

Re: Do you have a Knowledge Officer?

## Re: Do you have a Knowledge Officer?

---

*Source:* <http://coding.derkeiler.com/Archive/Cobol/comp.lang.cobol/2007-10/msg00295.html>

---

- *From:* Robert <no@xxxxxx>
  - *Date:* Fri, 05 Oct 2007 21:21:29 -0500
- 

On Fri, 05 Oct 2007 23:22:45 GMT, "William M. Klein" <wmklein@xxxxxxxxxxxxxxxxxxxx> wrote:

"Pete Dashwood" <dashwood@xxxxxxxxxxxxxxxxxxxxxxxx> wrote in message  
[news:5mntnsFefolrU1@xxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:5mntnsFefolrU1@xxxxxxxxxxxxxxxxxxxxxxxx)

<docdwarf@xxxxxx> wrote in message  
[news:fe5uol\\$K5b\\$1@xxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:fe5uol$K5b$1@xxxxxxxxxxxxxxxxxxxxxxxx)

In article <BMuNi.3425\$bM6.496@xxxxxxxxxxxx>, tlmfru  
<lacey@xxxxxx> wrote:

[snip]

One man's experience is never adequate.

Oh, I *\*cannot\** resist...

... seems like I've seen different attitudes manifested, say, by  
Mr  
Wagner.

Why pick on Robert? You yourself have manifested such an attitude, as,  
indeed,  
have I.

We all relate to our own experience; there's nothing wrong with that.

Pete.

—

"I used to write COBOL...now I can do anything."

## Re: Do you have a Knowledge Officer?

Pete,

It seems to me that you, DD, I, and MOST of us USUALLY qualify our statements with phrases such as "in my experience" or "from what I have seen". This is in stark opposition to how RW usually (not always) makes his statements. I believe that he has even stated that something along the lines of "of course I am expressing my opinion or my experience" and seems to expect us to "infer" these qualifications.

It is common for CLC postings to state a conclusion, unsupported by facts/premises and the canons of logic, much less citations. In those cases, added qualifiers doesn't change the opinions' essence, they just make the writer sound like a bureaucrat.

I thought it was axiomatic, taken for granted, that everything posted here is the author's opinion, unless (s) he supports it with verifiable facts and credible logic.

--- Quotations ---

Weasel words are almost always intended to deceive or draw attention from something the speaker doesn't want emphasized, rather than being the inadvertent result of the speaker's or writer's poor but honest attempt at description.

Generalization by means of grammatical quantifiers (few, many, people, etc.), as well as the passive voice ("it has been decided") are also part of weasel wording. Generalization in this way helps speakers or writers disappear in the crowd and thus disown responsibility for what they have said.

[http://en.wikipedia.org/wiki/Weasel\\_word](http://en.wikipedia.org/wiki/Weasel_word)

Weasel-Words Rip My Flesh! Spotting a bogus trend story on Page One of today's New York Times.

By Jack Shafer

Posted Tuesday, Sept. 20, 2005, at 6:38 PM ET

How many "many's" are too many for one news story?

Like its fellow weasel-words?some, few, often, seems, likely, more?many serves writers who haven't found the data to support their argument. A light splash of weasel-words in a news story is acceptable if only because journalism is not an exact science and deadlines must be observed. But when a reporter pours a whole jug of weasel-words into a piece, as Louise Story does on Page One of today's (Sept. 20) New York Times in "Many Women at Elite Colleges Set Career Path to Motherhood," she needlessly exposes one of the trade's best-kept secrets for all to see. She deserves a week in the stockades. And her editor deserves a month.

Story uses the particularly useful weasel-word "many" 12 times?including once in the headline?to illustrate the emerging trend of Ivy League-class women who attend top schools but have no intention of assuming the careers they prepared for.

She informs readers that "many of these women" being groomed for the occupational elite

Re: Do you have a Knowledge Officer?

## Re: Do you have a Knowledge Officer?

"say that is not what they want." She repeats the weasel-word three more times in the next two paragraphs and returns to it whenever she needs to express impressive quantity but has no real numbers. ....None of these many's quantify anything. You could as easily substitute the word some for every many and not gain or lose any information. Or substitute the word few and lose only the wind in Story's sails. By fudging the available facts with weasel-words, Story makes a flaccid concept stand up?as long as nobody examines it closely.

<http://www.slate.com/id/2126636/>

Developer Weasel Words  
Thursday, June 21, 2007

As a development manager or project manager, you here a lot of weasel words and excuses from your staff or from external consultants who are trying to hose you into believing things are better than they are. In many cases, developers use these phrases to even convince themselves that things are better than they are, resulting in chronic late delivery and poor quality.

So beware the following phrases from your teams:

- \* It should work: this usually means that it doesn't. It also means that it was probably not tested properly as the result is current undetermined. The word "should" should be taken out every developer's vocabulary – it either does or it doesn't.
- \* I just need: beware of that word "just". Its a belittling word meant to make things smaller than they are. Just take the phrase "I just need to write this component" to be "I need to write this component" and already the magnitude of the work involved grows. Developers tend to be chronic under-estimators and the use of the word "just" is a sign of that mentality.
- \* Almost done: this is also a weasel word. When a developer tells you things are "almost done" ask for the specific tasks that are left over immediately. In addition, keep in mind that projects do not progress linearly – the last 10% is always about 40–50% of the work of the total project. I've seen projects that are chronically late be "almost done" for 3 months.
- \* It was tested: this also usually means it that wasn't tested properly. Ask for a test plan and the specific tests that were done. If the developer cannot produce these with sufficient evidence that PROVES that it was tested, then it wasn't tested.
- \* It must be an environment/configuration/deployment problem: this may be actually true, but it usually points to a larger stability problem. If you cannot build and deploy reliably then why would you have confidence that the code works?
- \* If things go smoothly: this I hear a lot, e.g. if we don't hit any snags then we can be done by Friday. Guess what – you're likelihood of hitting a "snag" by next Friday is probably high and given the lack of risk based management, the team has probably got no mitigation or contingency strategy. Then next week, you'll hear the next phrase in our list, "Yes, it could have been done if it weren't for that Snag we had".
- \* Yes, but, as in "Yes it can be done, but": this means it cannot be done. Tell your staff to just come clean and say, "No it cannot be done". Another variant on this is, "Yes it could have been done, if it weren't for marketing, requirements, technical risk, etc." This simply shows that your developers work in an idealistic world where things never go wrong.
- \* We'll make up the time at the end: if you're already late by the end of

Re: Do you have a Knowledge Officer?

## Re: Do you have a Knowledge Officer?

requirements, you're likely going to be even later by the end if you simply keep going on the same track. In my experience, teams don't dramatically faster as they hit their stride. Even if there is some efficiency, its nowhere enough to make up for lost time.

\* I've got it done, but I need to build a few more components for you to be able to see it: then its not done! Encourage show and tells, code reviews, unit tests, etc. so that code is visible as soon as possible. Use slicing models so that you can see pieces of the application in weeks, not months. In addition, code that isn't checked in should never be counted – that means that someone cannot build it sufficiently to share it. It only counts if you can verify it. Ideally, its not "done" unless there are sufficient unit tests, a build script, and a document that someone walking into the code repository could check out the project run a build and have all the unit tests pass. Then the code is done – anything less is mythical.

\* I've got it done – I just need to integrate it: The word "Integrate" is a big weasel word. Think of a web service that adds two numbers together. The algorithm is one line of code. But the integration work is huge. In addition, integration usually means the first time that disparate teams are bringing code together which is always cause for issues. Don't under-estimate the integration, especially in today's world of distributing computing, web services, etc.

\* It Worked on my Machine!: programmers use this excuse to downplay a bug. The reality is actually the opposite – it means that you have an intermittent bug which is by far the worst kind of bug to have in your application. You want bugs to fail quickly and consistently – any variant such as "That's Weird", "That didn't happen yesterday", "That must be a data problem", etc. is admitting you have a bug that cannot be easily duplicated.

My recommendations to reduce the amount of excuse making from your team:

\* Encourage a culture of honesty and team work: You get these excuses when developers are hiding things, and sometimes this is because you've created a culture that encourages hiding because you don't want an honest answer.

\* Be ruthless with your quality and talent standards: don't excuse poor talent, bad management or chronic late delivery. If you create a culture where talent isn't rewarded the bar isn't kept high then you'll be excusing the team to continually strive for excellence.

\* Expect more than just code: measure performance based on estimation, quality, delivery and team work as well as pure code quality. If you have a developer who produces great code but cannot deliver on time then that's not a great developer.

\* Increase visibility and shrink delivery cycles: if you have to show your work on a constant basis and deliver on 2 week iteration cycles, your excuses tend to go away. You either deliver or you get found out pretty quickly. Use show and tells, code reviews and continuous integration to see what people are doing on a constant basis.

\* Don't give half credit for 50% done – its either done or not done: If your tasks cannot be managed this way, then you should split up your tasks until you can work this way.

\* Establish what "Done" really means: for example, at a minimum "Done" should mean checked into source code control and able to build into the current branch. If you're doing Test Driven Development, it should also mean all tests run successfully. If you have specific performance criteria, then its not "Done" until the performance tests pass.

\* Use counting techniques to measure wherever possible: this is a great suggestion from McConnell's book on estimation. The more you can count in units, the more accurate you estimation. So if you can count the number of pages, web services, objects, databases,

## Re: Do you have a Knowledge Officer?

tables, stored procedures, tasks, etc. that are left to accomplish then you can measure them more easily than if its a big blob of work. If your requirements aren't well defined enough to count objects, e.g. you don't know how many web pages you're building in your web site, then you're really not in a position to estimate your ship date.

\* Don't sucker, manipulate or bully your team: If as a project manager, you resort to traditional management tactics such as playing games, being political, establishing a blame culture, or bullying your team you'll lose your credibility and simply encourage lying. A tortured prisoner will tell you anything you want to hear – the same goes with development teams.

If you have a project that operates in the open, has a culture of honesty and establishes a high performance bar, you'll find that peer pressure as well as some overall guidance will get risks, problems and bugs out in the open. If when you discover these problems people work as a team to fix them instead of blaming each other then every problem solved becomes a victory and not a blame opportunity. You'll get better answers and improved morale on the team as you set clearer performance expectations.

<http://chriswoodill.blogspot.com/2007/06/developer-weasel-words.html>