

Re: Something has to be tested and maintained was Re: GoTo in Java

Re: Something has to be tested and maintained was Re: GoTo in Java

Source: <http://coding.derkeiler.com/Archive/Cobol/comp.lang.cobol/2006-10/msg00272.html>

- *From:* "Pete Dashwood" <dashwood@xxxxxxxxxxxxxxx>
 - *Date:* Mon, 30 Oct 2006 17:21:07 +1300
-

"Clark Morris" <cfmtech@xxxxxxxx> wrote in message news:3gtuj2hgm7roiraii0526t4avd5p7t6a8h@xxxxxxxx

My comments on the following from months back are top posted with nothing interspersed. As someone who is a modifier and reworker of existing systems, I won't really learn or understand Java or OO until I work with it which at my age may be never. However, I strongly believe that we in the field have failed to understand that scripts are code, control parameters be they sort control statements or table entries in a package are code and in fact anything used to control the manipulation of data is code.

They are indeed. Congratulations on your insight.

The systems that you describe in several of your postings scare me in many ways.

Not sure who 'you' is in the above sentence, but, in case it is me, I'll respond quickly...

Being scared by new approaches is a perfectly normal reaction; must of us are anxious when we move out of our comfort zones. I am not an Evangelist, preaching anything or trying to convert the masses. I'm not on commission, not trying to flog any particular software, and I really don't care any more whether people here take up what I'm saying.

For a decade now I have been advocating that COBOL people expand their skill sets, embrace OO, get to understand component technology, look again at traditional development methodology, make themselves more marketable and lever their COBOL knowledge into the 21st Century. Some have; most haven't. I feel sorry when I see people here watching their jobs disappearing overseas. But, it was predictable and it didn't happen overnight.

What I post here is based entirely on my own perceptions and experiences. It

Re: Something has to be tested and maintained was Re: GoTo in Java

Re: Something has to be tested and maintained was Re: GoTo in Java

works for me. I try to explain why it works, in the hope that others can benefit, but I am finally coming to the conclusion that most of what I'm saying is simply being perceived through a fog of pre-judgement and measurement against 'what we've always done', rather than open-minded assessment and experiment.

I am on record as saying that maintenance of source code is self defeating. Instead of trying to grasp what I'm getting at (and I've spelled it out several times here) it has been easier for people to say: "Well, we've ALWAYS maintained source code. That's what COBOL is good for. How could you possibly maintain a system if you DIDN'T change the source code?" (the last sentence is rhetorical because 'everybody knows' maintenance is impossible without changing code...)

And yet, I have built systems that use functional encapsulation and object orientation to ensure that their methods (behaviours) and properties (public and private data) DON'T need to be changed. Some of these systems have been live for over 5 years now and have had functionality added and amended WITHOUT changing existing source code, and WITHOUT having to regression test every module or program when new functionality is added.

It CAN be done; it just can't be done if you continue the same approaches that have been used for the last forty years.

If you decide to change one room of a brick house do you grind up the existing bricks and make them into new bricks? Why would you do that? The existing bricks are fine, they just need to be organised differently and maybe some new ones added. Maybe you'll have some bricks left over from the previous design. Perhaps you can use these to build a barbeque... The point is that a brick is a brick; it has the attributes and behaviours of a brick. There is no point in trying to change an existing brick into a corner brick (although you CAN do so and some companies do this; I believe it defeats the purpose of using bricks in the first place). And once a brick is made, it remains a brick forever. Don't try and sharpen it or round it; make sharper or rounder bricks for that. (You will find it is easy to modify the existing mould for this)

I hope that whatever tool or tool set was used to create these applications with just drag and drop combining of components also produced the documentation to say what was done so that when the business changes some poor soul has a fighting chance of figuring out where to make the change(s) to what was done.

Why is it always 'the poor soul' at 3 in the morning who has do maintenance? :-) Doesn't maintenance ever get done in working hours?

Re: Something has to be tested and maintained was Re: GoTo in Java

Re: Something has to be tested and maintained was Re: GoTo in Java

The tool sets I referred to DO generate documentation and they do it graphically and on-line. You can instantly see the structure of any given project and the relationships between the various objects. (You may have seen the diagrams generated by MS ACCESS if you click the 'Relationships' option... the principle is similar.)

Just coming back to the environment you are used to for a minute, when was the last time you went to a filing cabinet and retrieved a program listing so you could look at what it did? Not recently...? No, we all learned pretty quickly that what was filed with the documentation was pretty useless for maintaining the program. The current source stored in the system is the ONLY definitive documentation that is worth a damn. It is exactly the same with objects.

The processes I use for development ensure that the designs are documented as they are developed, and the system documents the objects, noting each method, property, and event, and the relationships between components. Help files for each Class are also built. There is no reason why an OO system should be any less 'documented' than a non-OO one, but the emphasis should be on USEFUL documentation, rather than documentation for the sake of having something in a filing cabinet.

I also hope that the development tools contain adequate means to handle the needed testing and not just unit testing.

Naw... debug it in live... that's my motto... :-)

The use of control parameters, control tables, etc. is powerful and allows the use of generalized packages such as operating system of choice or affliction, SAP or Oracle Financials, word processing packages, etc.. While some of these are bounded, others especially in house systems have no checking on them. Older portions of operating systems can be equally vulnerable to loose change mechanisms as I can testify from experience on IBM's MVS. I have nothing against languages and tools that operate at higher levels of abstraction. At least in theory they can be expressing things in ways that are closer to the description of the process to be implemented. The use of documented components is also a good thing and something that I am used to as a former systems programmer although not with some of the niceties of the OO paradigm. If these tools also are more self-documenting, especially to non-programmers, that too is something I applaud.

Re: Something has to be tested and maintained was Re: GoTo in Java

Re: Something has to be tested and maintained was Re: GoTo in Java

So why does it scare you?

However the idea of something being unchanging and never needing upgrade is not in keeping with the world as I know it.

Absolutely. The observable universe is apparently dynamic. No less so when it comes to business...

The question is not whether things need to change; they do. The question is HOW to manage those changes.

Personally, I don't think that altering procedural code is an ideal way to do it. The act of altering it introduces more chance of error. Just because that was the only option for several decades doesn't mean that it is right. Now we have alternative choices. We can encapsulate functionality into fundamental units (components). They do what they do. Ask a radio engineer about the last time he modified a given pcb or chip... He didn't. Everything is modular and encapsulated... even the integrated stuff...input, process, output... component technology analogised at an engineering level. (Software engineering could do well to look at electrical engineering...)

Determining

the pervasiveness of a change and whether it should affect all processing is something that probably has not been well thought in the past. I am used to a mainframe environment where a subroutine change is picked up by all users (COBOL programs compiled with the DYNAM option binding CALLED routines at execution time rather than link time if they are not nested programs) upon implementation. This has implications for retrospective processing.

Yes, it does. Even though it is probably the best approach to managing that particular environment.

The advantage is that an organization wide change takes effect immediately. The disadvantage is that a changed routine is immediately available to all callers. OO does not change the underlying dilemma.

It depends on how you implement the OO. It CAN change the underlying dilemma if you implement it so that there are no changes to any existing interfaces.

Re: Something has to be tested and maintained was Re: GoTo in Java

Re: Something has to be tested and maintained was Re: GoTo in Java

No changed parameter lists; everything calls what it always did, in exactly the way that it always did. The new functionality is simply a new method call.

Although invoked objects CAN have parameters passed to them, I don't personally do this. Instead, I use GET and SET on the properties before (and after) invoking the method I want.

It would take too long to go into the subtle details of why this is important, here, but the main benefit is that if interfaces don't change, then regression testing can be avoided. New functionality is turned on by new properties being set; the same method works exactly as it did before, for the cases where these properties are NOT set... Previous code executes exactly as it did previously (it has not been amended), but new code can use the same method to achieve a different result by setting properties before it is invoked. The new code can be added as a new method that wraps the existing method so that code and functionality is not duplicated. The details and examples are more than I am prepared to post here, but I'll gladly help anyone who's interested, if they contact me.

The question is whether you want the invoker/caller to pick up a changed version when it issues the initial invoke/call for an execution unit or if you want to use a new name or parameter to make any changed behaviour explicitly requested.

Exactly. See above.

Some further comments after the interesting discussion on code translation below...

On Sat, 28 Jan 2006 12:35:21 +1300, "Pete Dashwood"
<dashwood@xxxxxxxxxxxxxxxx> wrote:

"Richard" <riplin@xxxxxxxxxxxxxxxx> wrote in message
news:1138390683.396793.322920@xx

Someone else then translates it to C++. Then I come back, and notice there's a C++ version now. I tell my IDE to translate it to Java, and get something not too bad. I can clean it up

Re: Something has to be tested and maintained was Re: GoTo in Java

Re: Something has to be tested and maintained was Re: GoTo in Java

a
bit, and add it to the docs too.

While that mechanism may be a nice toy for learning different languages it would be useless for production code. Who is going to ensure that your 'cleaned up' Java now works the same as the Fortran code. If a change is made to the C++ how does the Java get the same change.

In many examples what is the Java equivalent of 'for record in file:' ?

It is several lines of class methods and a while loop. If the Java is modified then how could it ever get back to being just the one line ?

Source code translations have been tried for several decades and they just don't work. For example you cannot translate C code into Java and expect it to magically become object orientated.

It's a bit like saying: "Don't vote; the Government will get in..."

Procedural source is procedural source and it really doesn't matter what language you translate it into. The problem is not with the language, it is with the paradigm.

I've been following this thread with some interest and I understand what Oliver is trying to do. It is a noble idea, but I think Richard has hit the nail on the head in this (and previous) posts.

The effort involved in source translation, and the risks, do not match the rewards.

The point that is being missed entirely here is that the real solution is to NOT maintain source code. (In any language...)

Because of its COBOL and procedural programming roots, this forum is excited by language discussions and the problems attached to source languages. Yet

Re: Something has to be tested and maintained was Re: GoTo in Java

Re: Something has to be tested and maintained was Re: GoTo in Java

I see people every day, with no knowledge of ANY programming language (apart from scripting, which is not maintained in the traditional sense of code maintenance) producing really impressive applications that run on the web, access databases, share information, and produce reports.

Component based development means you write code once and do NOT maintain it. (I know there are many places that don't adhere to this and they try and maintain the components. There is only marginal gain over a normal source code maintaining shop if you do this.) The idea is to decompose functionality to lower levels than has traditionally been the case, write flexible generalised solutions to these low level functions then leave them alone! All you have to do is build the components into applications. You should never change a component once it is written and its methods, events and properties are established. If you need to 'enhance' it write a wrapper that interfaces to it and extends it for that specific case, or write a new component.

The true benefits of code reuse are not gained if you keep rewriting the code. Encapsulate functionality and reuse it. Do it once; do it right.

Can't believe I wrote that...:-) (It was a NZ Government slogan many years ago, probably buried in my subconscious...nevertheless, I stand by what I wrote above.)

Fortunately, modern environments are so component rich (even though many people do not realise the thousands of components that are provided free with your OS) that people are starting to simply plug things together and achieve impressive results. Scripts do not require thousands of lines of procedural code to achieve what WOULD require thousands of lines of procedural code, and maintenance is not a problem.

The rule of source code is over. The future belongs to object code. It will be developed (once :-)) in whatever language is most appropriate for it and things like ILs and CLR will level the playing field. I have one application where components written in VB, C++, and PowerCOBOL all play together nicely on a single windows screen. The user is unaware of what the

Re: Something has to be tested and maintained was Re: GoTo in Java

Re: Something has to be tested and maintained was Re: GoTo in Java

functionality is written in (and doesn't care), and I have NEVER changed these components in any way. (I couldn't if I wanted to, because I don't have the source for some of them...) This application has been live for several years, has had enhancements made to it, and has had some minor changes to its functionality. New screens have been added and some of the components mentioned appear on the new screens as well (there is only ever one copy of them in the system). The less source code you have to look after, the less risk to your investment.

Maintaining source code is yesterday's technology.

Having said that, I do realise that 'today' we have a huge investment in procedural source that HAS to be maintained so I don't think this discussion is pointless...:-)

But I also think people need to start thinking 'beyond the square'. Today's kids are doing it. In 50 years the idea of sitting down and 'programming' a computer line by line, will be a quaint curiosity, like us using flint tools or speaking Latin in the Supermarket.

OK, that's really everything I need/want to say about this... :-)

Disagree with me if you want (and I know many here do) but that is what I think, and I honestly believe the next breakthrough will be 'smart components' that will come with pre-programmed concepts... But that's another story, and the sun is shining here and I don't have to work so I'm off to the beach... :-)

Pete.

.