Does a Bug Make a Noise When It Falls in the Forest?

Noel Nyman, Microsoft Corporation

You've probably heard the question about noise in the forest this way:

Does a tree falling in the forest make any noise if no one is there to hear it?

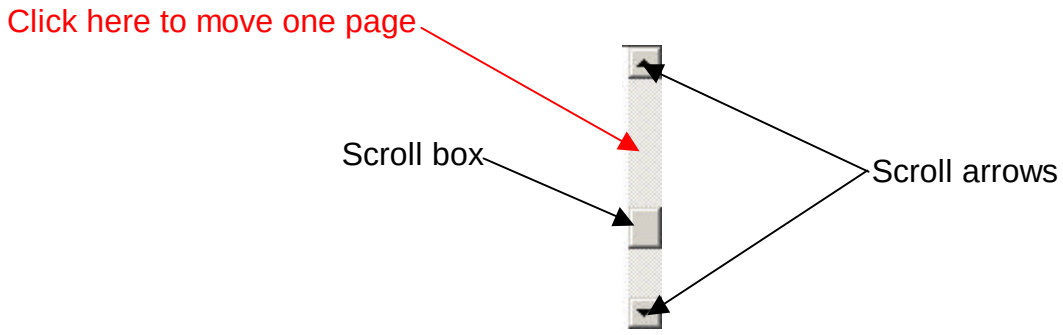
There are several ways to look at this. You could take a semantic approach and say it depends on how you define "noise." Or, you could look at the physics and set up equipment to measure the changes in air pressure that occur when the tree falls and humans aren't present. What makes this question interesting from a software quality assurance point of view is not the answer, but the fact that anyone thought to ask it in the first place. It implies that trees make noise (or not), or that noise in the forest is important (or not) only if there are people for whom tree noise is important. If no one ever visited this forest, it can be as quiet as outer space or a cacophony of creaking timber, and nobody will know...or care.

Rephrased in software bug terms, the question becomes:

Is a bug a bug if no user can ever make it happen?

Should we make any effort to find and report bugs that our users cannot reproduce? Using a specific example may help clarify the issue.

A human user can move a scroll bar in "page" increments instead of discrete steps by clicking the mouse on the space between the scroll box (the part you drag with the mouse to scroll) and a scroll arrow (the part at either end of the bar that you click with the mouse to scroll).



The team I'm part of at Microsoft uses many different companies' applications as test engines to find bugs in Microsoft Windows operating systems. As part of our automated tests, we force scroll bars to their minimum or "home" location by sending them SB_PAGEUP commands. (I'm simplifying here.) These are the same commands the operating system sends when the user clicks in the space between the scroll box and scroll arrow. Once the scroll box is moved all the

way up against the scroll arrow, we send some "extra" SB_PAGEUP commands for good measure. Most scroll bars can absorb the extra SB_PAGEUPs without complaint.

We recently found an application that freezes when the automated tests send the first "extra" SB_PAGEUP to a particular custom scroll bar. This is a classic "bug in the forest" situation. There's no way for a human user or tester to reproduce this bug. To crash the app's scroll bars you have to click between the scroll box and the scroll arrow when the scroll box is fully home. A human armed only with a mouse can't do that...there's nowhere to click!

Many of my testing colleagues think this bug should not be reported in the bug database. "It's an artifact of testing," they say. "If a human user can't reproduce a problem, the problem is not a bug." In tree terms, the falling tree is silent when no humans are in the forest, because only humans care about noise.

I disagree. How sure are we that there's no one there to hear that tree fall? Or, in software bug terms, is this bug really not reproducible by humans? Perhaps there are other ways to get that "extra" click into the system, things our clever users will try that we haven't thought of. Here are some ideas a tester might try:

- Will the Page Up key on the keyboard cause the problem? (In this application, the answer is "no.")
- Perhaps the click doesn't have to be in the scroll bar. Perhaps a click somewhere else will cause a crash. ("No" again, at least we couldn't find any such place to click during testing.)
- Can a user write a macro or script using the application itself to do what the testing tool does? In that case, a simple user scripting error might cause a crash. (Not in this application.)

Having failed to find a more reproducible case, should the tester still file a bug report? Yes.

Consider the tree analogy to illustrate. The tree really *did* fall. Why did that happen? Ignoring the noise part of the situation, do we have a "trees are falling in the forest" problem? In software terms, we've found one symptom of some underlying problem. What other symptoms might that problem have?

In this case, we *did* report the bug. When the vendor examined this custom scroll bar code, they discovered that the error was in a common code library. All the products in their application suite used that library and all similar controls in all of their applications had the same bug. The code fix itself was minor, but to properly test it, all controls in all applications in the suite would have to be tested again, and that would be very expensive. Worse, this suite has already been shipped. The only effective fix was a patch or new revision, with added marketing and delivery expenses. The bug was not fixed, and that was probably the best decision. At least it was an *informed* decision.

On most projects, only important bugs will get fixed. The standard for what's important varies with the project and with time. Bugs that were rated high priority when found early in the development cycle may be downgraded when the product must be shipped and there's only time left to fix the most critical problems. Every product ships with some known bugs. Selecting the

bugs *not* to fix is a compromise between high quality and market realities. We all ultimately produce what James Bach calls “Good Enough” software.

But no one can accurately decide that their software is “Good Enough” if they don’t know about all the bugs. The original decision that the suite was Good Enough to ship was *not* an informed decision because this bug was not in the bug database. It was unknown and unconsidered by program management and developers. If the bug had been reported in the bug database, developers might have recognized it as a symptom of a larger problem, and decided to fix it. Or, they might have scheduled it for a fix in the next release when it could be fully tested. Or, they might have *still* decided not to fix it. Regardless, they would have made the decision with all facts in hand.

As testers, we should investigate bugs thoroughly so we don’t give developers any false information. We should tell developers when we think the bug is something our customers aren’t likely to see. But we should never, never throw away bugs because we think they’re unimportant.

Bugs *do* make noise, regardless of where they fall. We should listen for that noise and enlist all the measurement and automation tools we can find to help us listen when we’re not in the woods. Every bug is important. Some bugs are more important than others. But we can’t figure out which are which until we look at each and every one.