THE UNSPOKEN REQUIREMENT: TESTING FOR CONSISTENCY

By **Michael Stahl** - July 17, 2017
https://www.stickyminds.com/article/unspoken-requirement-testing-consistency

It's easy to see that style consistency is important when discussing the user interface. But there are other areas where being consistent is just as important, even though they are not as visible. Consistency is one of the quality attributes of a product—any product—even if it is not stated clearly in the requirements documents, and testers have a responsibility to check for it.

About ten years ago I was testing the installation package for a Windows application. The installer was a standard one, similar to what we are all used to: a set of screens through which most users will usually click Next, Next, Next to install the software.

As long as I was using the mouse, everything was fine. But because I prefer to use the keyboard whenever possible, I tried to install without using a mouse. It turned out that even though the focus on each screen was on the Next button, hitting Enter did not activate the control.

I opened a bug report. I even quoted **Microsoft's forms controls documentation**:

"On any Windows Form, you can designate a Button control to be the accept button, also known as the default button. Whenever the user presses the ENTER key, the default button is clicked regardless of which other control on the form has the focus."

It turned out that I am not the only one who can quote Microsoft. The developer dealing with my bug report **responded in kind**:

"The spacebar activates the control with input focus, whereas the Enter key activates the default button."

If you think about it, you can see that the developer was right. On the other hand, I argued, the user expects Enter to work because this is how most other installers work. I was advocating that our program be consistent with the market, despite this implementation being technically incorrect.

Eventually I convinced the development manager that consistency is important enough to justify the fix and that it trumps the Microsoft documentation.

It is easy to agree that consistency is important when discussing the user interface or the way in which a product implements actions that are closely related (for example, having the program use the same file-explorer interface when using Save As and Open). But I think there are other parts of a software product where being consistent is just as important, even though they are not as visible as the user interface. Additionally, I see consistency as one of the product aspects that we, as testers, have a responsibility to check.

Here are some areas where consistency is important but is frequently overlooked.

APIs

System interfaces—especially those that are exposed externally, such as the system's published APIs—are a fertile area for consistency bugs.

If the application offers a rich API, it is almost guaranteed that the code for these APIs was written by a number of developers. Each developer has a somewhat different style and a different way of thinking about what is convenient, self-evident, and easy to use. This impacts the function names, the input and output variable names, and the order in which they appear in the function call. The result is that the consistency of the API package is degraded.

On top of this you will find simple mistakes that are the result of the developers knowing their code so well that they don't notice when some small detail can mislead a person who encounters the API for the first time.

Here are some examples I have witnessed:

- Three functions defined one after the other in a .h file. All use a list of input variables, one of which is a pointer to a specific structure. Two of the APIs have this variable named "descriptorId" but the third one uses "descriptor_Id"
- Signatures for functions that don't have any input variables. In some places the signature is functionName(void) and in other places functionName() .
- A variable name that is inconsistent with the variable type:
  SetArea(Mode area)
  "Mode" is the type of the variable called "area." So what's the input: mode or area? Is the function setting the area, the mode, or the mode in which area is used?
- A series of functions that get two similar variables and a different third one. The order of input variables in f3() call is inconsistent:
  f1(fileName, path, permission);
  f2(fileName, path, attributeList);
  f3(inString, path, fileName);

Functionally, all these bugs are cosmetic. The functions would work fine even if they weren't fixed. But they make the use of the software cumbersome and impacts the image of your company as the creator of quality software.

Assigning one person to be in charge of defining all the public API function names and signatures would eliminate most of the examples above. However, when a software package contains hundreds of functions, spread over multiple .h files, a single person will have a hard time keeping track of all the details and will likely miss some cases. This is where we can contribute.

Documentation

Part of our job as testers is to review the documentation released with the software and make sure it is accurate, including containing clear explanations, updated screen shots, and examples that actually work as advertised. We all know how frustrating it is when the user manual refers to a menu item that no longer exists, recommends a missing option, or gives a detailed example that just doesn't work.

We can all agree that accuracy is important. *Consistency* is another aspect that needs attention. There are questions you can ask yourself to discover **the most common inconsistencies** in your documentation:

- Are fonts and styles used in a consistent manner throughout the manual (e.g., does bold text always mean "this is a part of the command that must be used as is"?)?
- When using a specific term, is it always used in the same manner?
- Is the naming convention consistent (e.g., FileName and UserName versus FileName and user_name)?
- Are numbers styled the same way (e.g., using either words or digits)?
- Is there a period (or not) at the end of each sentence in a bulleted list?
- Are certain words always capitalized or lowercase?
- Is only English or American spelling used?

A word of caution: Proper documentation that is free of cosmetic problems is important and justifies investing effort to make it so, but if your time or resources are constrained and you need to give up either functional testing or proofing the documentation … I think it's obvious what's more important.

The advice given here is assuming you have a documentation testing task with some resources assigned to it—as you should! The involved testers should be aware that the testing effort should not only focus on correctness but also on consistency.

Test Documents

Unsurprisingly, it's not only developers who make consistency errors. Testers also should invest effort in being consistent, even when the documents are only used internally.

Years ago I tested Wi-Fi cards. One of the features of the product was the ability to turn off the card’s radio transmitter—the common name for this feature today is “airplane mode.” At the time, we called it differently—and inconsistently. I encountered various names in test documents: RF Kill, RF Disable, RF Off (with RF referring to radio frequency). When, during a test procedure, the transmitter needed to be turned off, the instructions could say Turn the RF Off, Set RF Kill to ON, Set the RF Kill to RF Disable, Enable RF Kill, etc.

This, of course, is confusing and increased the chance of a mistake while conducting the test. It is hard to read and even harder to understand a document that has inconsistencies.

Software engineer Karl Wiegers articulated this problem well in his book *More About Software Requirements: Thorny Issues and Practical Advice*. While the following was written about requirements, it is relevant to any document you write:

Too many requirements specifications use a random mix of different verbs: *shall, must, should, could, would, is recommended, is desirable, is requested, will, can, may,* and *might*. Many of these words are used interchangeably in casual conversation. The listener relies on context and the chance to ask for clarification to understand the speaker's point. But this can become confusing in a written specification. The reader is left to wonder whether there's a subtle but important distinction between these various keywords. Does *must* have some different connotation than *can*? Does *might* (which conveys a sense of possibility in normal dialog) mean the same thing as *may* (which conveys a sense of permission)?

Consistency is one of the quality attributes of a product—any product—even if it is not stated clearly in the requirements documents. Don’t let inconsistency stand in the way of making your product easy to use. Test more than just the user interface for consistency. Your customers will thank you.