

**developer.\*****A Web Magazine for Software Developers**

## Consistency, Correctness, and Craftsmanship

By Daniel Read

The concept of “correctness,” as it relates to software, is usually discussed in relation to the known requirements for building or using a particular piece of software. When building software we ask questions like *Does this algorithm produce the correct result?* or *Does the workflow of this application correctly conform to the needs of the user?* This is one of the ways we measure the success of our efforts. We also judge correctness when using software tools and platforms, whether to build software or to accomplish other types of tasks. If we don’t use a tool correctly, then we likely won’t get the result we want. For example, if the syntax of the code is not correct, then the compiler will not compile it.

However, in this essay, I will be using the term *correctness* in a different way. I would like to discuss correctness as it relates to convention and personal preference. For example, how do you prefer to name variables? What size should a button be on a GUI form? Where should buttons be placed on a GUI form? How should you design compound keys in a database schema? What error handling technique should you use? Should implicit meaning be attached to the `null` value in database columns? For these types of questions, each of us over time develops our own concepts of correctness.

Sometimes these concepts of correctness come from conventions we have encountered in books or on previous projects. For example, for data and function naming we can choose from the Hungarian convention, camelCase, and PascalCase, to name just a few. These are examples of conventions, which are presumably agreed upon by some group of people, and are often formalized in some way. Concepts of correctness also arise out of pure personal preference. For example, maybe you really hate typing underscore ( `_` ) characters, so you don’t use them when naming database tables or named constants. Often, the distinction between convention and preference is unclear, since preferences often mutate from established conventions, and vice versa.

Correctness in this sense, then, is quite relative. Different people in different situations at different times will have different ideas about what is “correct.” As individuals, when we create software, or when we look at other people’s software, we are constantly making judgments as to this relative correctness. We judge things like technique, look and feel, format, and style. We compare what we see against what we perceive to be the prevailing convention and against our own preferences—or against our preferred convention, prevailing or not.

### Making Changes

Judging someone else’s code according to our own relative correctness measurements is one thing, but what happens when we need to modify that code? At that point, our (often

unconscious) concepts of relative correctness can work for or against us. This is the point where we must weigh correctness against consistency. When it comes to convention and personal preference, consistency has a higher value than correctness. At a certain point, our desire for what we believe to be correct becomes a liability. If we selfishly cling to our conventions and preferences, we can taint the things we change by creating inconsistency and incongruence. We might make a change that is correct in terms of the requirements, and correct in terms of our own conventions and preferences, but which creates disharmony because of a lack of consistency with what's already there.

Let's look at a quick example to illustrate what I'm talking about.

Bill is a developer of business software who has always worked with Oracle brand databases running on Unix systems. He recently joined a new team that is using Microsoft's SQL Server database. As a first task, Bill's boss has assigned him to add a new feature to an application that is already in production. This change will require the addition of two columns to an existing table. After receiving the requirements, he adds the two new columns. He names the first column `DT_PURCHASE_DATE` and the second column `IND_IS_FIRST_ORDER`. He updates all the appropriate places in the system code to work with these new columns. He tests his new feature, and it all works great. Everything he has done matches the customer's requirements exactly. When it goes to production, the users are thrilled with the new feature.

There is one problem, however: all of the other columns and tables throughout this database use a totally different naming convention than the one Bill used for these two new columns. In this particular database, all object names use PascalCase with no underscores. Furthermore, no prefixes like `DT_` (which Bill uses for date columns) or `IND_` (which he uses for Boolean "indicator" columns) are used anywhere in the database structure. Bill should have used names such as `PurchaseDate` and `IsFirstOrder` (assuming, that is, that these two names are consistent with the way that date and Boolean columns are named in this database). Now these new columns stick out on the E/R diagram like a cockroach on a wedding cake.

So what happened here? Why did Bill do this? If we want to give Bill the benefit of the doubt, we could say he was simply not paying attention. We could assume that consistency is not a priority to him. He did things the way he is accustomed to doing them—the way he considers to be "correct."

If we do not want to give Bill the benefit of the doubt, we could say either that he is lazy or that he does not care. If we want to say he is lazy, then we'd have to acknowledge that he knows better, but did not want to go to the trouble of changing his normal way of doing things. But if you follow this logic through, you have to say that he is lazy *and* that he does not care. In order for him to decide to be lazy, he'd have to notice that he was creating inconsistency. This does not speak well of Bill, for if he knowingly used an inconsistent naming convention, then it appears that he puts his own preferences above the good of the database, application, team, and company. We expect more from a professional.

The trouble for Bill is that people on Bill's team who encounter Bill's work on this new feature are going to think less of him. Even if they give him the benefit of the doubt and conclude that he was simply not paying attention, that still is going to lower their estimation of Bill. If Bill continues this behavior in more serious matters, he may find himself out the door, or relegated to meaningless tasks so that he can do less harm. Here's another thought: maybe this first simple feature that Bill was assigned was a test to see how he would do with it. When the team leader or review team comes behind him to see how he did, this behavior will raise a red flag.

## The Cost of Inconsistency

Now, you might be saying at this point, "Come on, Dan, lighten up. All Bill did was use a different naming convention for a couple columns. What's the big deal?"

To this I would say that the principle illustrated in this seemingly trivial example is critical to our measure of ourselves as professionals. A professional goes out of his or her way to maintain consistency when making changes. However harmless a couple of inconsistently named columns may seem, it represents a real problem. If these are the first two inconsistently named columns in the database, and no one catches this in a review process, and the changes go to production, they may stay there forever.

These inconsistently named columns are a problem for two reasons. The first reason is simple aesthetics. When we create software, the aesthetic value of what we create is important not only because it is pleasurable to look upon, but because it adds real value to what we are building (see [The Principle of Aesthetics at developerdotstar.com](http://ThePrincipleofAestheticsatdeveloperdotstar.com)). Which leads to the second reason: now that the E/R diagram is polluted, it's a whole lot easier for other developers to add more of their own pollution, and those people probably won't be consistent with Bill's inconsistency, so now we've got multi-faceted inconsistency. Before long we've got a total mess.

I live in a condominium building. We have a common trash dumpster in the back of our building so that residents of the building can throw their trash away. We keep the dumpster locked because if we don't, the other buildings around ours will fill the dumpster with trash, taking away all the space in the dumpster for the people in our building, who are paying for the dumpster. Whenever I see that one of our considerate neighbors has thrown their bags of trash and boxes and junk on the ground around the dumpster, I pick it up and put it in the dumpster. Why?

Because if I leave it there, then it sends a message to all the people in the neighboring buildings that it's okay for them to throw their trash on the ground around our dumpster. I have seen this happen over and over, which is why I started picking up the trash as soon as I see it. If the trash stays there, then other people will start piling their trash all around it. If I pick it up right away, then weeks will pass between incidents of people throwing their trash around our dumpster. Trash attracts trash. Inconsistency attracts more inconsistency. (Authors Andy Hunt and Dave Thomas write about a similar principle they call "Fix Broken Windows" in their book *The Pragmatic Programmer*.)

Going back to the database example, as the inconsistency piles up, the understandability of the database degrades. The barrier of entry for someone new coming to the database goes higher and higher. The likelihood of mistakes goes up, because the inconsistencies make it harder to make assumptions.

For the project on which I am currently working, just about zero attention was paid to consistency for the code and user interface for the first year and a half of the project (this was before I joined the project). What came out as a result is a damn mess. Some user interface forms have buttons on the bottom right, some in the bottom center. Some of the buttons are big, some are small. Some forms have an `OK` button, others have a `Save` button, while still others have an `Apply` button. Within the code, naming conventions are all over the place, as are data access techniques, class designs, etc., etc. Without conscious and conscientious attention to consistency by everyone involved, inconsistency is the inevitable result.

## Areas of Concern

So where is the line? When do you decide to make a stand and say “Consistency be damned, this is just the wrong way to do this.”? What do you do when you encounter a project like the one I just talked about, one that is already a mess of inconsistency? How do you try to achieve consistency when such a goal seems hopeless? These are tough questions to answer, and I don’t know that we can formulate absolute guidelines. Using our best judgment, these kinds of dilemmas need to be handled differently based on the circumstances. There are probably several factors influencing the outcome, including at the least:

- What the is the future viability of the project/product? If it is definite that the product will be killed or retired in the near future, then maybe consistency is not all that important.
- How much time and resources are available to put effort into clean-up and refactoring activities so that some consistency baseline can be reached?
- What politics are at play in and around the project?
- How much latitude does the management give to the members of the team to implement quality initiatives, which are often viewed by management as non-value-adding?
- What emotional investment do the existing team members have in what they’ve already created, even if what they’ve created is a mess?

When the way is unclear, the key is this: by paying attention and asking explicit questions about consistency, and by giving real thought to the answers, we are more likely to create quality and add value, even our actions result in a little inconsistency. However, if we don’t pay attention, and just barrel in and do it our own way, inconsistency is the likely result, and worse, inconsistency for no good reason at all.

Before wrapping up this part of the discussion, I'd like to briefly list a few areas where consistency often gets neglected:

**Adding a new form/window to an existing GUI application.** Inconsistency creeps into user interfaces all the time. When multiple developers are working on a GUI application, if there is not a published GUI standard, and if those developers are not consciously making an attempt to preserve consistency, then different parts of the GUI will end up looking and acting different. Even when there is not a published standard, a developer has a responsibility to take a look at what is already there and do her best to match the look, feel, and behavior.

**Adding new code to an existing application.** Consistency here applies to things such as data and function naming, style, layout, technique, etc. Even when you hate the naming conventions being used in the app (It's no secret that developers get very attached to their naming conventions.), you have a responsibility to use the existing naming conventions. The same goes for module and routine comment blocks, indenting styles, error handling conventions, data access techniques, etc.

**Coming to a new language, or switching between different languages.** This is a pet peeve of mine, and could probably be the subject of a whole essay. Too many developers try to impose the conventions, styles, and techniques of one language onto another language. Often the developer will carry the conventions and techniques from his first language for years, trying to impose what he learned about his first language onto every other language he encounters subsequently.

Every language has its own "ways of doing things." Even the same language from platform to platform, context to context, will have unique conventions. In my opinion, when you learn a new language, you have a responsibility to not only learn the syntax of the language, but to also learn the common conventions that people use when they write code in that language. You can do this by reading books, magazines, web sites, and of course, code. For mature languages, you might find different camps of people with ideas and conventions that clash. This is normal. Apply your best judgment, exercise your own personal preferences, and code away.

**Writing documentation.** If you are creating documentation for a project/product, then do your best to make that documentation look like the rest of the documentation already produced for that project. Are there document templates or published standards? Try to find out. If you find that there are none, maybe you can be the one to create some templates and standards.

**Process consistency.** If some process is clearly already being followed, make an effort to follow it, even if you have not been fully schooled in it. This is especially important when it comes to change management processes. If you have questions, ask them. Don't just roll on and do it your own way, because all you are going to do is create work for someone else.

**Joining a new team.** This is an all-encompassing item. When you join a new team, make an effort to learn what they are doing and try to do things in the same way. The members of the team will welcome you more easily if they see you are paying them the courtesy of trying to learn their customs, standards, and conventions. Granted, an attentive team leader will explicitly introduce you to this information, but that often does not happen when teams are working hard to ship software. So even if you are a little annoyed that no one took the time to give you the information, seek it out for yourself. If you have concerns that something the team is doing is wrong or ineffective, then tactfully bring that up in a constructive way. However, don't be surprised when the team leaders decide to not change what they are doing, even if they agree that your advice is sound. Inertia is a powerful force.

**Starting a brand new project.** If you are starting a whole new project from scratch, it makes sense that you would have free reign to create a whole new set of standards and conventions. You should absolutely do this, since, as I already mentioned, if you don't make an explicit effort towards consistency, inconsistency will inevitably result. In addition to published standards, it's a great idea to have review processes in place so that inconsistency will not slip through the cracks. However, there might be concerns about consistency between projects at the organizational level. If you are building software for an organization that has multiple systems already in place or in progress, it's a good idea to check for standards and conventions already in use within the organization and adopt those for your new project.

## The Craftsmanship Connection

There are a few keys to preserving consistency: first, you must decide that you care about consistency and that it is your responsibility to do your best to maintain it; second, you must pay attention to "the little details" in all aspects of your work (David Pye calls this "the habit of taking care" (1)); and third, you must possess the humility to subjugate your own concepts of correctness to the higher aim of consistency. Responsibility, pride, and attentiveness (and a little humility) is what software craftsmanship is all about.

In his book, *Software Craftsmanship: The New Imperative*, Pete McBreen writes:

"Software craftsmanship is all about putting responsibility and pride back into the software development process....[it] is a mindset and an attitude, rather than a body of knowledge." (2)

## References

(1) David Pye, *The Nature and Art of Workmanship*, (Bethel, Connecticut: Cambium Press (Cambridge University Press), 1971, 1998), p 52

(2) Pete McBreen, *Software Craftsmanship: The New Imperative*, (Boston, Massachusetts: Addison-Wesley, 2001), pp 52, 81

###

Daniel Read is editor and publisher of the **developer.\*** web magazine. He lives in Atlanta, GA, where he works as a software architect. He is currently at work on a book about software development crafted for a business audience.