have filed another bug report complaining about the usability

of the system because of the spurious error, or else I would have filed an enhancement request. At this point, I would have expected that it could take a while to get the issue resolved, but at least I would have known that I had tried my best.

I should mention that this approach might not work if you don't already have good end-user documentation. I've spent a lot of time testing application programming interfaces (APIs), which tend to have detailed documentation intended specifically for developers. The developers I work with tend to write this API documentation in lieu of a functional specification, so there's additional motivation to keep it accurate. If you don't have end-user documentation yet, you could still find some place where the behavior in question might be mentioned—perhaps a design document or even the error messages themselves.

Getting Your Fix

When filing bugs, try to leverage documentation and exploit error messages to get your bugs fixed.

I filed the following "thread exited" bug report (several details have been sanitized). The bug was of low severity and the developer didn't consider it a problem. When I asked the developer to warn the user in the documentation that this error message might pop up, he chose to fix the behavior instead—exactly what I wanted.

Summary	"thread exited" error from ps and sweep after killing process
Severity	minor
Description	While trying to reproduce bug #385, I saw several errors like this from ps: <i>ps: read on 51844: thread exited.</i>

I mentioned this in the discussion of a stress-related bug a few months ago, and the developer wanted to explore it further, but we weren't able to reproduce it. This one is easy to reproduce.

1. Start the foo application and load the data file that's attached to this bug report.
2. Click "Login."
3. After a few seconds, click the "X" gadget to kill the foo application.
4. Run ps or sweep and you'll get the "thread exited" error.

It's complaining about the "Foosrv" process. Here's a snapshot of one such process before it disappears from the ps output and becomes an error:

51844 58143 testuser release 26K Foosrv

The error stops occurring after a short while, generally less than a minute. The taskview application does not issue an equivalent error when refreshing the process listing. There don't seem to be any detrimental effects from the error.

Making Them Think

For example, I filed bug 3639 against AbiWord, an open-source word processor (see the exact text of the main description of the bug in the "Triggering Change" sidebar). In an earlier bug report (bug 3619), I had complained that AbiWord couldn't import most types of html documents. The bug was closed as invalid because AbiWord expects xhtml files, so it can't import html format unless an extra plug-in is installed.

As a bug advocate, should I have been happy with that result? Of course not! I zeroed in on the misleading error message. As a user who wasn't aware of the html importer plug-in, I found this error message confusing:

AbiWord cannot open D:\foo.html. It appears to be a bogus or invalid document.

Because the exact wording was important to me, I filed bug 3639, in which I suggested a way to fix the error message (just a single word added):

AbiWord cannot open D:\foo.html. It appears to be a bogus or invalid xhtml document.

The first person who responded to bug 3639 said, "There may be issues regarding whether the html or xhtml importer is used to load a file which is detected as one or the other." Aha! That's what I wanted. When pondering the misleading error message, someone realized that the functionality of the software wasn't quite

Triggering Change

I filed the following AbiWord html import bug report. The developer replied that this was not considered a bug because a plug-in was required to import html files. I followed up with an error message change request. That request triggered discussion about design flaws among the developers—a step in the right direction! (You can also view this bug directly with all of the added comments at http://bugzilla.abisource.com/show_bug.cgi?id=3639.)

Summary	misleading error message when trying to open html file
ID	3639
Severity	minor
Description	This is a follow-up on bug 3619, which was closed as invalid. When I try to use the normal File→Open mechanism to open an html file (not xhtml) with a .html extension, I get this error: <i>AbiWord cannot open D:\foo.html. It appears to be a bogus or invalid document.</i>

I'm told that AbiWord expects xhtml here, but the error message doesn't say so. In fact, it is a valid html document. A small change to the error message could be a big improvement:

AbiWord cannot open D:\foo.html. It appears to be a bogus or invalid xhtml document.

right. I had spawned discussion that went beyond merely the wording of the error message.

The end result on this bug was, alas, that it was closed as a duplicate of yet another bug report, bug 1406. I'm not convinced that this was the right decision, so my adventures as a bug advocate will continue when 1406 is closed, because I'll need to verify that the fix for 1406 really does address my issue.

Advocating Advocacy

When you're writing up a bug report, be-

fore you send it, stop to think about whether there is any documentation that you should compare against the current behavior. Read the error messages carefully. Consider what the user will read as well as what the application will do. Be clear with developers about what kind of solution you want, if you have a preference. As a result, your bug reports will be more effective.

If you get a response from the developer that you don't like (such as, "But it's a feature!"), think again about whether the behavior is consistent with the writ-

ten cues. You may be able to present the bug in a different light and improve the chances of getting a fix. [STQE](#)

Danny R. Faught regularly argues in the court of bug appeals as a customer defendant. He is the proprietor of Tejas Software Consulting. You can reach him at faught@tejasconsulting.com and www.tejasconsulting.com.

**STQE magazine is produced by
Software Quality Engineering.**