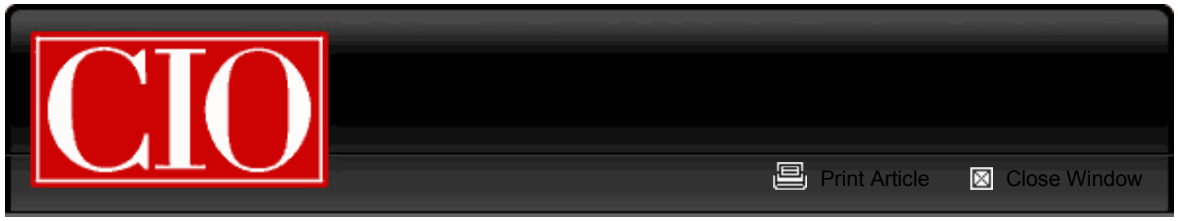

From: www.cio.com

How To Document in the Cloud

– David Taber, CIO

July 26, 2011

There's never been a good excuse for not commenting code or documenting systems, even though these practices are every bit as helpful to cost and quality as test-driven development. But let's face it, documenting systems isn't exactly macho and besides, it doesn't give you the ugly job- security of unmaintainable code (take a look at this page for some terrific worst-practice suggestions).

Working in the clouds, unfortunately, can give developers a new excuse: "I don't have any access or control over that (public-cloud) service, so I can't document anything about it."

Uh-huh. So, call their bluff this way: all *consumers* of a (public or private) cloud service need to document what the service does, how it's used, and what behaviors (including bugs and error conditions) that the consuming side uses or depends on. Immediately, the whine will come: "but that means we have to document the same cloud service in every module we use it." To which your response should be, 'Put your documentation info in a Wiki, a Google (GOOG) doc, or other collaborative system that the entire development and integration team can access."

This pushes everyone toward the best practice for the external interfaces and dependencies of modules and services: a centralized repository whose content is created and maintained by every member of the development team — as distributed as they may be. In my experience, the alternative of having a centralized team responsible for creating and maintaining data dictionaries, entity-relationship models, business-process descriptions, or interface documentation produces only two things: an excuse for not doing documentation or, at best, obsolete or often incorrect tomes.

But is a Wiki the best place to document the internals of a module or the administrivia of a variable? It's not a bad idea, but most of the time developers just won't be in the documentation area while they're hacking code or tweaking a variable inside an app. When they're working in their code, in-line commenting must not just be encouraged: it must be measured. For the code we develop, we don't allow any module to be checked in to the system unless at least 10% of the lines are explicit comments, and another 10% of the lines have in-line remarks. The inline code requirement goes double for test modules, as it can be difficult to guess what a particular routing is testing for, and where to look in the

module under test when there is a failure. Your code integration system may already have counters and enforcement mechanisms for this, but if they don't you can typically script it to do so.

In some Cloud apps and frameworks there are scripting, workflow, and formula languages that don't natively support comments, but there's always a way to plant comments via with inoperative code fragments containing the documentation message (e.g., IF 0=1, "The comment goes here — maybe as long as 80 characters").

So we've removed all the excuses for not commenting code. In modern Cloud based applications, though, much of the action is isn't in code: it's declarative programming and custom fields/objects /relations. Here, too, document-as-you-go must be explicitly measured with incentives for both the developer and administrator. In systems like Salesforce.com, every new field has two opportunities for self-documentation: the description area, and the help-bubble information area. I encourage both to be used, with different information in each. If a cloud system doesn't have this, we use dummy fields with names that shadow the real field, adding metadata information in a human-readable way. For example, for the numerical field "salesteam" we'd add a text field "♦salesteam" and set the default value to descriptive text. If your system doesn't support extended-ASCII in the field name, use a punctuation mark like { or ¡ that forces the variable name to come at the end of any alphabetized list. If your system doesn't support default values for text fields, you'll have to embed as much intelligence as you can into the dummy field name.

What about self-documenting the messages that go between modules? With WSDL or other XML dialects, you can put lots of metadata into the XML itself, or into supporting DTDs. While these can be verbose and arguably slow system response times, most of the time the extra overhead is barely noticeable. When you're making a call to the Cloud directly from your code, the libraries that actually form the REST or SOAP messages typically won't let you control much of what goes over the wire. In this case, sending an extra information-only text variable or two in the outbound request will make both troubleshooting and learning curves a lot easier in the long run.

The bottom line: even though the cloud doesn't force any documentation or make it any easier to enforce, there are clever ways to use cloud app features and Web service protocols to make modern systems more self-documenting. Call it sky-writing, and enforce it in your next cloud project.

David Taber is the author of the new Prentice Hall book, "Salesforce.com Secrets of Success" and is the CEO of SalesLogistix, a certified Salesforce.com consultancy focused on business process improvement through use of CRM systems. SalesLogistix clients are in North America, Europe, Israel, and India, and David has over 25 years experience in high tech, including 10 years at the VP level or above.

Follow everything from CIO.com on Twitter @CIOonline.