Consequences of using the Copy-Paste method in programming and how to deal with it

I create the analyzer detecting errors in source code of C/C++/C++0x software. So I have to review a large amount of source code of various applications where we detected suspicious code fragments. I have collected a lot of examples demonstrating that an error occurred because of copying and modifying a code fragment. Of course, it has been known for a long time that using Copy-Paste in programming is a bad thing. But let's try to investigate this problem closely instead of limiting ourselves to just saying "do not copy the code".

Usually, when saying about the Copy-Paste method in programming, people mean the following case. Some function or a large code fragment is copied and then this copied code is modified. It causes large amounts of similar code to appear in the program, which complicates its maintenance. You have to replace the same fragments of an algorithm in different functions, so you may easily forget to fix something.

In this case, it is really appropriate to advise not to copy code. If you have some function and want to create a function with similar behavior, you should make a refactoring and arrange the common code in separate methods/classes [1], or use templates and lambda-functions. We will not dwell upon the question how to avoid doubling code because it does not relate to the main issue. What is the most important, you should avoid doubling code in different functions wherever possible. It has been written a lot about this and most programmers are familiar with recommendations.

Now let's focus on the thing authors of books and articles on writing quality code usually do not speak of. Actually, programming is impossible without Copy-Paste.

We all copy small code fragments when we need to write something like this:

Code: Cpp

```
GetMenu()->CheckMenuItem(IDC_ LINES_X, MF_BYCOMMAND | nState);
GetMenu()->CheckMenuItem(IDC_ LINES_Y, MF_BYCOMMAND | nState);
```

In good conscience, we always feel reluctant to type a line that differs from another line only in the 'Y' character used instead of 'X'. And this is right and reasonable. It is faster to copy and edit text than type a second line from the very beginning even with the help of special tools such as Visual Assist and IntelliSence.

Note that it is unreasonable to speak about doubling code here: you cannot make it simpler anyway. There are a lot of such examples in every program. If you don't like that we deal with GUI in the sample above, well, take some other task - you will get the same thing:

Code: Cpp

```
int texlump1 = Wads.CheckNumForName("TEXTURE1", ns_global, wadnum);
int texlump2 = Wads.CheckNumForName("TEXTURE2", ns_global, wadnum);
```

The problem is that an error is also highly probable when using this "microcopying". Since you copy such small code fragments much more often than large blocks, it is really a crucial issue. It is not clear how to deal with it, so they try not to speak about it. You cannot prohibit programmers from copying code.

Many of such errors are detected at the first launch of the program and are eliminated quickly and painlessly. But a lot of them remain in code and live for years waiting for their time to show up. Such errors are rather difficult to detect because a person has to review similar code lines and gradually gets less attentive. The probability of Copy-Paste related errors does not depend on programmer's skill. Any person might make a misprint and miss something. Defects of this type occur even in very famous and quality products.

To make it clearer what errors we mean, let's consider several code samples taken from open-source projects. I detected errors described in this article using the general analyzer included into PVS-Studio [2].

The following code is taken from the Audacity application intended for sound recording and editing.

Code: Cpp

```
sampleCount VoiceKey::OnBackward (...) {
  ...
  int atrend = sgn(
    buffer[samplesleft - 2]-buffer[samplesleft - 1]);

  int ztrend = sgn(
    buffer[samplesleft - WindowSizeInt-2]-
      buffer[samplesleft - WindowSizeInt-2]);
  ...
}
```

The programmer was courageous and wrote initialization of the 'atrend' variable correctly. Then he started to write initialization of the 'ztrend' variable. He wrote "sgn(buffer[samplesleft - WindowSizeInt-2]", gave a sigh and copied the line fragment which he then forgot to edit. As a result, the 'sgn' function gets 0 as an argument.

The following scenario is the same. The programmer writes a long condition in 3D SDK Crystal Space:

Code: Cpp

```
inline_ bool Contains(const LSS& lss)
{
  // We check the LSS contains the two
  // spheres at the start and end of the sweep
  return
    Contains(Sphere(lss.mP0, lss.mRadius)) &&
    Contains(Sphere(lss.mP0, lss.mRadius));
}
```

One cannot resist the urge to copy "Contains(Sphere(lss.mP0, lss.mRadius))" and replace the name 'mP0' with 'mP1'. But it is so easy to forget about it.

Perhaps you noticed sometimes that program windows started behaving in a strange way. For instance, many programmers will remember the search window in the first edition of Visual Studio 2010. I think such strange things occur due to luck and code like this:

Code: Cpp

```
void COX3DTabViewContainer::OnNcPaint()
{
  ...
  if(rectClient.top<rectClient.bottom &&
     rectClient.top<rectClient.bottom)
  {
    dc.ExcludeClipRect(rectClient);
  }
  ...
}
```

This code was taken from a famous class set Ultimate ToolBox. Whether the control is drawn correctly or not depends upon its location.

And in eLynx Image Processing SDK, programmers copied a whole line therefore spreading the misprint throughout the code.

Code: Cpp

```
void uteTestRunner::StressBayer(uint32 iFlags)
{
  ...
  static EPixelFormat ms_pfList[] =
    { PF_Lub, PF_Lus, PF_Li, PF_Lf, PF_Ld };
  const int fsize = sizeof(ms_pfList) / sizeof(ms_pfList);

  static EBayerMatrix ms_bmList[] =
    { BM_GRBG, BM_GBRG, BM_RGGB, BM_BGGR, BM_None };
  const int bsize = sizeof(ms_bmList) / sizeof(ms_bmList);
  ...
}
```

The pointer dereferencing operation missing here causes the 'fsize' variable to equal 1. Then this code was adapted for initializing 'bsize'. I do not believe that one can make such a mistake twice without copying the code.

In the EIB Suite project, it was the line "if (_relativeTime <= 143)" which was copied and edited. But they

forgot to change it in the last condition:

Code: Cpp

```
string TimePeriod::toString() const
{
  ...
  if (_relativeTime <= 143)
    os << ((int) relativeTime + 1) * 5 <<  (" minutes");
  else if (_relativeTime <= 167)
    os << 12 * 60 + ((int)_relativeTime - 143) * 30 << _(" minutes");
  else if (_relativeTime <= 196)
    os << (int)_relativeTime - 166 << _(" days");
  else if (_relativeTime <= 143)
    os << (int)_relativeTime - 192 << _(" weeks");
  ...
}
```

It means that the code "os << (int)_relativeTime - 192 << _(" weeks");" will never get control.

Even programmers in Intel company are only programmers and not demigods. Here is a bad copying in the TickerTape project:

Code: Cpp

```
void DXUTUpdateD3D10DeviceStats(...)
{
  ...
  else if( DeviceType == D3D10_DRIVER_TYPE_SOFTWARE )
    wcscpy_s( pstrDeviceStats, 256, L"WARP" );
  else if( DeviceType == D3D10_DRIVER_TYPE_HARDWARE )
    wcscpy_s( pstrDeviceStats, 256, L"HARDWARE" );
  else if( DeviceType == D3D10_DRIVER_TYPE_SOFTWARE )
    wcscpy_s( pstrDeviceStats, 256, L"SOFTWARE" );
  ...
}
```

The "DeviceType == D3D10_DRIVER_TYPE_SOFTWARE" condition is repeated twice.

Well, it is quite easy to miss an error in the jungle of conditional statements. In the implementationMulti-threaded Dynamic Queue, one and the same branch of the code will be executed regardless of the value returned by the IsFixed() function:

Code: Cpp

```
BOOL CGridCellBase::PrintCell(...)
{
  ...
  if(IsFixed())
    crFG = (GetBackClr() != CLR_DEFAULT) ?
      GetTextClr() : pDefaultCell->GetTextClr();
  else
    crFG = (GetBackClr() != CLR_DEFAULT) ?
      GetTextClr() : pDefaultCell->GetTextClr();
  ...
}
```

By the way, how easy and pleasant it is to copy code! You can afford one more line.

Code: Cpp

```
void RB_CalcColorFromOneMinusEntity( unsigned char *dstColors ) {
  ...
  unsigned char invModulate[3];
  ...
  invModulate[0] = 255 - backEnd.currentEntity->e.shaderRGBA[0];
  invModulate[1] = 255 - backEnd.currentEntity->e.shaderRGBA[1];
  invModulate[2] = 255 - backEnd.currentEntity->e.shaderRGBA[2];
  invModulate[3] = 255 - backEnd.currentEntity->e.shaderRGBA[3];
  ...
}
```

It does not matter that the 'invModulate' array consists only of three items. This code is taken from the legendary game Wolfenstein 3D.

And here is a more complicated sample in the end. This code is taken from a rather useful toolNotepad++.

Code: Cpp

```
void KeyWordsStyleDialog::updateDlg()
{
  ...
  Style & w1Style =
    _pUserLang->_styleArray.getStyler(STYLE_WORD1_INDEX);
  styleUpdate(w1Style, _pFgColour[0], _pBgColour[0],
    IDC_KEYWORD1_FONT_COMBO, IDC_KEYWORD1_FONTSIZE_COMBO,
    IDC_KEYWORD1_BOLD_CHECK, IDC_KEYWORD1_ITALIC_CHECK,
    IDC_KEYWORD1_UNDERLINE_CHECK);
```

```
  Style & w2Style =
    _pUserLang->_styleArray.getStyler(STYLE_WORD2_INDEX);
  styleUpdate(w2Style, _pFgColour[1], _pBgColour[1],
    IDC_KEYWORD2_FONT_COMBO, IDC_KEYWORD2_FONTSIZE_COMBO,
    IDC_KEYWORD2_BOLD_CHECK, IDC_KEYWORD2_ITALIC_CHECK,
    IDC_KEYWORD2_UNDERLINE_CHECK);

  Style & w3Style =
    _pUserLang->_styleArray.getStyler(STYLE_WORD3_INDEX);
  styleUpdate(w3Style, _pFgColour[2], _pBgColour[2],
    IDC_KEYWORD3_FONT_COMBO, IDC_KEYWORD3_FONTSIZE_COMBO,
    IDC_KEYWORD3_BOLD_CHECK, IDC_KEYWORD3_BOLD_CHECK,
    IDC_KEYWORD3_UNDERLINE_CHECK);

  Style & w4Style =
    _pUserLang->_styleArray.getStyler(STYLE_WORD4_INDEX);
  styleUpdate(w4Style, _pFgColour[3], _pBgColour[3],
    IDC_KEYWORD4_FONT_COMBO, IDC_KEYWORD4_FONTSIZE_COMBO,
    IDC_KEYWORD4_BOLD_CHECK, IDC_KEYWORD4_ITALIC_CHECK,
    IDC_KEYWORD4_UNDERLINE_CHECK);
  ...
}
```

You have to strain your eyes greatly trying to find an error here. So let me abridge this code to make it clearer:

Code: Cpp

```
styleUpdate(...
  IDC_KEYWORD1_BOLD_CHECK, IDC_KEYWORD1_ITALIC_CHECK,
  ...);
styleUpdate(...
  IDC_KEYWORD2_BOLD_CHECK, IDC_KEYWORD2_ITALIC_CHECK,
  ...);
styleUpdate(...
  IDC_KEYWORD3_BOLD_CHECK, IDC_KEYWORD3_BOLD_CHECK,
  ...);
styleUpdate(...
  IDC_KEYWORD4_BOLD_CHECK, IDC_KEYWORD4_ITALIC_CHECK,
  ...);
```

The developer's hand shook and he copied a wrong resource's name.

I can give you other defect code fragments in this article, but it is not interesting. I just wanted to say by all these examples that such errors can be found in various projects and both novice programmers and

skilled programmers make them. Now let's discuss what we should do with all that stuff.

Well, to be frank, I do not have a complete answer. At least, I never read about such situations in books but often came across consequences of small Copy-Paste's in practice, including my own applications. So I will have to improvise while answering the question.

Let's proceed from the following suggestion:

Programmers are copying code fragments and will continue doing this because it is convenient. So, these errors will always occur in programs.

My conclusion is:

You cannot prevent such errors completely but you may try to make them less probable.

I see two ways of how we could make errors of this type fewer. First, we should use static code analyzers. They allow us to detect many errors of this class at the very early stages. It is cheaper and easier to find and fix an error right after writing the code than handle the same error detected during the testing.

The second method to make errors fewer in some cases is to discipline oneself and edit the code being copied in a special way. For example:

Code: Cpp

```
int ztrend = sgn(
  buffer[samplesleft - WindowSizeInt-2]-buffer[samplesleft - WindowSizeInt-
2]);
```

It is much easier to notice an error when the code is written in the following way:

Code: Cpp

```
int ztrend = sgn(
  buffer[samplesleft - WindowSizeInt-2] -
  buffer[samplesleft - WindowSizeInt-2]);
```

You should edit the code so that fragments which must differ from each other are visually arranged in a column. It is much more difficult to make an error if you use this method. Of course, it will not save you in many cases - I have mentioned such samples above. But still it is better than nothing.

Unfortunately, I do not know any other ways to reduce the number of Copy-Paste related errors. You may use tools to search for repeated and similar code but it rather refers to my advice concerning using static analyzers.

So, I appeal to you readers. I will appreciate if you share some of your ideas concerning this issue with me and offer some other methods of avoiding Copy-Paste related errors. Perhaps we will get nice ideas that will help many programmers.

Source: http://www.go4expert.com/articles/consequences-using-copy-paste-method-t24705/