

Response to Weisert review of C++ Common Knowledge

Stephen C. Dewhurst

Thank you for the opportunity to comment on your self-published review of *C++ Common Knowledge* before you proceed with your rather ominous intention to "...post shorter reviews on Amazon and other commercial web sites," thereby artificially lowering the book's current enthusiastic 5-star rating. I'll try to keep my comments as brief as the text of the book itself, but it won't be easy.

The review starts with a pre-title comment "Who's the intended audience?" One way to get this information is to examine the back cover of the book, where there are five bulleted items that describe the readers for whom the book was written. If you received your copy of the book without a cover, the same information is present in more detail in the introduction, and you should additionally know that neither the author nor publisher has received payment for the book.

For the purposes of the discussion that follows, I think the operative issue regarding the readers of the book is stated in the first bullet item, "You're no dummy, and you need to get quickly up to speed in intermediate to advanced C++." That is, the intent of the presentation is to confront the readers' ignorance while respecting their intelligence, and to value their time by not dithering.

Your review is short enough that it's practical to include it in its entirety here (with shaded background in order to distinguish it from my comments), with comments interspersed. I recognize that this is not standard practice, but like our readers, authors are also often pressed for time:

At first you might think this set of 63 articles ("items") is another collection of recommended programming practices, like the respected volumes by [Scott Myers](#) and [Joshua Bloch](#). On further examination, however, you note a fundamental difference. While Myers and Bloch emphasize *good programming practice* Mr. Dewhurst tells us mostly *how things work*. Unfortunately, he doesn't do that very well.

As the introduction explains, this book is intended to support books like Meyers', Sutter's, and Alexandrescu's. In fact, I've taught Scott Meyers' own courses based on his three "Effective" books scores of times over many years, and I continually run into the same issues of inadequate background knowledge in my students. Filling these pervasive knowledge gaps was one of the major motivations for my writing the book.

Strong start and short shrift

Item 1, "Data abstraction", made a strong favorable impression. It begins (p. 1) with a warning against an all-too-common bad practice:

"Never, ever, simply provide a bunch of get/set operations on the data members of [a class]."

Readers of this web site know that I've been campaigning against just that deplorable but growing [practice](#), so I was delighted by Mr. Dewhurst's concurrence, even if he didn't really explain it.

If, by "didn't really explain it" you mean I didn't blather on about the subject for several pages rather than simply state that get/set interfaces do not support abstraction as earlier described, you're right.

The titles of next three items are startlingly broad:

2. Polymorphism
3. Design Patterns
4. The Standard Template Library

raising the readers' expectations. Unfortunately, there's not much to those items. What can one say in three pages?

Actually, I'm rather proud of the item on polymorphism. In three short pages it gives the busy reader a solid underpinning on the concept of multiple types/interfaces to an object, is-a relationships, and design-by-contract in the context of a class hierarchy. It also has pointers to other items that provide additional technical details and implementation techniques. Your complaint here seems to be that the item is not long enough, but I see its brevity as a positive, inasmuch as we're not supposed to be selling books by the pound. Paradoxically, I labored long and hard to make sure that the book would be inexpensive as well as short (you can currently get it for less than \$20), so the price per pound is not unattractive either!

The items on design patterns and the STL are indeed too short. I said this in the introduction and in the items themselves. However, they are present for two very important reasons: First, they dispel some common misapprehensions that affect how programmers learn about and use patterns and the STL. Second, they emphasize the importance of these subjects and encourage the reader to learn about them.

Unexplained motivation

Mr. Dewhurst introduces highly specialized techniques with no explanation at all of why one would need or want to use them. For example:

1. The single-page article, Item 32, presents a technique for "Preventing Copying" of objects of a given class, but says nothing about the circumstances that make copy-inhibiting necessary or desirable.

One page was all that was needed, and in that one page I showed how the technique could be used to prevent copying in both assignment and initialization contexts. As an ancillary benefit, the reader is reminded that argument passing is an initialization context (another common problem is the assumption that argument passing involves assignment), that reference initialization does not involve copy construction, and many other issues. It's short, but it does the job, and there are many other items that take up issues related to copy operations. The upshot is that a reader of the book will have a good grounding in the purpose and mechanics of copy operations—including how to turn them off.

2. Item 36, "Class-Specific Memory Management" begins (p. 123):

"If you don't like the way standard **operator new** and **operator delete** are treating one of your class types, you don't have to stand for it."

but gives the reader no clue as to why one might not "like" the standard operations' behavior or what advantages a user-defined alternative might offer.

Actually, I think that I give the reader plenty of “clues” about why one might want to customize memory management. First, I use the example of a handle class of a handle/body pair (that had been seen earlier in the book). This is probably enough motivation for any reader; readers are not as clueless as most textbook authors make them out to be. They know that handle classes are small and that general purpose memory management will likely be suboptimal, and that's motivation enough for them to undertake the technical details that the item is really about.

However, I have noticed time and again that many of my students don't realize that use of the new and delete operators does not necessarily imply use of heap memory for allocation. Therefore the item ends with a complete example of use of member operator new and delete for allocating from a static block.

Given this, I really don't understand your comment.

Errors

The author notes in the preface (p. xii) the need for rigor:

"The author's ignorance of these complexities could result in an uninformed description that could mislead the reader ..."

leading us to expect the most careful attention to accuracy. But then:

- To illustrate a function with state memory, Mr. Dewhurst introduces the tired Fibonacci series example (item 18, P. 63), and then shows us this usage:
- ```
cout << "next two in series: " << fib()
 << ' ' << fib() << endl;
```

falling into the very trap that other writers take pains to warn against. Which call to **fib** in the same expression will your compiler evaluate first? Take your chances.

The Fibonacci example isn't "tired" it's "common knowledge" (read the introduction where I discuss the advantage of having shared, baseline examples for efficient and accurate communication).

That said, you're right. My invocation of the same member function twice without an intervening sequence point is bad programming practice that results in undefined behavior. I do have to say, though, that if that is the only real fault you could find in the book, a more collegial response would have been to submit a bug report for me to put in the book's errata page on semantics.org, rather than write what seems to me to be an unfairly panning review of a book whose motivation and structure you don't seem to understand.

All responsible authors take pains to avoid errors of this kind, but they inevitably creep in. This one made it past me (the guilty party), the expert reviewers, and two different C++ compilers. That's not an excuse. I'm just pointing out that responsible due diligence can't prevent all error; all the books you recommend highly elsewhere have long lists of errata.

- The curious reader will wonder about this function that appears to yield no result (item 6, p. 17):
- ```
void average( int ary[12] );
```

It's true that a better declaration of "average" would have had a return type. It does not, however, affect the validity or correctness of the example. (It's also possible that the function puts the average somewhere else, or that it modifies the array itself, but I'm not going to go there...)

- Near the end of Item 13 "Copy operations" (p. 47) Mr. Dewhurst offers this advice:

"It's often necessary for correctness, and occasionally more efficient, to perform a check for assignment to self, that is, ensure the left side (**this**) and right side . . . of the assignment have different addresses."

In fact, the accompanying code he recommends here is rarely if ever necessary or desirable, and may sometimes lead to a *less* efficient implementation. See [Assignment to Self Overkill](#).

Ahem. First of all, checking for assignment to "this" is a standard, idiomatic way to write copy assignment that you will find in virtually any competent C++ text written in

the past 15 years. Second, recall that this item first introduced the newer copy and swap idiom for copy assignment which doesn't require checking for assignment to self. Third, in this example I am not copying a string (as you do in "Assignment to Self Overkill") but the handle part of a handle/body implementation. Fourth, your argument in "Assignment to Self Overkill" is very highly specialized because it depends on specific aspects of a particular abstract type (a string) and a particular implementation of that type (the string maintains its length in a data member), and it allows no optimization for the case where one is assigning a shorter string to a longer one. Your string example is not "wrong," but it's not idiomatic, and I was discussing copy operation idioms.

Basically, your entire criticism of this item is based on the arguments of a one-page, un-refereed, minimally-supported, self-published article versus the accumulated idiomatic coding practice of the entire C++ community. I think it's fine to hold unusual technical opinions (I hold a few, myself), but I don't think it's fair to criticize a book on "common knowledge" because it doesn't ascribe to your unusual opinions.

Those are minor goofs, but once the reader's trust is undermined he or she is likely to be skeptical of the author's other examples, especially those with vague explanations.

That the problem with passive voice, isn't it? In your concluding statement it's not clear who is undermining the reader's trust. You've certainly made an effort, but I don't think it's a convincing one.