

Re: mfc pitfalls

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2007-02/msg00116.html>

- *From:* Joseph M. Newcomer <newcomer@xxxxxxxxxxxxx>
 - *Date:* Thu, 01 Feb 2007 22:46:26 -0500
-

I'm not sure what "new way" requires callbacks; in fact, callbacks are usually considered a bad idea. virtual methods usually work better than callbacks for most designs, and ActiveX (for all my complaints about it as a Web technology) was designed to eliminate the need for callbacks—as—callbacks using instead a more rigorous notion of event notification.

I would actually be scared of any project that thought callbacks were a way of life. The result is usually an unmanageable mess. Languages like Ada recognized this and there was no syntax in the design for function pointers.

I'm curious why you avoid threads. I wouldn't know how to write an interesting system without them. What good is having an n-way multiprocessor if your only mechanism is to use a single thread. When everyone else has their multiple oxen pulling their carts, being vastly more efficient, getting everything done faster, the single-threaded approach is clearly a dead-end paradigm, and should be considered only in rare and simple cases. Otherwise, I get no benefit from spending money on concurrency. Sure, I can run more apps without major interference between them, but suppose I only have *one* app that matters? I want it to get maximum throughput.

As I tell my students, "if you need to do something long, use a thread. If you need to do lots of things, use many threads." There are design criteria, such as if the maximum number of feasible threads is larger than the number of processors, then you can actually lose efficiency, but if the threads can block, then you need more threads than processors. Multithreading also simplifies a lot of architecture, and in fact eliminates the need for callbacks in some cases. I would never use callback I/O for asynchronous I/O, for example. It is far too complex. But I'll spin off a thread without a second thought, just to simplify the logic and keep the GUI alive.

So I fail to see how good design precludes threading; on the contrary, I believe modern good design *exploits* concurrency. The art is to do so as simply as possible. So the "positive-handoff protocol" is, for example, better than using synchronization. Queues are good models for interthread communication. "The best synchronization is NO synchronization" is one of my design principles: that is, do not design threads that require a lot of synchronization; make them independent entities that touch each other as little as possible with synchronization primitives. Synchronization is required where threads touch. And like mechanical systems, where things touch, they generate heat and waste energy. Queues turn out to form virtually friction-free mechanisms for these situations, especially if you know how to do them efficiently. You can do VERY good OO design and use threads at practically zero cost.

Re: mfc pitfalls

Given you have only one thread to do everything, how do you keep the GUI alive? PeekMessage? Alternatively, you piece out the work and handle the little pieces one at a time during the idle time? Do you at least use fibers? Shades of Win16! I certainly don't want to waste a lot of effort producing code I stopped writing around 1993 (that's fourteen years ago!), that runs with the same performance as fourteen years ago except scaled by the processor speed. Why is having a single thread be the bottleneck to all performance a good design methodology? If I'm running on an 8-way multiprocessor, I want to use as much computing power as I can get my hands on to minimize the turnaround time on a transaction. I'm not going to sacrifice the quality of the user experience to some silly design principle of single-threading.
joe

On Fri, 02 Feb 2007 01:31:20 GMT, "jmarc" <jmarc@xxxxxxxxxxxxxxxxxxxxxxxx> wrote:

Just fews comments..

– I didn't say you should write down a specs document to give to your outsource. Talking in few words may help a lot to get what you want.

– Data flow (flowcharts) is one of the UML diagrams defined. Indeed, it not the most used. We should use only diagrams that are usefull to be elaborated, in the goal to explain the design we have done to the next programmer that will work with the source code.

UML goal is also to elaborate a right and powerful design of a small or a very huge applications, as the implementation will go fast and with minimum error. (design should comes before implementation.. right?)

What really matter are data structure diagrams to show the relationships of the information. This is ultimately the only thing that matters....

Just fine!! And UML is the right way to describe relationships!

UML and OO coding are not religious. I just see them as tools..! I also write programs for long time, but not as much as you... I begin in 1973, and the biggest workstation application I work on in 1995 had 2 and half billion lines of code, using over 60 different services on the server. But sure, many of us had used to much cut and paste in that code. and bunch of very discrete bugs to find out in the tests phase...

– Now, programming in OO environment requires new

Re: mfc pitfalls

thinking of the design. The new way of programming fill a very huge place to Callbacks (every reimplemented func are just those), that it become impossible to follow the code like we done in the past with a just fine Structured app.

OO programming doesn't prone using cut and paste aproach in the code (despite template that I tried to avoid). And over a year and half, I average over 110 lines of code per day, efficient in the final release, comprising tested time and bugfree.. .. and full comments along in the code as well !

– Now, using OO techniques and a good desing, I also able to avoid the use of threads in my applications. Whatever I have to manage communications with many services at same time, and keeping GUI still operating!

jmarc..

"Joseph M. Newcomer" <newcomer@xxxxxxxxxxxx> wrote in message news:r114s2ppl9gc4e2rcaocrr5gannlj8qbd6@xxxxxxxxxxx

On Thu, 01 Feb 2007 19:32:52 GMT, "jmarc" <jmarc@xxxxxxxxxxxxxxxxxxxxxxxx> wrote:

There are many styles of programming. Each may vary a lot from one to another, but not because one or the other is badly implemented.

But there are practices that are just wrong. You should see some of the outsourced device drivers I've had to review! Pure crap, written by people who should not be allowed to program, let alone deliver products. So there ARE bad styles of programming, and they are very obviously bad.

You should have defined the programming style you want, or those you'l be easy with afterward, before contracting the outsource.

Writing a specification of what is forbidden (e.g., my "n Habits of Highly

Re: mfc pitfalls

Defective

Windows Programs" might be a good start) is also important.

A source code may be very well implemented,
but hard to be understood by someone who
don't know much about MFC tricks.

What is scarier is the programs which show the programmer did not have a
clue as to how

MFC works. The code is not only hard to understand, it is hard to
understand because it

is wrong. But it is so hard to understand that only someone really
familiar with MFC can

see that it is written by someone totally clueless and that ti doesn't
actually work. Or

worse still, gives the temporary illusion of working but fails to lock
thread-shared

variables, properly free memory, or any of a dozen blunders.

Finally, my opinion is that you should be
able to understand the design of the imple-
mentation the outsouce have been followed.

To achive that, I really think the outsource
should be able to provide you at least few
UML diagrams.

UML diagrams are often used as a substitute for thinking, and definitely
as a substitute

for proper design. My experience in this has been fairly consistent.

I've seen code

delivered with massive UML diagrams that accurately reflect what the code
does;

unfortunately, the code is wrong, so the UML diagrams are wrong. But they
are impressive,

and they satisfy someone's requirement that the code have UML diagrams.

When I was first learning to program, back in 1963, we were told that
flowcharts were

essential. You wrote your flowchart and all you had to do was translate
it to code. The

flowcharts fell into three categories: (a) high-level diagrams which we
would now call

'workflow diagrams' but which had nothing to do with actual coding (b)

detailed code-level

Re: mfc pitfalls

flowcharts that, once coding started, became irrelevant and obsolete and (c) detailed flowcharts built from the code, which accurately reflected every error in the code.

Ten years later, I was asked to give a talk at a local college about professional programming. One student asked me about flowcharts. I pointed out that I hadn't done one in at least seven years, and that they were an intellectual crutch that rarely had any value, and they were largely a waste of time and effort beyond your first year of programming. This so thoroughly pissed off the professor, who treated flowcharts as a religious experience, that I was pointedly never invited back. But I can now say that I haven't done a flowchart, except some high-level decision trees for teaching, in about four decades. I see UML as being the modern substitute for flowcharts.

Last year someone sent me a question about one of the programs on my Web site. He wanted the DFDs. I had no idea what a DFD was. I had to do a google search. It turns out it was a flowchart. I explained that I would not waste time writing one of these for a trivial 15,000-line program, and for larger programs (those in the range of 100,000-200,000 lines) there was no point in wasting time with them because of the large scale. My message: they're useless for small programs, and they're useless for large programs, so what good are they, exactly?

What really matter are data structure diagrams to show the relationships of the information. This is ultimately the only thing that matters. Flowcharts don't help because they describe process, not information. But I got a program back after five years, and it was full of "ASCII art" of the data structures; without that, I could not possibly have added all the new features they wanted.

UML, at least the forms I've seen, define either high-level dataflow diagrams (not overly useful for programming) or pretend to do detailed analysis...in one case, analyzing the 18 possible states of the state machine (we came up with around 40 states necessary to

Re: mfc pitfalls

represent the problem, but since they'd analyzed "all possible states" the code handled them perfectly. All the bugs were in the states they didn't analyze! But since there were UML diagrams everyone knew the code was perfect...)

Well-designed and well-implemented code usually doesn't need UML diagrams (which we tend to refer to as "software voodoo")

... all the class he does implement should be covered somewhere..

For my part, I also like controlling all the graphical resource ID numbers defined in a project, and their naming, at all time during the implementation.

It might be surprising, but I also avoid the use of warning popup, and never use Try and Catch method. I try to build applications to stay up and running all the possible time.

Designing apps that never use popups is an interesting challenge, and I've done it a couple times. Alas, the file dialog, which in an intelligent world would be an ActiveX control embedded in a dialog, is a real pain. I should be able to do any of the "standard dialogs" just by embedding the associated control in whatever window you want. I had to create my own file dialog, a real pain to get right. I made it a tabbed dialog page in the master program.

I highly prefer using a Logger in my framework, (all situation should be handle), along with a good Tracer that can be unlock to catch problems in the field later, if any still undiscovered during the test phase.

Take a look at my Logging ListBox control ... it appears in every serious app I write.

Re: mfc pitfalls

.. At least.. Uncommented code might be
a very good hint about that outsource!

jmarc..

"telepet9" <telepet9@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote
in message
news:32AD356D-F048-4798-9CF6-DA5709CCD092@xxxxxxxxxxxxxxxxxxxx

Hi,
I need to review code from our outsourcing.
I have some experience with MFC but not to
the extent that I can easily
identify potential problems.
So, can you give me some pointers on what
from your experience should be
avoided while programming using MFC. (We
trying to make the outsourcing
to
provide us code that will be easy to
maintain/debug)
Thank you in advance.

Joseph M. Newcomer [MVP]
email: newcomer@xxxxxxxxxxxx
Web: <http://www.flounder.com>
MVP Tips: http://www.flounder.com/mvp_tips.htm

Joseph M. Newcomer [MVP]
email: newcomer@xxxxxxxxxxxx
Web: <http://www.flounder.com>
MVP Tips: http://www.flounder.com/mvp_tips.htm