

Re: Infinite Loops and Explicit Exits

Source: <http://coding.derkeiler.com/Archive/Cobol/comp.lang.cobol/2004-12/0434.html>

From: Robert Wagner (*spamblocker-robort_at_wagner.net*)

Date: 12/06/04

Date: Mon, 06 Dec 2004 21:21:08 GMT

On Mon, 6 Dec 2004 23:10:03 +1300, "Pete Dashwood"
<dashwood@enternet.co.nz> wrote:

>
> "Robert Wagner" <spamblocker-robort@wagner.net> wrote in message
> news:g237r0hcilmnrmsqhi94i57up4ov2b9cjp@4ax.com...
>> On Mon, 6 Dec 2004 01:41:27 +1300, "Pete Dashwood"
>> <dashwood@enternet.co.nz> wrote:

>> The *PERFORM ... THRU* we're discussing looks like this:
>>
>> *PERFORM some-paragraph*
>> *THRU some-paragraph-exit.*
>>
>> *some-paragraph.*
>>
>> *some-paragraph-exit.*
>> *exit.*
>>
>OK, so what exactly is your point?

I just wanted to make sure we're talking about the same thing.

>> > it is certainly true that *SECTION* can adequately
>> > describe (and document as a descriptive header) a piece of related,
>> > contiguous code, that implements a given function.
>>
>> A flower-boxed comment works better. It's not limited to 30
>> characters.
>>
>Neither is a flower boxed comment referenceable from somewhere else in the
>program.

That's what paragraph names are for. Appending the word 'section'
doesn't add any information.

>Perform calculate-pay
>(it is a fair bet to any English speaking programmer that *THIS* section will

>calculate how much someone should be paid. There is no requirement to go and
>lookup a flower boxed comment somewhere down the bottom of the program...
>that would be the comment that was never updated when the paragraph was
>changed... <G>)

Tax calculation is very often done in third-party code that's out of your control. Suppose the entry point is named VERTEX-ITX. You might want to document the call with a flower-boxed comment.

Suppose your shop has an ancient mainframe naming standard that says programs must be named ssbbbnP (just in case we have to go back to VSE). You think I'm making this up, don't you?

>Note that if this was an Object Method (and I wrote it) it would contain
>Properties (PUBLIC) for the employee ID and the current pay field. All the
>other fields it needed (derived database access keys, foreign keys,
>relations, tables, semapores, calc fields, etc.) would be PRIVATE. It is a
>black box where you simply give it the employee you want paid and it returns
>the amount to pay him. All of his various rates would be accessed from
>databases, along with the hours or part hours for each rate. Tax and
>deductions, FICA, pension fund, whatever, would all be found on databases
>and manipulated by this one Object Method using rules based processing,
>implemented by soft rules, that can be changed outside the object.

OO is not required to do that. For decades, people have done it with called programs, remote procedure calls, etc.

>I would not expect to ever maintain this code again, once it was written,
>and, as the interface to it would never change either (linkage was not used
>to pass data in and out), there would be no impact on code that invokes it
>if other innate features of it were activated later.
>
>In effect, the whole payroll system would be dependent on proper database
>design. The behaviour could only be changed by changing the database. I
>believe that is a good thing.

I design large systems the same way. For instance, at aforementioned supermarket company, a frequent activity was determining the cost of a product on a given date. I completely re-engineered the tables containing that information without touching, recompiling or re-testing the many programs that called for cost. Before the Big Rewrite, dozens of programs would have needed substantial logic changes.

>In fact, I am coming to the conclusion that there may be a good case for
>designing proper databases and using stored procedures on them, to implement
>the system, with virtually NO application code at all.

Stored procedures ARE application code.

>*The team I last
>managed tried to persuade me in this direction, and they were very clever
>people who had spent their whole professional lives working on this kind of
>technology for COMPAQ. Guys in their 30s with 15 years experience, all on
>Database driven systems <G>. For COBOL this is not a good solution ..*

Sure it is. On Oracle, at least, stored procedures can be written in any language, including Cobol. The technical term is External Procedures. They work just like stored procedures, with an entry in a Library table pointing to a .dll or .so.

>.. *and I
>still believe that embedded SQL should see the DB as purely a repository.
>with no application code embedded in the database. But OO solutions may well
>use things that COBOL is not good with, and there could be a case for
>Extended SQL stored procedures and triggers, in conjunction with transaction
>processing, and no COBOL (or ONLY minimal OO COBOL...)*

No Cobol?! Are you trying to put us out of work? :)