Cleaner, more elegant, and harder to recognize

devblogs.microsoft.com/oldnewthing/20050114-00

January 14, 2005

Raymond Chen

It appears that some people interpreted the title of one of my rants from many months ago, "Cleaner, more elegant, and wrong", to be a reference to exceptions in general. (See bibliography reference [35]; observe that the citer even changed the title of my article for me!)

The title of the article was a reference to a specific code snippet that I copied from a book, where the book's author claimed that the code he presented was "cleaner and more elegant". I was pointing out that the code fragment was not only cleaner and more elegant, it was also wrong.

You **can** write correct exception-based programming.

Mind you, it's hard.

On the other hand, just because something is hard doesn't mean that it shouldn't be done.

Here's a breakdown:

Really easy	Hard	Really hard
Writing bad error-code-based code Writing bad exception-based code	Writing good error-code-based code	Writing good exception-based code

It's easy to write bad code, regardless of the error model.

It's hard to write good error-code-based code since you have to check every error code and think about what you should do when an error occurs.

It's **really** hard to write good exception-based code since you have to check every single line of code (indeed, every sub-expression) and think about what exceptions it might raise and how your code will react to it. (In C++ it's not quite so bad because C++ exceptions are raised

only at specific points during execution. In C#, exceptions can be raised at any time.)

But that's okay. Like I said, just because something is hard doesn't mean it shouldn't be done. It's hard to write a device driver, but people do it, and that's a good thing.

But here's another table:

Really easy	Hard	Really hard
Recognizing that error-code-based code is badly-written	Recognizing that error-code-base code is not badly-written	Recognizing that exception-based code is badly-written
Recognizing the difference between bad error-code-based code and not-bad error-code-based code.		Recognizing that exception-based code is not badly-written
		Recognizing the difference between bad exception-based code and not-bad exception-based code

Here's some imaginary error-code-based code. See if you can classify it as "bad" or "not-bad":

```
BOOL ComputeChecksum(LPCTSTR pszFile, DWORD* pdwResult)
{
  HANDLE h = CreateFile(pszFile, GENERIC_READ, FILE_SHARE_READ,
       NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  HANDLE hfm = CreateFileMapping(h, NULL, PAGE_READ, 0, 0, NULL);
  void *pv = MapViewOfFile(hfm, FILE_MAP_READ, 0, 0, 0);
  DWORD dwHeaderSum;
  CheckSumMappedFile(pvBase, GetFileSize(h, NULL),
           &dwHeaderSum, pdwResult);
  UnmapViewOfFile(pv);
  CloseHandle(hfm);
  CloseHandle(h);
  return TRUE;
}
```

This code is obviously bad. No error codes are checked. This is the sort of code you might write when in a hurry, meaning to come back to and improve later. And it's easy to spot that this code needs to be improved big time before it's ready for prime time.

Here's another version:

```
BOOL ComputeChecksum(LPCTSTR pszFile, DWORD* pdwResult)
{
  BOOL fRc = FALSE;
  HANDLE h = CreateFile(pszFile, GENERIC_READ, FILE_SHARE_READ,
       NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  if (h != INVALID_HANDLE_VALUE) {
    HANDLE hfm = CreateFileMapping(h, NULL, PAGE_READ, 0, 0, NULL);
    if (hfm) {
      void *pv = MapViewOfFile(hfm, FILE_MAP_READ, 0, 0, 0);
      if (pv) {
        DWORD dwHeaderSum;
        if (CheckSumMappedFile(pvBase, GetFileSize(h, NULL),
                               &dwHeaderSum, pdwResult)) {
          fRc = TRUE;
        }
        UnmapViewOfFile(pv);
      }
      CloseHandle(hfm);
    }
    CloseHandle(h);
  }
  return fRc;
}
```

This code is still wrong, but it clearly looks like it's trying to be right. It is what I call "not-bad".

Now here's some exception-based code you might write in a hurry:

```
NotifyIcon CreateNotifyIcon()
{
 NotifyIcon icon = new NotifyIcon();
 icon.Text = "Blah blah blah";
 icon.Visible = true;
 icon.Icon = new Icon(GetType(), "cool.ico");
 return icon;
}
```

(This is actual code from a real program in an article about taskbar notification icons, with minor changes in a futile attempt to disguise the source.)

Here's what it might look like after you fix it to be correct in the face of exceptions:

```
NotifyIcon CreateNotifyIcon()
{
 NotifyIcon icon = new NotifyIcon();
 icon.Text = "Blah blah blah";
 icon.Icon = new Icon(GetType(), "cool.ico");
 icon.Visible = true;
 return icon;
}
```

Subtle, isn't it.

It's easy to spot the difference between bad error-code-based code and not-bad error-code-based code: The not-bad error-code-based code checks error codes. The bad error-code-based code never does. Admittedly, it's hard to tell whether the errors were handled correctly, but at least you can tell the difference between bad code and code that isn't bad. (It might not be good, but at least it isn't bad.)

On the other hand, it is extraordinarily difficult to see the difference between bad exception-based code and not-bad exception-based code.

Consequently, when I write code that is exception-based, I do not have the luxury of writing bad code first and then making it not-bad later. If I did that, I wouldn't be able to find the bad code again, since it looks almost identical to not-bad code.

My point isn't that exceptions are bad. My point is that exceptions are too hard and I'm not smart enough to handle them. (And neither, it seems, are book authors, even when they are trying to teach you how to program with exceptions!)

(Yes, there are programming models like RAII and transactions, but rarely do you see sample code that uses either.)

Raymond Chen

Follow