the magazine of the accu

www.accu.org

{cvu}

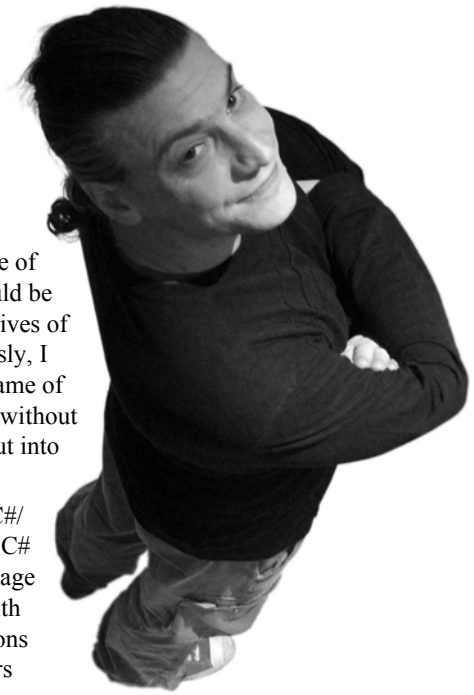
Volume 25 • Issue 5 • November 2013 • £3

Features

Regulars

{cvu}
Volume 25 Issue 5
November 2013
ISSN 1354-3164
www.accu.org

IInterface and Other Religions



It's become deeply ingrained in many C# developers that interfaces have a capital 'I' prefix to indicate that they are interfaces. In fact, the .Net naming guidelines suggest it's good practice; more than that, those guidelines also recommend that some 'standard' implementation's name should differ only by removing the 'I'. IFindThisStrange. Amongst other reasons, the name of the interface is the name that *most* of the code should be using, and not having the prefix would extend the lives of programmers' keyboards everywhere. More seriously, I think that having a class/interface pair where the name of the class is the same as the name of the *abstraction* without an 'I' suggests that insufficient thought has been put into either the name or the abstraction.

There are other seemingly religious beliefs in the C#/.Net community (not just there, but I'm picking on C# programmers this month) such as the idea that Garbage Collection absolves you from any responsibility with regard to the lifetimes of your objects, that Singletons (plural, obviously) are Good™ and that constructors shouldn't throw exceptions. A quick hunt on say StackOverflow.com for 'Singleton' will reward you with more results than you could possibly read in a lifetime! Within those results, you'll find myriad implementations, with accompanying arguments as to why that one is better than this one, along with techniques for creating multiple instances of a Singleton type, and how to write one where the underlying implementation can be changed. It's one of the reasons I find StackOverflow so entertaining. Just occasionally you'll stumble across a small voice enquiring if anyone's thought about *not* using a Singleton..."If you only want one, only create one."

There are, then, things I would like to know. Why is it so important to be able to distinguish an interface from a class by its type? How many people have ever needed just one instance of a type? Really really *really*? How hard would it be to create a new language with constructors that have return values? When, exactly, can I have my memory back? You know where to send the answers by now.

STEVE LOVE
FEATURES EDITOR

The official magazine of ACCU

ACCU is an organisation of programmers who care about professionalism in programming. That is, we care about writing good code, and about writing it in a good way. We are dedicated to raising the standard of programming.

ACCU exists for programmers at all levels of experience, from students and trainees to experienced developers. As well as publishing magazines, we run a respected annual developers' conference, and provide targeted mentored developer projects.

The articles in this magazine have all been written by programmers, for programmers – and have been contributed free of charge.

To find out more about ACCU's activities, or to join the organisation and subscribe to this magazine, go to www.accu.org.

Membership costs are very low as this is a non-profit organisation.

DIALOGUE

REGULARS

FEATURES

SUBMISSION DATES

C Vu 25.6: 1st December 2013
C Vu 26.1: 1st February 2014

Overload 119: 1st January 2013
Overload 120: 1st March 2014

WRITE FOR C VU

Both C Vu and Overload rely on articles submitted by you, the readers. We need articles at all levels of software development experience. What are you working on right now? Let us know!

Send articles to cvu@accu.org. The friendly magazine production team is on hand if you need help or have any queries.

ADVERTISE WITH US

The ACCU magazines represent an effective, targeted advertising channel. 80% of our readers make purchasing decisions or recommend products for their organisations.

To advertise in the pages of C Vu or Overload, contact the advertising officer at ads@accu.org.

Our advertising rates are very reasonable, and we offer advertising discounts for corporate members.

COPYRIGHTS AND TRADE MARKS

The Ethical Programmer (Part 2)

Pete Goodliffe acts ethically. Again.

Treat others as you would want them to treat you.
~ Matthew 7:12

This is the second installment in a mini-series on 'ethics' in programming. We've already looked at an ethical approach towards the code we write and considered the specific legal obligations we must observe.

However, being *ethical* drives much deeper than this; towards an equivalent of the medical Hippocratic Oath.

An ethical approach implies certain attitudes towards other people: towards users, team mates, managers, employers and so on. Even towards yourself. Let's look at this in more detail.

At the end of this article, I promise you a new *Hippocodic Oath*. Why should doctors have all the fun?

Attitude to people

We've already considered some 'ethical attitudes' towards people, since we write code primarily for the audience of other programmers, not for the compiler. Programming problems are almost always people problems, even if the solutions have a technical nature.

> Good attitudes towards code are *also* good attitudes to other programmers.

Imagine yourself as a heroic coder. The kind of programmer who wears their underwear atop their trousers, and not simply as a geeky fashion faux pas. Now: *do not abuse your super powers for evil*. Only write software for the good of mankind.

In practice, this means: don't write viruses or malware. Don't write software that breaks the law. Don't write software to make people's life worse, either materially, physically, emotionally or psychologically.

Don't turn to the dark side.

> Do not write software that will make another person's life worse. This is an abuse of power.

And here we open a wonderful new can of worms: is it ethical to write software that makes some people very rich at the expense of poorer people, if it doesn't break any laws? Is it ethical to write software to distribute pornographic content, if the software itself breaks no laws? You can argue that people are exploited as a byproduct of both activities. Is it ethical to work in these industries? This is a question that I can only leave for the reader to answer.

What about working on military projects? Would an ethical programmer feel comfortable working on weapons systems that could be used to take a life? Perhaps such a system will actually *save* lives by acting as a deterrent against attack. This is a great example of how the ethics of software development is a philosophical topic, not an entirely black and white affair. You have to reconcile yourself with the consequences your code has on other peoples' lives.

Team mates

The people you encounter most frequently in your programming career are your team-mates; the programmers, testers, etc, who you work with closely day by day. The ethical programmer works conscientiously with all of them, looking to honour each team member, and to work together to achieve the best result possible.

Speak well of all people. Do not engage in gossip or back-biting. Do not encourage jokes at the expense of others.

Always believe that anyone, no matter the level of mature or how inexperienced, has something valuable to contribute. They have an opinion that is worth hearing, and should be able to put forward points of view without being shot down.

Be honest and trustworthy. Deal with everyone with integrity.

Do not pretend to agree with someone when you believe they are wrong; this is dishonest and rarely useful. Constructive disagreements and reasoned discussions can lead to genuinely better code design decisions. Understand what level of 'debate' a team member can handle. Some people thrive on intense, passionate intellectual debate; others are frightened by confrontation. The ethical programmer seeks to engage in the most productive discussion result without insulting or offending anyone. This isn't always possible, but it the goal is to always treat people with respect.

Do not discriminate against anyone, on any grounds, including: gender, race, physical ability, sexual orientation, mental capability, or skill.

The ethical programmer takes great care to deal with 'difficult people' in the most fair, transparent way, attempts to diffuse difficult situations, and works to avoid unnecessary conflict.

> Treat others as you would like them to treat with you.

Manager/scrum master/team

Many issues that may be seen as agreements between you and your manager can also be seen as ethical contracts between you and the other members of the team, since the manager acts as a bridge with the team.

Do not take credit for work that is not yours, even if you take someone else's idea and modify it to fit the context a bit.

Do not give an unnecessarily high estimate for the complexity of a task, just so that you can slack off and do something more enjoyable on the pretence of working hard on a tricky problem.

If you see looming issues that will prevent the smooth running of the project, report it as soon as you notice. Do not hide bad news because you don't want to worry or offend someone, or be seen to be a pessimistic killjoy. The sooner issues are raised, planned around, and dealt with, the smoother the project will go for everyone.

If you spot a bug in the system, report it. Put a bug in the fault tracking system. Don't turn a blind eye and hope that no one else will notice it.

> The ethical programmer takes responsibility for the quality of the product at all times.

Do not pretend to have skills or technical knowledge that you do not possess. Don't put a project schedule in danger by committing to a task you cannot complete, just because you think it's interesting and you'd like to work on it.

If you realise that a task you're working on is going to take significantly longer to complete that you expected, voice that concern as soon as

PETE GOODLIFFE

Pete Goodliffe is a programmer who never stays at the same place in the software food chain. He has a passion for curry and doesn't wear shoes. Pete can be contacted at pete@goodliffe.net or @petegoodliffe

possible. The ethical programmer does not keep it quiet in order to save face.

When given responsibility for something, honour the trust placed in you. Work to the best of your ability to fulfil that responsibility.

Employer

Treat your employer with respect.

Do not reveal company confidential information, either source code, algorithms, or inside information. Do not break the terms of your employment contact.

Don't sell work you have done for one company to another, unless you have express permission to do so.

However, if you realise that your employer is engaged in illegal activities, it is your ethical duty to raise it with them, or report their malfeasance as appropriate. The ethical programmer does not turn a blind eye to wrong-doing just to keep their own job secure.

Do not falsely represent, or bad mouth your employer in public.

Yourself

The ethical programmer keeps themself up to date on good programming practice.

They do not work so hard that they burn themself out. This is not only personally disadvantageous, but also bad news for the whole team. Hours and hours of extra work, week in, week out, will lead to a tired programmer, which will inevitably lead to sloppy mistakes, and a worse final outcome. The ethical programmer understands that, although its nice to look like a hero who works incredibly hard, it's not a good idea to set unrealistic expectations and to burn yourself out.

> A tired programmer is no use to anyone. Do not over-work. Know your limits.

You have a right to expect the same ethical conduct from others you work with.

The Hippocodic Oath

What would the ideal programmers' *code of ethics* look like? The documents formulated in [1] and [2] are formal, lengthy and hard to recall.

We need something pithier; more of a mission statement for the ethical programmer.

I humbly suggest:

> I swear to cause no harm to the code, or to the business; to seek personal advancement, and the advancement of my craft. I will perform my allotted tasks to the best of my ability, working harmoniously with my team. I will deal with others with integrity, working to make the project, and the team, maximally effective and valuable.

Conclusion

Ethics is in origin the art of recommending to others the sacrifices required for cooperation with oneself.
~ Bertrand Russell

How much you care about this kind of thing depends on your level of diligence, your professionalism, and your personal moral code. Are you in the programming game for fun, enjoyment, and the development of great code? Or are you in it for yourself, for career development (at the expense of others, if necessary), to make as much money as you can, and hoist yourself above others on the professional ladder?

It's a choice. You can chose your attitude. It will shape the trajectory of your career.

I find my attitudes shaped by my desire to write good code and to participate in a community that cares about working well. I seek to work amongst excellent developers that I can learn from. As a Christian, I have a moral framework that encourages me to prefer others over myself and to honour my employers. This shapes the way I act.

I conclude from what we've seen above that there are (at least) two levels to the ethical programming career: the mandate to 'do no harm' is the base level, to not tread on people, or be involved in work that exploits others. Beyond this is a more involved ethical mantra: to only work on projects that provide sound social benefits, to specifically *make the world better* with your talents, and to share knowledge in order to advance the programming craft. I suspect that many people ascribe to the first camp of ethics. Fewer feel the urge, or able to devote themselves to the cause of the second.

How do your beliefs and attitudes shape the way you work? Do you think you are an ethical programmer? ■

References

[1] ACM Code of Ethics http://www.acm.org/about/code-of-ethics

[2] BSI code of Conduct http://www.bcs.org/server.php?show=nav.6030

Questions

1. What should you do if you identify another programmer acting 'unethically'. How does this answer differ if they are a co-worker, a friend, someone who you've been asked to give a reference for, or a coder you've met but do not work directly with?
2. How do software patents fit into the world of ethical programming?
3. Should your passion for software development have any bearing on how much you care about ethical issues? Does a passionate programmer act more ethically than a career coder?

Trying Python for Size (Part 1)

Andy Burgess takes his first steps in Python.

My client, for whom I've written an industry leading standards-checking super checklist tool, used by Tesco and Morrison supermarkets' suppliers on Windows, suggested they'd like to make it cross-platform. I also had a requirement from a hardware-developer contact to create a cross-platform front-end for his devices, so I started investigating where I'd like to direct future efforts for my software development.

I experimented with multiple cross-platform development environments – PHP-GTK appealed, because I develop websites in PHP, but the environment seems largely abandoned nowadays, so I moved away from investigating that further. I experimented with MonoDevelop – a good environment making Windows-style exes run on other platforms, but I didn't really want to be tweaking Windows EXEs to run on other platforms. I never really got into Qt, but again it looked promising. Real Basic also seemed really good, but I wanted to be able to develop software to run on the Raspberry Pi too, and Real Basic fell down here. I also looked into Servoy, which is a fantastic piece of cross-platform software, but the licensing model wouldn't work for me.

I've always been a bit averse to using IDE's (Interactive Development Environments), because I don't necessarily want to work in the same way as has been programmed. If it's at all possible, I'd rather use the command line and a basic editor (I can even use DOS's EDLIN!), rather than someone else's idea of what's best to program with (however, I've got quite good with Eclipse for Android Programming), but I digress.

I develop quite a few websites through a basic text editor (TextWrangler on the Mac) and a local Apache server, and although they don't have the 'Intellisense' of helping you with the code – like Microsoft's Visual Studio tools, I'm happier that way, probably because it's the way I originally learnt to program! I'm pretty quick at the Unix VI (or VIM) editor too.

What about Python?

I bought a Raspberry Pi when the devices first came out, and waited 3 months for it to arrive. As a RISC OS enthusiast, I wanted to see if that worked on the Raspberry Pi – it does, but I'm not discussing that here.

Whilst browsing a book in my local bookshop on developing for the Raspberry Pi in the Python language, and thinking of my hardware client's job, I thought I'd find out a little more about the Python language. I always seem to have problems installing software on Unix or Linux – there always seems to be a missing 'dependency', and I always hit a wall trying to find out how to get through all the dependencies. My main driving force into looking into python was that Mac OS is written over a Unix implementation (Darwin), and that a lot of the Unix command-line tools were already installed, but was python?

I was already comfortable with the Terminal application in Mac OS (Applications → Terminal), so started it up and at the prompt:

```
[MacBookPro:~] andy%
```

I typed

```
python
```

and received:

```
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr 9 2012,
20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on
darwin
Type "help", "copyright", "credits" or "license"
for more information.
>>>
```

Good! That meant python was already installed on my Mac's OS X (Mountain Lion). I was on the older python (version 2.7.3). There's a new version, python 3 which is heavily under development, and is promised to be vastly different, but I was happier with the older version for the moment.

Now it's not abundantly clear what to do here, so I typed

```
quit
```

and received

```
Use quit() or Ctrl-D (i.e. EOF) to exit
>>>
```

Ah, that's how I get out! I typed

```
quit()
```

... and left python.

Linux

I did the same on my Suse Linux computer and received the same kind of prompt, except that I only had python version 2.5.2 on that computer.

Windows

I checked on my Windows 7 PC in the 'cmd' (command) window, which I still – incorrectly – call the 'DOS window'.

```
python
```

and received

```
Invalid command or file name.
```

That was to be expected. I'd not installed Python, and I didn't expect Windows to have it as standard.

Downloading for Windows

The Python website [1] has downloads for all the usual platforms, including more improved versions than those already installed on Mac OS. I had to work out which processor my Windows environment was using (by using Control Panel → System), I was using 32 bit Windows, so downloaded the relevant Python 2.7.5 installer, and the relevant Python 3.3.2 installer. They can both live together on your Windows system. I concentrated on Python 2.

The Windows installation of Python, introduced me to the windowing Python Shell. It was also available somewhere on my Mac, but I'd not found it. From the windowing shell, I could create a new program window by using File → New. The IDLE editor appeared.

Right, now, I knew that I had Python on all three environments, I needed to learn what what it's about.

Learning Python

I browsed a few online tutorials about Python – initially in assessing how I could use it on my Raspberry Pi. One of the best I found at www.zetcode.com

ANDY BURGESS

Andy Burgess comes from a background in green-screen programming, using COBOL and ALGOL on mainframes. He now runs his own business in Web development and is interested in cross-platform mobile development. He can be contacted at andy@jet-net.co.uk

On that site is an introduction to the language [2]. There are also further tutorials on GUI programming in python.

The code can be entered in a text editor – e.g. Unix's Vi or VIM (or Windows' Edit or Notepad), or interactively using the IDLE editor. IDLE has some useful features for someone like me who doesn't normally like IDEs. If you have a bug in your program, the error is displayed in the interactive window, and the line is highlighted in red (if a semantic error) in the IDLE editor. Also pressing the tab key on the Mac brought up some intellisense-type syntax suggestions, but not always when I wanted it!

Python works like a well-structured program, but is more intelligent in its structure than, say PHP or Visual Basic. It's 'minimalistic'. No more do you have to fiddle about matching missing curly brackets (C, C++, PHP, Java etc), **If**s and **End If**s (Visual Basic, ASP, ASP.NET) or even **begin**s and **end**s (Algol and Pascal). The number of white space characters determine the beginning and end of the block.

Whitespacing indentation

Instead of C, PHP or Java code:

```
class fred()
{
  function open()
  {
    command 1;
    command 2;
  }
  function close()
  {
    command 3;
    command 4;
  }
}
```

You would use

```
class fred():
<tab>open(self):
<tab><tab>command1:
<tab><tab>command2:
<tab>close(self):
<tab><tab>command3:
<tab><tab>command4:
```

so you get code looking like:

```
class fred():
    open(self):
        command1
        command2

    close(self):
        command3
        command4
```

The fact that the **close()** function has only one <tab> rather than two, tells python that this is another function at the same level as **open()**. Neat, and no need for wondering if you've closed the right bracket at the right time (and the amount of times I've had to do that in PHP, I won't mention!).

The one 'gotcha' in creating classes in Python is that you have to include a reference to the class using **self** as a parameter to functions.

I considered this a very elegant method of programming, as you don't have to remember if the statement ends with a curly bracket, an **End If**, **End While** or whatever; however I did have to remember that the **else** statement needs a colon after it – **else:**.

The only 'common' statement that other languages seem to have that Python does not is the **Case** or **Switch** statement. That statement is implemented using **if** .... **elif** statements. Not brilliant, but not too much of a heartache.

Comparisons with other languages

Array indexes are like Algol using colons to separate the indices. For example, in PHP an Array would be referenced with **$myarray[0,1]**, in Visual Basic this would be **myarray(0,1)**, and in Python would be **myarray[0:1]**.

Python provides **try** ... **except** structures, which seem to be common in most modern languages, but still a bit of a new concept for a dinosaur like me.

Comments are prefixed with **#** – like they can be with PHP, but **//** and **/*** ... ***/**"are more commonly used. You cannot use **/*** ... ***/** in python.

Python is extremely strict with code spacing, whereas languages like PHP, C and Visual Basic aren't that bothered if one line (not in an **if** or a **while**) is a little indented. e.g. for the code

```
if fred==1:
    print "one"
        print "two"
```

Python will moan

```
"Theres an error in your program:
Unexpected indent"
```

Also mixing tabs and fixed spacing causes the language to moan too - it's best to use one or the other. There is a function in IDLE Format → Untabify Region to convert tabs to spaces, but in practice for me, sometimes it worked, sometimes it didn't when I used a file I'd created in another editor.

Like internet pages or email, you can specify a encoding type for the content of the program. Commonly UTF-8 is used through the directive:

```
-*- coding: utf-8 -*-
```

at the top of your program. Also like in Unix-based languages like Perl you can specify a 'she bang' to tell the interpreter that this code is Python:

```
#!/usr/bin/env/python
```

but in my experience this can be omitted.

Python can also be used to create HTTP requests which would allow applications to communicate with an external website.

GUI Development

There are two options I've considered for developing apps with a GUI interface in Python – TkInter and wxPython. I believe GTK can be used too, but I've not looked into that. The one thing I really love about GUI development on Python is that you can write the code manually – like I used to do with Fox Pro. This is also a drawback in that it doesn't really help you if you don't know where to start.

Tkinter

I discovered that there was a GUI thing for Python called Tkinter [3], which looked really promising as code generated looked 'native' for the destination operating system. What was more appealing was that it was already available for my Mac, and was already installed, so no messing about with missing dependencies. Tkinter is python's de-facto standard GUI package, which also lent it credibility for me. It's great for really simple apps, but if you need to use more complicated things like webpages within your application you may struggle.

WxPython

I'd come across wxWidgets as being available for Python (called wxPython [4]), this GUI framework is more powerful than Tkinter, but it's an additional installation for Python.

I really struggled previously trying to install wxWidgets for my Mac previously outside of Python (the dreaded dependencies issue struck again), but installing it for Python was relatively straightforward for Mac, PC and Linux. From my experience, wkPython can do far, far more than Tkinter can do.

Watch out for more details of GUI development using Python in a later article.

The Windows XP Threat: A Call to Action

Silas S. Brown brings an imminent problem to our attention.

My friend showed me her new computer. She had bought it quite cheaply from her company, as they were getting rid of their old stock. You can probably guess the rest. The box is running Windows XP, which reaches end of life on April 8, 2014, and her company are evidently dealing with their obsolete hardware by dumping it on employees.

Continuing to run Windows XP after April 2014 will be a major security risk. As a Microsoft security engineer pointed out on their blog [1],

> When Microsoft releases a security update...criminals will...identify the specific section of code that contains the vulnerability...develop code that will allow them to exploit it on systems that do not have the security update installed on them. They also try to identify whether the vulnerability exists in other products...if a vulnerability is addressed in one version of Windows, researchers investigate whether other versions of Windows have the same vulnerability...the Microsoft Security Response Center...[releases] security updates for all affected products simultaneously...But after April 8, 2014, organizations that continue to run Windows XP won't have this advantage over attackers any longer. The very first month that Microsoft releases security updates for supported versions of Windows, attackers will reverse engineer those updates, find the vulnerabilities and test Windows XP to see if it shares those vulnerabilities. If it does, attackers will attempt to develop exploit code that can take advantage of those vulnerabilities on Windows XP. Since a security update will never become available for Windows XP to address these vulnerabilities, Windows XP will essentially have a 'zero day' vulnerability forever.

I shouldn't have to tell ACCU readers what this means. If the owners of these machines continue to connect them to the Internet (as the majority of them undoubtedly will), then malware is going to steal credentials and perform other nasties on a grand scale. People who receive these machines delude themselves into thinking that a simple virus checker will save them. It will not.

If organizations want to get rid of their Windows XP machines, they shouldn't just dump them on employees who don't know the risks. Otherwise they are shooting themselves in the foot. What if some of those employees keep company confidential information on their home computers? Even if that's not the case, does the company really want their employees to suffer from malware at home?

Since it's not usually possible to upgrade the version of Windows on a piece of hardware that's not up to spec for newer versions, the obvious choice is to put GNU/Linux on the old hardware. Ideally the machines should be donated to an outfit which can do this before finding good homes for them, or at the very least, employees and others who take them should

SILAS S. BROWN

Silas Brown is a partially-sighted Computer Science post-doc in Cambridge who works part-time in assistant tuition and part-time for a startup. He's been developing language-related software since events in Cambridge led him to acquire fluent Chinese. He has been an ACCU member since 1994. Silas can be contacted at ssb22@cam.ac.uk

Trying Python for Size (Part 1) (continued)

Database

see https://www.ssucet.org/mod/resource/view.php?id=440

Python can read mySQL databases. This is a major draw to the language for me, as my websites make use of mySQL databases. There are two implementations of mySQL for Python – MYSQL-python and pyMySQL. I didn't manage to get MYSQL-Python to cooperate for me, but had great success with pyMySQL. I intend to do a later article on mySQL for python (using pyMySQL).

Web-Capable

I had come across many Python code web pages with the `.py` extension on the web – so also knew that I could potentially incorporate it into a website. In reality you need to ensure that your ISP will provide access to the cgi-bin folder to run server scripts. ISPs appear to be reluctant to do this as it opens up potential vulnerabilities to the world wide web in Python or Perl programs. I've run successful basic Python scripts on my test server, but not really gone much further. It may be something I'll want to investigate in future.

Executable

Python can be made executable. py2app makes an Apple Mac OS X application, and py2exe makes a Windows EXE. I've experienced success with these two applications making Python executables on Mac OS and Windows (I couldn't get Tkinter programs working, but wxWidgets ones did), but have not had success on Linux. PyInstaller promises to provide the ability to make EXEs on Mac OS, Windows and Linux, but I couldn't get it to work properly on my Mac, so have not, as yet pursued it further.

Conclusion

The things I really like about Python are its simplicity, like old interpreted languages like BBC BASIC or Microsoft's QBASIC. I feel it has the power of a language like C, but is more meaningful to read and work with. I like its ability to create cross-platform GUI applications, and I like the way it's web-capable and can use databases.

I like its block-structure, and love the fact that you don't have to remember to open and close brackets (or **`if`**s and **`endif`**s), just reduce the indentation to close the block. I like the fact that extras – like wxWidgets can easily be downloaded and incorporated into the language.

But it's not just me who obviously thinks this way. There are a vast amount of code-snippets available on the internet to help you achieve what you want to achieve. The language is well supported, and appears to be in constant development. I'm completely sold on it, even though it's free! ■

References

[1] http://www.python.org/download/
[2] www.zetcode.com/lang/python/
[3] www.zetcode.com/gui/tkinter/
[4] www.zetcode.com/wxpython/

be provided with information regarding a suitable Linux distribution (my current recommendation for beginners is Mint Xfce 13 [2] which has a 2017 end-of-life and can be upgraded thereafter, but it's possible that this won't run on all old hardware so perhaps a range of distributions should be considered).

To play a small part in the warning work, I have now placed a snippet of Javascript (Listing 1) in the template that drives my personal website [3].

Using document.write to add an extra script in this way means that I can make xp.js as big as I want without slowing down users of more modern browsers. (In HTML 5 it's not necessary to give a 'language' or 'type' attribute to the 'script' element, and all earlier browsers defaulted to Javascript anyway.) True, the above will work only if Javascript is available and switched on, but it's a safe bet that the vast majority of the target audience will have it, and using Javascript means I can host the site on a server that can't do a lot of server-side processing. Anyway if you do browser-specific things on the server then you have to tell web proxies that your pages are User-Agent specific, which is not very nice to the proxies. Also, making it Javascript means that inside xp.js I can add the contents of Listing 2.

Listing 1

```
<script><!-- check for old Windows:
if(navigator.userAgent.indexOf("Windows NT 5")>-1)
document.write("<script src=\"xp.js\"></"+"script>")
//--></script>
```

Listing 2

```
if(navigator.userAgent.indexOf("Windows NT 5.1")>-1){
    var d=new Date(), d2=new Date("April 8, 2014");
    document.write("<h1>Only " + \
  Math.floor((d2.getTime()-d.getTime())/604800000)+ \
  " weeks more, and your computer will be overthrown!</h1>");
}
```

I would welcome more suggestions for beginner-friendly Linux distributions that will work on as much Windows XP hardware as possible

I then tell them how to repent in sackcloth and ashes by installing Linux (after taking a backup of their important files, of course, in case it goes wrong). I'll probably change the countdown to a days count somewhere around the 40-days mark to make it more 'proper', and I'll have to think of something to say after doomsday has passed as well.

I urge everyone reading this to consider if there is any website on which they have latitude to place a similar script. It won't disturb most of the readers, and you will be doing a big favour for those who ended up with old XP machines. Of course, if your site is entirely about software development then it's likely that all your readers will be aware of this problem already, but if you have any non-development material (or any material that you send friends and family to) then you could be doing people a favour by warning them. If anyone here works for a major Web portal or search engine then you might be able to perform the favour for even more people.

Additionally, I would welcome more suggestions for beginner-friendly Linux distributions that will work on as much Windows XP hardware as possible (i.e. machines from around the year 2000 onwards), but still have good security support, a reasonable collection of drivers, and a fairly modern WINE setup for running those pesky few Windows applications that it's hard to find alternatives for (e.g. the CD-ROM reference book they like or whatever). Mint fits the latter two requirements quite nicely, and the Xfce version is the least resource-hungry, but I know there are some old laptops that modern Mint Linux can't load its desktop on (you might need an old version of Knoppix or similar instead, which is more of a security compromise although it'll almost certainly still be better than XP).

Finally, if anyone works for an organization currently involved in throwing out a load of XP kit, please try to encourage them to warn the recipients that the OS will need changing. I'd like to see Microsoft themselves put out a final security update to Windows XP which causes the user to be warned in no uncertain terms that their machine is no longer secure, but I doubt if they will: they'd probably think it's not in their commercial interests if it might result in people switching to a non-Microsoft operating system rather than continue to be addicted to Windows and upgrade their PCs once they're fed up with the malware. On the other hand, adding a final warning via an update ought to gain them good publicity points for 'doing the right thing', and the users concerned are not the highest spenders, so perhaps it would be a good commercial move for them after all. ■

References

[1] http://blogs.technet.com/b/security/archive/2013/08/15/the-risk-of-running-windows-xp-after-support-ends.aspx

[2] http://www.linuxmint.com/edition.php?id=113

[3] http://people.ds.cam.ac.uk/ssb22/ (add xp.js to see the source of the XP warning script)

Static Code Analysis

Chris Oldwood wants more than just syntax checking.

There is a certain level of satisfaction to be gained from tracking down a really gnarly bug. If it's a really tough one you might even be inclined to punch the air in a physical gesture that reflects the effort you put into finding it. Of course we'd all wish this was never needed; that we'd always write bug free code. Or at least track it down very quickly during (unit) testing when the code is freshest in our minds.

The antithesis of this is tracking down a bug only to find that a tool could have easily pointed it out for you. I find this kind of bug soul destroying because I know I've just wasted part of my life doing something the very machine I'm programming could have saved me from. Static Code Analysis is one of those tasks that a machine is perfectly suited to as it systematically reviews your code and points out those places where the dragons may be lying in wait.

To err is human

One of the most common mistakes I used to make in my earlier days was the 'assignment instead of comparison' inside an **`if`** statement, e.g. writing **`if (x = 2)`** instead of **`if (x == 2)`**. Another common mistake I used to fall foul of was uninitialised variables. There was always a desire to avoid unnecessarily initialising a variable with a default value despite the danger of ending up with a code path that might use its uninitialised value. The mantra of 'not paying for what you don't use' can lead you to prematurely optimise your code's execution speed at the cost of significantly increased development time, and subsequent hair loss.

There have been attempts to try and write code in a different style as a way of avoiding some of the more common mistakes. One of the most notable is probably the swapping of the arguments in an **`if`** statement so that you would end up trying to assign to a literal value instead of the variable, e.g. instead of writing **`if (x == 2)`** you use **`if (2 == x)`**. But that just makes the code harder to read, and anyway it doesn't work when non-**`const`** variables are involved. The SINGLE ASSIGNMENT pattern [1], which gets an added boost via **`const`** in C++ and **`final`** in Java, is a more human friendly approach to the problem, e.g.

```
const std::string input = getInput();
const size_t index = input.find_first_of('<');

if (index == std::string::npos)
```

The free lunch?

Contorting your code to avoid common coding problems should not be your first line of inquiry. This is even more true today as the very first bit of advice in Steve Maguire's 1993 book *Writing Solid Code*, is this:

> Enable all the optional warnings in your compiler.

Those more familiar with the similarly aged *Code Complete* by the other Steve (McConnell) have to wait a little longer for the same advice, almost until the end of the book (Chapter 26 – Debugging) but it's still there. This should be done in companion with enabling 'warnings as errors' to make sure you don't just try and ignore what the compiler might be trying to tell you. Yes, in many cases the warnings are possibly benign, but if you fail to deal with the issue there-and-then you risk missing the really important stuff in a sea of trivia when it finally rears its ugly head.

The best part about that advice is that it's pretty much free. You've already paid for the compiler (either in pennies or time) and so you might as well put it to good use. In these modern times where there is an abundance of Open Source Software, even if you aren't using it for your production builds, it can still be used as an additional pair of eyes that will cast a second opinion on your codebase.

For many years I was perfectly happy using /W4 /WX with Visual C++ and felt reasonably smug that I was getting some decent static analysis (this was long before they added the extra code analysis). Then, after a fair bit of refactoring I managed to get GCC to at least compile parts of my codebase (not all and it still wouldn't link) and yet I was amazed at how much extra free static analysis I had been missing out on. GCC highlighted another 14 potential issues that Visual C++ was silent on. And this was before I had turned on the standards compliance and portability settings (which I had little need for production-wise). Clearly a propriety company like Microsoft has a vested interest in making money from their extra smarts and with an Enterprise-style budget you may get more out-of-the-box, but it just goes to remind us that the OSS offerings can often play and even outsmart their rivals in the big leagues.

As an aside the main place where the most refactoring was needed was in my COM components. I had a bit of extra work to manually convince the MIDL compiler to throw out something GCC could directly consume, as Visual C++ invokes it automatically, and I had to provide some workarounds for VC++ specific extensions like **`__uuidof`**. But, once that was done it meant a whole lot more code could be put under the microscope. In particular it highlighted a number of places in my error handling where I had switched from a custom string implementation (that had the same memory footprint as a **`char*`**) to **`std::string`** and I was passing it by value to a **`printf()`** style variable args function.

Custom tooling

With potentially so much on offer before getting your wallet out you might be thinking there is little point in spending any money on custom tooling. Aside from the grief I'd get from at least one prominent ACCU member by suggesting that strategy, it's simply not true – there is definitely a lot of value in tools like the venerable Lint. For starters not all languages even need a compiler, e.g. JavaScript, and so that's not even an option. Secondly, even if you do have a compilation step, custom tools like Lint can provide more extensive coverage simply because they're more focused in finding bugs in your code instead of generating binaries. Going back to JavaScript for a moment, Douglas Crockford, author of *JavaScript: The Good Parts*, illustrates nicely in his book how a tool like JSLint is essential due to some pretty ugly language features.

If you think that the cost of a licence for a commercial tool like PC-lint (as opposed to one of the more extortionately priced tools) is too much, consider how much programmer time you've lost through bugs like uninitialised variables or missing/badly written copy constructors. Whilst working for a large bank, I was turned down access to PC-lint because they only had a handful of licences. During that project I know of at least two occasions where programmers had lost an entire day tracking down a bug caused by just such an issue and both developers were paid more per day than a single licence of the software that could have found it for them.

Of course there is a start-up cost in introducing such a tool and that cost is much larger on a legacy system, but as Anna-Jayne Metcalfe showed in her 2008 ACCU London [2] talk, you don't have to jump in and try to fix

CHRIS OLDWOOD

Chris is a freelance developer who started out as a bedroom coder in the 80s writing assembler on 8-bit micros; these days it's C++ and C#. He also commentates on the Godmanchester duck race. Contact him at gort@cix.co.uk or@chrisoldwood

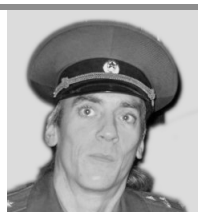
Generating Code From a Unit Test, Part the Second

Richard Polton continues his quest to generate code from tests.

Hello again! Last time we were here I was speculating about the possibility of automatically generating programme code from a suite of unit tests, reasoning that this, really, was the trick that TDD has missed. Clearly, given a full set of requirements and a suite of tests which encode these requirements, we should be in a position to produce a working programme that satisfies the requirements. However, right now, to the best of my knowledge, this is not possible (at least for those of us who use .NET such as myself). When I had thought about this originally I had thought that some complex parser that understands C# (given that C# is going to be the target language) would be required and, rather than fill valuable CVu space with parsers, which we've all seen before, I wondered about adding attributes to the test code and using reflection on the compiled assembly. The problem with this, though, is that you have to be able to compile the code in order to load the assembly and this somewhat defeats the object of the exercise. That is, you would have to implement stubs at the very least for all of the interfaces being tested, and that is exactly what we are trying to achieve automatically!

However, it occurred to me that a parser for C# already exists courtesy of those nice folk at Microsoft. Roslyn [1] is Microsoft's codename for their new .NET compiler toolchain which we are told will become part of the Visual Studio suite at some point. Using Roslyn, it should be possible to extract the required metadata from the unit tests. And so, as soon as I could, I downloaded the latest release of Roslyn and commenced reading. If you want to reproduce my code then you will need to download the Sept2012 CTP [2] (or newer if one becomes available during the writing and

RICHARD POLTON



Richard has enjoyed functional programming ever since discovering SICP and feels heartened that programming languages are evolving back to LISP. He likes 'making it better' and enjoys riding his bike when he can't. He can be contacted at richard.polton@shaftesbury.me

Static Code Analysis (continued)

everything right at the start. You need to start small, enabling only the analysis of the most serious problems and then progress from there. Whilst integration into your build pipeline is an excellent eventual goal, just running it regularly manually should pay early dividends.

Design smells

Once you get passed the hardcore problems you might begin to wonder how much value there is in some of the softer complaints or suggestions the tool makes. ReSharper, which is a tool for the .Net world (although they're moving into the realms of C++) will point out where you can pass an iterator instead of a container, because it can see that you've only iterated it. While both forms are technically correct, given the lack of `const` in C# along with the standard containers being mutable, you're probably taking unnecessary liberties. Similarly I've seen cases where ReSharper spots that methods and even whole classes can be declared static which saves on unnecessary heap churn. The lack of free functions in C# coupled with an attitude that 'static is bad' [3] means that some designs overuse Object-Orientation when a simple function will suffice.

Learning aid

This brings me along nicely to my final reason for using such a tool – it's also a learning aid. As we get more experienced we make far less of the fundamental errors as we begin to remember the things that constantly trip us up, but in the early days we're like a toddler as we keep falling over. Switch to a new language though, and even if we're an experienced developer we can still regress back to that toddler stage if there are enough differences. Even moving to a new project that uses a newer version of the same language can surprise you as you might have forgotten all the cool things they added but you've yet to be able to get practical experience with.

My own move from long term C++ developer to n00b C# developer was helped in part by a similarity in the languages around the basic constructs but also through the use of a static code analysis tool to show me the finer details. More recently the move from .Net 3.5 to .Net 4.0 means that named and optional arguments are now at my disposal which I keep forgetting (although it's the former rather that latter which I'm more interested in).

By far the biggest win though has been in learning LINQ, which can be accessed either directly via the C# extension methods, or through the syntactic sugar the language provides. In the beginning I could write the really obvious SQL-like format pretty quickly but some of the more subtle forms, such as when transforming to a Dictionary, I struggled with. When I saw the tool suggest it could change the structure of my code I first guessed what it might do, then reveal what the answer was. If I didn't get it right I'd switch back and forth with the editor's Undo/Redo command to try and understand what it was proposing.

Even now when I see an error message, say, from the compiler I generally try not to read what the actual error messages says, instead favouring to just jump right to the code to see if I can quickly spot what the compiler has picked up.

Keep control

It's tempting when you see a barrage of issues reported by a tool like this to just go through blindly and accept the changes it suggests on the basis that 'it's always right'. In the case of some of the LINQ transformations I've seen it do I'd definitely question that wisdom. A tool like this should be there to watch your back and do your bidding, don't let it get out of control and start writing your code for you. The code it generates may be syntactically correct but it may not express it in the most human readable way and share your design intent. ■

References

[1] http://en.wikipedia.org/wiki/Assignment_(computer_science)#Single_assignment

[2] Taming the Lint Monster – 20th November 2008.

[3] https://twitter.com/codemonkey_uk/statuses/385518295958683649

Listing 1

```
using System;
using NUnit.Framework;
using Moq;

namespace UnitTestTesting
{
  [TestFixture]
  public class TestsInHere
  {
    [Test]
    public void TestDoIt()
    {
      var a = Mock<AnInterface>();
      Assert.IsTrue(a.DoIt());
    }

    [Test]
    public void TestVerify()
    {
      var b = Mock<AnInterface>();
      Assert.IsTrue(b.Verify(1));
    }

    private static bool Predicate(int c)
    {
      return c>=0;
    }
  }
}
```

Anyway, the Roslyn API exposes a function **SyntaxTree.ParseText()** that accepts a string of programme code and parses it to return a syntax tree which can subsequently be queried.

Looking at our example code, we can see that the interfaces in which we are interested are used in methods, each of which is contained within the same **TestsInHere** class, which are annotated with the **Test** attribute. We would expect the syntax tree to be something like: **Using**, **Using**, **Using**, **Namespace -> Class -> Method**, **Method**, **Method**. For the purposes of this article, we will assume some structure to the test methods. Therefore, we will search for identifiers having the value **Mock** which are contained within variable declarations (instead of temporary instances of **Mock<>**). If we can find these declarations, we should be able to find the variable name associated with the declaration and then search the test method for accesses of that variable name. In this way we can build up a collection of names (properties, methods) that the interface is expected to present. We also assume that we are searching for these accesses within **Assert** statements. First, though, take a look at the syntax tree which we produced by applying the SyntaxWalker presented in [4] in which we have highlighted the nodes of interest. (See Listing 2.)

Armed with the syntax tree, we can issue an XPath-like query to extract all the syntax nodes that represent method declarations that have a '**Test**' attribute. Owing to the syntax for attributes in C#, it is possible for a method to have a list of lists of attributes. For example, we could theoretically be presented with code like

```
[Author="Bob",Place="office"]
[Test,Ignore("Because I want to go home")]
public void ATestMethod() { }
```

where **ATestMethod** has a list of two lists of attributes; **Author** and

Listing 2

```
[-] Roslyn.Compilers.CSharp.CompilationUnitSyntax
[...]
 [-] Roslyn.Compilers.CSharp.***NamespaceDeclarationSyntax*** -- corresponds to 'namespace
UnitTestTesting'
  [.] Roslyn.Compilers.CSharp.IdentifierNameSyntax
  [-] Roslyn.Compilers.CSharp.***ClassDeclarationSyntax*** -- corresponds to 'public class TestsInHere'
   [-] Roslyn.Compilers.CSharp.AttributeListSyntax
    [-] Roslyn.Compilers.CSharp.AttributeSyntax
     [.] Roslyn.Compilers.CSharp.IdentifierNameSyntax
   [-] Roslyn.Compilers.CSharp.***MethodDeclarationSyntax***  -- corresponds to 'public void
TestDoIt()'
    [-] Roslyn.Compilers.CSharp.AttributeListSyntax
[...]
   [-] Roslyn.Compilers.CSharp.***MethodDeclarationSyntax*** -- corresponds to 'public void
TestVerify()'
    [-] Roslyn.Compilers.CSharp.AttributeListSyntax
[...]
   [-] Roslyn.Compilers.CSharp.***MethodDeclarationSyntax***   -- corresponds to 'private static bool
Predicate(int a)'
    [.] Roslyn.Compilers.CSharp.PredefinedTypeSyntax
[...]
```

publication of this article) because there were some API changes which were incompatible with the prior CTP release.

With this all in mind, let us begin. I present below a simple test harness for a set of equally simple requirements. We must produce a programme that implements an interface, called **AnInterface**, which provides two functions, called **DoIt** and **Verify**. **DoIt** is a function that takes no parameters and returns a boolean value that indicates the success or otherwise of the operation. **Verify** is a function that expects an integer parameter and returns a boolean indicating whether the integer conforms to some predicate, to be determined later. Below then, is a non-exhaustive test harness. I make no attempt to cover all possibilities at this point; the purpose of this exercise is to show that it is possible to generate code automatically from the tests provided. (See Listing 1.)

The first thing we need to do is pass the code above to our Roslyn programme. In our case we are going to pass it as a string but if this were for real then we would doubtless be loading a file or something similar.

Place in the first list; and **Test** and **Ignore** in the second. Therefore, the code to extract the **Test**-annotated methods looks more complicated than it really is. (See Listing 3.)

If we pass the syntax tree that describes our example code above then **methodsHavingTestAttributes** contains the nodes in the syntax tree that correspond to the methods **TestDoIt** and **TestVerify**. These

Listing 3

```
var methodsHavingTestAttributes =
   syntaxTree.GetRoot().DescendantNodes().
   OfType<MethodDeclarationSyntax>().
Where(mdecl=>mdecl.AttributeLists.
   Any(attrListSyntax=>attrListSyntax.Attributes.
   Any(attrSyntax=>(attrSyntax.Name as
   IdentifierNameSyntax).Identifier.
   ValueText=="Test")));
```

Listing 4

```
var compilation =
   Compilation.Create("UnitTesting").
AddReferences(MetadataReference.
   CreateAssemblyReference("mscorlib")).
//AddReferences(MetadataReference.
// CreateAssemblyReference("nunit.framework")).
AddSyntaxTrees(syntaxTree);
```

Listing 5

```
var theMockVariableDeclarations =
   testMethodDeclaration.Body.Statements.
   OfType<LocalDeclarationStatementSyntax>().
Select(stmt => stmt.Declaration).
   Cast<VariableDeclarationSyntax>().
Where(varDecl => varDecl.DescendantNodes().
   OfType<GenericNameSyntax>().
   Any(nm => nm.Identifier.ValueText == "Mock"));
```

two nodes are also trees and so the syntax nodes beneath them (in the trees) describe the bodies of these two methods. Given these two nodes, the next step in our algorithm is to identify the interfaces that are being mocked in the tests. Before we can do that, however, we need to create a semantic model of the syntax tree for reasons we shall come to later.

To create the semantic model we must first create a **Compilation**. Let us give it a name, we choose **"UnitTesting"** (no prizes for originality there) and attach **mscorlib** as a reference. As we are not currently aiming to establish the meaning of all the symbols, we won't include the NUnit Framework just yet. That may come in a future article. For now, we want to parse the syntax tree and find all the methods that have a **Test** attribute. Add the syntax tree and lo! we have a **Compilation** (Listing 4).

Having created our **Compilation**, we can then create a semantic model of the code.

```
var model =
   compilation.GetSemanticModel(syntaxTree);
```

And so with that little diversion behind us let us continue in our quest to extract the interface and methods.

Having calculated **methodsHavingTestAttributes**, we loop through the sequence and, for each method having a **Test** attribute, we find all of the variable declaration nodes in the method body which are declaring a variable of generic type **Mock<>**. (See Listing 5.)

In the test methods in our example, we find the nodes containing the declarations **var a = Mock<AnInterface>();** and **var b = Mock<AnInterface>();** As **theMockVariableDeclarations** contains the tree of syntax nodes which describes the variable declaration, we need to navigate down the syntax tree to the literal **Mock** token again, bearing in mind that there may be multiple identifiers in the variable declaration. From this point we know the relative location of the generic type parameters and so we read them into the **MockedTypes** property in our helper class **NameAndInterface**. We can also assume that we know the relative location of the variable declarator node and so we traverse back up the tree (in a neither-robust-nor-generic manner, see I told you there would be certain assumptions made didn't I?) so that we can determine the name of the mock variable itself. (See Listing 6.)

In our example, **varNamesAndInterfaces** will contain **"a"** and **{"AnInterface"}** for **TestDoIt()** and **"b"** and **{"AnInterface"}** for **TestVerify()**.

Now that we know the names of the variables which have been declared to have our **Mock<>** type in our test method, we need to search for all the call sites in the code where they are accessed so that we can determine the set of properties and methods that are being called in the test method and are, therefore, expected to be present in the interface. The nodes we are looking for (yes, these are the nodes you are looking for) are **MemberAccessExpressionSyntax** nodes so let us find all nodes in the syntax trees for each of our test methods which are **MemberAccessExpressionSyntax** nodes. From these we can

Listing 6

```
var varNamesAndInterfaces =
   theMockVariableDeclarations.
SelectMany(varDecl => varDecl.DescendantNodes().
   OfType<GenericNameSyntax>()).
Where(nm => nm.Identifier.ValueText == "Mock").
Select(mock => new NameAndInterface
{
  Name = (mock.Parent.Parent.Parent as
     VariableDeclaratorSyntax).
     Identifier.ValueText,
  MockedTypes = mock.TypeArgumentList.Arguments.
     OfType<IdentifierNameSyntax>().
     Select(ident => ident.Identifier.ValueText)
});
```

Listing 7

```
var variableAccesses = testMethodDeclaration.
  Body.DescendantNodes().
  OfType<MemberAccessExpressionSyntax>().
Where(acc => (acc.Expression as
  IdentifierNameSyntax).Identifier.
  ValueText == varName);
```

exclude any member access nodes which do not belong to the variables we have discovered. So for a given variable **varName**, the nodes which access it are as shown in Listing 7.

Taking the **variableAccesses** container we can determine the names of the property or method that is being accessed.

```
var functionNames =
   variableAccesses.Select(acc => (acc.Name as
   IdentifierNameSyntax).Identifier.ValueText);
```

For our two example test methods, we expect the **functionNames** to contain **{"DoIt"}** and **{"Verify"}** respectively. Okay, so for our test method we have found all the **Mock<>** variables and all the property and method names that the interface is expected to supply. Now we need to determine the number and type of each of the parameters expected by each of the methods. This code snippet is not very sophisticated. We have assumed that the method will have only one parameter even though we have written it as a container operation. Of course, we could dynamically generate the name of each of the parameters but for simplicity we have chosen to call the parameter **p**. (See Listing 8.)

Having calculated the parameter list, we need to determine the expected return type of the property or function. For the purposes of this article we assume that only **Assert.IsTrue** is possible and therefore we can deduce that the return type must be boolean. Were we to do this in a more general way, we would make use of the semantic model that we

Listing 8

```
var invocationExpr = accesses.First().Parent as
   InvocationExpressionSyntax;
// Could call model.GetTypeInfo(invocationExpr)
// but none of the names involved are defined so
// the function returns ErrorType, ie unknown
var invocationArgList =
   invocationExpr.ArgumentList as
   ArgumentListSyntax;
var invocationArgs = invocationArgList.Arguments;
var functionParameterTypes =
   invocationArgs.Select(argSyntax =>
{
  var argType =
     model.GetTypeInfo(argSyntax.Expression).
     Type;
  return Syntax.Parameter(Syntax.Identifier
     ("p")).WithType(Syntax.ParseTypeName
     (argType.Name));
});
```

Listing 9

```
var callingAssertionExpr =
   invocationExpr.Parent.Parent.Parent as
   InvocationExpressionSyntax;
   // Assert.IsTrue(a.DoIt())
// Need SemanticModel here to determine type of
// the assertion function parameter. For now, we
// know that IsTrue expects a bool
var n = (callingAssertionExpr.Expression as
   MemberAccessExpressionSyntax).Name.
   Identifier.ValueText;
var returnType = Syntax.ParseTypeName
   (n == "IsTrue" ? "bool" : "void");
```

constructed earlier in order to find the types of the expressions. The syntax tree only shows us the tokens that are supplied and the grammatical types of these tokens.

From the invocation expression node we navigate up the tree to find the assertion function that is calling. In the first test method this is `Assert.IsTrue(a.DoIt())`. Given the limited set of assertion expressions available it would be possible to build a simple lookup table containing assertion procedure and expected parameter types although the more generic approach of using the semantic model is to be preferred.

Listing 10

```
yield return
   new InterfaceNameAndMethodDeclarations
{
  Name = mockedInterfaceTypes.First(),
  MethodDecls = new[]{
  Syntax.MethodDeclaration(returnType,
    functionNames.First()).
    WithModifiers(Syntax.TokenList
      (Syntax.Token(SyntaxKind.PublicKeyword))).
    WithTfmIfTrue(
    t => t.WithParameterList(
      Syntax.ParameterList(Syntax.
         SeparatedList<ParameterSyntax>(fpt,
         Enumerable.Repeat(Syntax.Token
         (SyntaxKind.CommaToken),
         fpt.Count - 1)))),
      () => functionParameterTypes.Any()).
         WithSemicolonToken(Syntax.Token
         (SyntaxKind.SemicolonToken))}
};
```

Without the semantic model we recognise that `IsTrue()` expects a boolean parameter. Therefore we know that the return type of our tested function or property must be bool. (See Listing 9.)

Listing 11

```
var interfaceProgramCode =
   interfacesAndMethodDecls.Select(
o => Syntax.InterfaceDeclaration(
   attributeLists: new
     SyntaxList<AttributeListSyntax> { },
   modifiers: Syntax.TokenList
     (Syntax.Token(SyntaxKind.PublicKeyword)),
   typeParameterList: null,
     //TypeParameterListSyntax,
   baseList: null, //new BaseListSyntax(),
   constraintClauses: Syntax.List
     <TypeParameterConstraintClauseSyntax>(),
   identifier: Syntax.Identifier(o.Key),
   members:
Syntax.List<MemberDeclarationSyntax>(o.ToList().
   electMany(i=>i.MethodDecls))));
```

Listing 12

```
var interfaceProgramCode_asString =
   interfaceProgramCode.Select(pc =>
   pc.NormalizeWhitespace().ToFullString());

interfaceProgramCode_asString.ToList()
Count = 1
  [0]: "public interface AnInterface\r\n
  {\r\n    public bool DoIt();\r\n
    public bool Verify(Int32 p);\r\n}"
```

Then, armed with the function name, the parameter list and the return type we construct the syntax tree for the method declaration and add it to the results container. (See Listing 10.)

As a last step in the syntax tree parsing, we group all of the discovered interfaces and methods by the interface name.

```
interfaceMethodDecls.GroupBy(ifn => ifn.Name);
```

Finally we constuct a new syntax tree containing the discovered interface together with its properties and methods (see Listing 11), and the resulting programme code is in Listing 12.

Don't just take my word for it though, download the code [3] and run it yourself. You never know, it might even work :-D

Also, let me point you to a couple of other related sites I found while battling with the Roslyn API documentation. They should give you more launch points if you are thinking about writing some Roslyn code of your own or even if you only want to look at some other people's code that makes use of Roslyn. Anyway, you could do worse than visit these URLs in references [5], [6] and [7]. In addition, there are plenty of questions of StackOverflow [8]. Even I asked one.

Next steps in no particular order:

- Generate an implementing class (easy),
- Generate function signatures with distinct parameter names (easy),
- Parse the unit tests to establish the rules that are being tested so that general statements can be made about the expected behaviour of the methods being tested (less easy),
- Generate function bodies that satisfy the tests in the implementing class (also less easy),
- Use the semantic model to determine the function return types instead of a hard-coded lookup table (relatively easy),
- Remove some, or all, of the assumptions made about the structure of the test code and replace them with tree-parsing code (not impossible).

References

[1] Roslyn http://social.msdn.microsoft.com/Forums/vstudio/en-US/home?forum=roslyn

[2] Roslyn Sept 2012 CTP release http://msdn.microsoft.com/en-us/vstudio/roslyn.aspx

[3] Google Code prototype-builder https://code.google.com/p/prototype-builder/

[4] Syntax Walker http://www.amazedsaint.com/2012/07/bending-your-code-like-anders-with-c.html

[5] Using the Roslyn Syntax API http://blogs.msdn.com/b/csharpfaq/archive/2011/11/03/using-the-roslyn-syntax-api.aspx

[6] Custom refactoring http://visualstudiomagazine.com/articles/2012/03/15/roslyn-ctp-custom-refactoring_1.aspx

[7] Implementing a Code Action http://blogs.msdn.com/b/csharpfaq/archive/2012/02/06/implementing-a-code-action-using-roslyn.aspx

[8] StackOverflow: Roslyn http://stackoverflow.com/search?q=roslyn

Why We Need To Reconsider How We Do Technical Presentations

Dirk Haun makes a case for doing presentations differently.

Technical presentations aren't TED talks. With all the facts and figures that you have to present, it's impossible to make them exciting. Besides, the audience expects to see all those technical details and will frown upon slides that only show photos.

Right?

No, of course not. The perceived lack of actual content is a common misunderstanding of how modern and more visual presentations really work and what they can accomplish. Presentation Zen, probably the most influential modern presentation style, is not really about showing pretty pictures – it's about giving the audience what they need, but in a way that helps them remember the content and that encourages them to act on the message of the presentation.

it's about giving the audience what they need, but in a way that helps them remember the content and that encourages them to act on the message

In the tech world especially, presentations are used to transfer knowledge and information. So we have this urge to put all the information that we have about a topic up on our slides, to make it available to our audience.

However, there's scientific evidence that this approach doesn't work; people simply can't read and listen at the same time. So your audience will have to decide what to do – which usually means that they're going to read. It's up there on the screen, so it must be important, right?

Also, if all the information you want to pass on with your talk is on your slides – why are you standing in front of them? It would be more effective to send the information by email and save the audience the travel cost.

In other words, trying to transfer information this way is rather inefficient.

There's another aspect to consider: People will soon forget the majority of what you told them. 30 days after your presentation, they will have forgotten about 90% of the content. These numbers actually date back to research published by the German psychologist Herman Ebbinghaus in 1885. So it's not a problem of our modern age; it's how the human memory works.

All this bad news should make it clear that we need to reconsider our approach to technical presentations. Your talk should still include all the information, but it should be presented in such a way that your audience listens to you, the speaker, instead of trying to read your slides. In order to help them remember, you can provide them with visual hooks, since our brains are better at remembering visuals – and the emotions they evoke.

In these days of the omnipresent internet search, it's easy to find the information you need with only a few keywords; that's all your audience needs to remember.

Mind you, your talk should still make a strong case for your topic. You still need to research it and prepare it in a way that convinces your audience that the topic is relevant. But you don't need to put all that information up on your slides.

The tradition for tech talks is to make the slides available after the talk. Slides of the more visual kind obviously aren't going to help the audience much then. But that's okay. As we've seen, the traditional bullet point-laden slide isn't good as a slide during a presentation. It turns out it isn't good at helping people look up things afterwards either.

The typical slide from a typical tech talk is what Garr Reynolds refers to as a "slideument": A strange hybrid of a slide and a document, that doesn't really work as either of them. So it only makes sense to separate them entirely: Have visually oriented slides that support you during your presentation; and if the audience is of the sort that insists on having the hard facts being made available to them, provide a handout; i.e. a proper document. Or you could post a write-up of the talk on your company's website (which has all sorts of other benefits, including more traffic to your website). Or you could simply record your presentation and make the video available online.

To summarise, it's time to say goodbye to the old bullet point-laden slides in tech talks. If you really want to make an impact with your audience, your slides need to take a step back, into a supporting role. Free them from all the clutter that could just as easily be found with a simple web search. Instead, focus on preparing your topic for your target audience: What do they already know about the topic? What do they expect from you?

Yes, it's more work doing presentations this way. The presentation is important to you, isn't it? Then it should (and will!) be worth the effort. ■

This article was originally written for CTOvision.com and subsequently featured on May 1st, 2013, on that website.

DIRK HAUN

Dirk Haun is an Open Source enthusiast interested in presenting, learning, and continuous improvement. He currently serves as a non-executive member on the ACCU Committee and can be contacted at dirk@haun-online.de

Standards Report

Mark Radford reports the latest from the C++ Standards meetings.

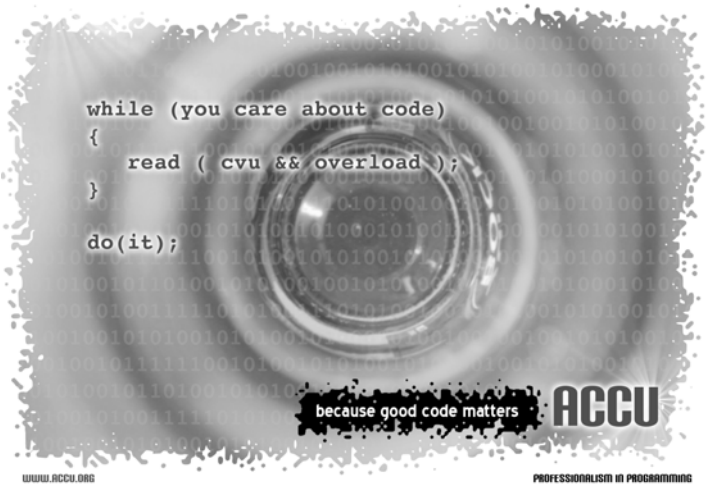
Welcome to my latest standards report!

Before I start talking about recent events, let me just tie up a loose end from my previous standards report. You may remember that I talked a bit about the SG1 (the Concurrency and Parallelism study group) meeting held in Santa Clara. At the time the minutes of that meeting had not been published in a mailing, but I was assuming they would be in the next mailing. This has now happened: these minutes were published in the pre-Chicago mailing and can be found in document N3709 (see below).

Since last time two face to face C++ standards events have taken place: The BSI C++ Panel met on 16th September, and the international ISO committee met in Chicago (23rd-28th September). Also, the pre-Chicago mailing has been published (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/#mailing2013-09). At the time of writing this report, the post-Chicago mailing has yet to be published, so some of the papers I mention will not be publicly available yet. Each ISO meeting has a wiki for that meeting, and papers not in the pre-meeting mailing can be posted on the wiki so they are available at the meeting. I have seen it mentioned on one of the mail reflectors that the deadline for the post-Chicago mailing was October 11th, so I can't imagine it will be very long now.

Naturally the dominant topic for the BSI C++ Panel meeting was the forthcoming Chicago ISO meeting. Much of the day was taken up with discussing the national body ballot comments. Much discussion time was taken up with the matter of **`std::future`** (i.e. its destructor potentially blocking when the future is returned from **`std::async`**), as a number of the national body comments are about this. The biggest problem surrounding this whole issue is that none of the potential solutions are very popular. One solution is to just deprecate **`std::async`** and introduce a replacement with a different return type. However (as someone pointed out) the correct name for a function that does what **`std::async`** does, is '**`async`**'. Therefore, if **`std::async`** were to be deprecated, what would we call the replacement? More about deprecating **`std::async`** below.

Another topic of discussion was the paper N3747, 'A Universal Model for Asynchronous Operations'. There are papers already out there containing proposals to extend **`std::future`**: N3634 proposed the addition of a member function '**`then()`**' which would allow a continuation to be supplied, while N3650 is about resumable functions. These proposals set a clear direction with **`std::future`** being the standard way to handle asynchronous operations in C++. In N3747, author Christopher Kohlhoff argues that there are some disadvantages with tying C++ in to an approach using only **`std::future`**. He makes the case that **`std::future`** is not the only approach, arguing for a universal approach that uses both futures and callbacks.



Moving on to the Chicago ISO meeting...

The issue of **`std::future`** returned from **`std::async`** took up most of the Monday afternoon SG1 discussion time. While there was support for deprecating **`std::async`**, doing so in C++14 was ruled out. The majority support was for deprecating it, and introducing its replacement, post C++14 i.e. aiming for C++17. However, there was a consensus for adding wording (to the working paper) clarifying the current **`std::async`**/**`std::future`** behaviour; Herb Sutter has written a paper containing suggested wording in N3776 (one of those not yet published outside the standards committee). A formal motion to incorporate N3776 into the working paper was passed.

Since the Bristol meeting, another concurrency/parallelism topic SG1 has started to discuss is vectorisation (see the Wikipedia article if you want to read an explanation [1]). In Bristol, 'A Proposal to add Single Instruction Multiple Data Computation to the Standard Library' (N3571) was discussed, but did not get a consensus to move forward. However, in Chicago two new papers were included in the discussions: 'SIMD Vector Types' (N3759) proposes a library solution, while 'C++ Needs Language Support For Vectorization' (N3774) argues that vectorisation needs support from the language/compiler, and a lack of such support incurs a considerable performance cost of between a factor of 2 and 4. Note that N3774 is not yet published outside the standards committee.

The paper 'A Parallel Algorithms Library' (N3724) proposes an extension to the C++ library providing access to parallel execution for a broad range of algorithms. At sixty-seven pages, it is a very long paper. I mention it here because, with N3724 also proposing support for vectorising algorithms, there is overlap with the vectorisation papers I talked about above. SG1 discussed N3724 on Thursday afternoon, the result being a strong consensus (nobody was against it) for turning the paper into the working draft for a TS.

Finally, returning to N3747, which I've already mentioned, above, when talking about the BSI Panel meeting. Author Christopher Kohlhoff was unable to be in Chicago, but his paper was presented by other members of the UK delegation. The objective was to determine if there was enough interest to make it worth the author doing more work in this area. Following some discussion, a straw poll revealed the answer to be 'yes', with fifteen of those present in favour, and none against!

Reference

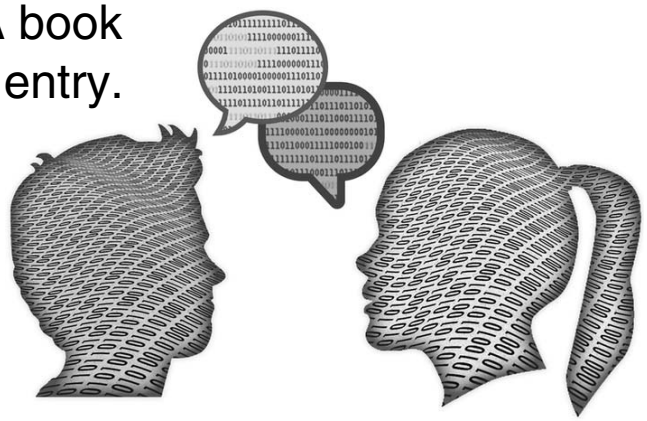
[1] http://en.wikipedia.org/wiki/Vectorization_%28parallel_computing%29

MARK RADFORD

Mark Radford has been developing software for twenty-five years, and has been a member of the BSI C++ Panel for fourteen of them. His interests are mainly in C++, C# and Python. He can be contacted at mark@twonine.co.uk

Code Critique Competition 84

Set and collated by Roger Orr. A book prize is awarded for the best entry.



Please note that participation in this competition is open to all members, whether novice or expert. Readers are also encouraged to comment on published entries, and to supply their own possible code samples for the competition (in any common programming language) to scc@accu.org.

Last issue's code

I want to split the set {1,2,3,...,21,22} into two disjoint subsets of 11 numbers such that neither subset contains three consecutive numbers. The following was an attempt to produce one possible subset. Having got an ordered sample of 11 from {1,2,3,...,22} I check that the sample does not contain three consecutive numbers (y[i]-y[i-2] is never 2), nor are there three consecutive numbers **not** in the sample (y[i]-y[i-1] is never greater than 3). But it produces samples meeting *neither* condition. Help!

The code is in Listing 1.

```
import random
x=range(1,23,1)
y=random.sample(x,11)
y=sorted(y)
for i in range(2,11,1):
   if y[i]-y[i-2]==2 or y[i]-y[i-1]>3 or y[1]-y[0]>3:continue

print y
```

Listing 1

Critiques

Juan Antonio Zaratiegui Vallecillo a.k.a. Zara <zara@coit.es>

This CC is too easy: You are testing a condition and doing nothing with the test result.

Using the condition as written, we can use its result to select a new list of integers:

```
import random
x=range(1,23,1)
found=False
while not found:
   y=random.sample(x,11)
   y=sorted(y)
   found=True
   for i in range(2,11,1):
      if y[i]-y[i-2]==2 or y[i]-y[i-1]>3 or y[1]-y[0]>3:
         found=False
         break

print y
```

Now it works as required.

Russel Winder <russel@winder.org.uk>

Well, where to start? Obviously this is a Python 2 script and it should be Python 3! We can deal with this by making the `print` a function call rather than a statement. Of course there is so much else wrong with this code that the Python version is just the beginning.

The specification mentions sets but there are no sets anywhere in the code, everything is lists. Agreed the test for acceptability of a subset is best handled by using a sorted list but the result should be a set not a list.

The lines:

```
y=random.sample(x,11)
y=sorted(y)
```

should (arguably) either be:

```
y=random.sample(x,11)
y.sort()
```

or:

```
y=sorted(random.sample(x,11))
```

the rebinding of `y` in the original is not wrong but is potentially misleading.

In a similar (improve the code presented) vein, the code uses the three parameter range function with third parameter `1`, which is the default. It should just use the two parameter range function: it's more Pythonic not to mention values that are defaults.

Then there is the `for` loop which is clearly a long-winded way of doing nothing: the loop iterates making a check and then moving to the next iteration. This is of course at the heart of the failing of the code, there is no difference in control flow whether the expression is true or false. (Given this is Python that should, of course, be `True` or `False`.) Dare a I say, classic misuse of `continue`?

The code sequence tries to create one subset, but in fact both subsets need checking independently to see if the constraints on a successful result are met.

So trying to construct a solution, the start point has to be creating a predicate to check whether an iterable (could be a set, list, tuple,…, but we expect a set with its uniqueness property) meets the condition of not having three or more consecutive numbers. Perhaps something such as:

```
def isAcceptable(x):
   l = sorted(x)
   for i in range(2, len(l)):
      if l[i] - l[i-2] <= 2:
         return False
   return True
```

Then if the requirement is only to find the first acceptable solution involving some random selection, then perhaps the following suffices to deliver up a pair of sets:

ROGER ORR

Roger has been programming for over 20 years, most recently in C++ and Java for various investment banks in Canary Wharf and the City. He joined ACCU in 1999 and the BSI C++ panel in 2002. He may be contacted at rogero@howzatt.demon.co.uk

```
def partition(original, sampleSize):
    while True:
        putative_1 = frozenset(sample(original, sampleSize))
        putative_2 = original - putative_1
        if isAcceptable(putative_1) and isAcceptable(putative_2):
            return putative_1, putative_2
```

The script is then

```
if __name__ == '__main__':
    print(partition(frozenset(range(1, 23)), 11))
```

If randomness is not required then perhaps just do less work:

```
print({1, 2, 5, 6, 9, 10, 13, 14, 17, 18, 21, 22}, {3, 4, 7, 8, 11, 12, 15, 16, 19, 20})
```

This solution is correct by construction. Not to mention I actually tested it. Talking of which, a final point, there should be tests for this, best place to encode the requirements! Everyone will have noted that my proposed solution has been structured for testability and, indeed, has been tested. Code available on request.

Commentary

This program suffers from one fairly obvious problem; the `if` statement does not actually affect the control flow. However this check simply verifies whether the initial random sample fails to satisfy the required conditions, an additional loop and a little more framework is needed to try another random sample until a successful solution is found.

The algorithm itself is slightly complicated as it only tests one of the sample sets and infers the other set contains no runs of three consecutive numbers by checking there are no large gaps in the existing sample. Unfortunately the test doesn't work at the end points; so if the other set starts {1,2,3,…} or ends {…,20,21,22} the program will incorrectly report success. It is possible to fix this, but I suspect the approach Russel takes of doing the simpler test for consecutive numbers on **both** the sets is a much safer option, unless and until (a) performance is shown to be a problem and (b) the alternative algorithms is shown to be faster.

Of course, the while program does somewhat assume a successful solution is *possible* otherwise the resultant program will loop forever. Perhaps in this case it is sufficiently obvious; if not it might be necessary to 'place an elephant in Cairo' [1] to ensure the algorithm terminates.

Additionally, simply trying another random sample could be slightly inefficient if solutions are sparse as the algorithm may well result in trying permutations already covered before – and the result is unlikely to be different a second time (!). An initial random selection followed by a cycle through all the possible permutations might be slightly better if the overhead is proved to be significant.

Reference

[1] http://en.wikipedia.org/wiki/Elephant_in_Cairo

The winner of CC 83

This critique was deliberately relatively simple to try and encourage some new entrants; so I welcome Zara to the list of published code critiques.

Zara's critique was correct and his example does now produce correct output (apart from the boundary condition I mention above), but perhaps needs a little more explanation to help the writer of the code understand what they've done wrong.

Russel gave a more complete critique, although his solution doesn't actually mention 'random' (there is (presumably) an implicit **from random input sample** at the start of the script.) Russel also refactored the code into a couple of helper functions, which I feel makes it easier to understand and also allows the possibility of testing the parts of the algorithm.

So I have awarded Russel the prize for this critique.

Late critique for CC82

We have had a late entry for Code Critique 82. The code is in Listing 2, to save you hunting.

Balog Pál <pasa@lib.hu>

The provided code presses hard to abuse name lookup, reusing names the same identifier `i` and `l` hoping it is still valid due to hiding rules. It actually succeeds in some places, say exploiting that the name of a function argument hides properly. (Interesting that we even missed one usual trick, constructor of `l` could name the argument `x` and in the initializer list `x(x)` would have the proper behaviour in isolation...)

However it went one step too far. In the standard we have this rule:

> **3.3.7 Class scope [basic.scope.class]**
>
> 1 The following rules describe the scope of names declared in classes.
>
> 1 The potential scope of a name declared in a class consists not only of the declarative region following the name's point of declaration, but also of all function bodies, default arguments, exception-specifications, and *brace-or-equal-initializers* of non-static data members in that class (including such things in nested classes).
>
> 2 A name N used in a class S shall refer to the same declaration in its context and when re-evaluated in the completed scope of S. No diagnostic is required for a violation of this rule.
>
> 3 If reordering member declarations in a class yields an alternate valid program under (1) and (2), the program is ill-formed, no diagnostic is required.

We have `x` at the global scope and also as a member in `l`. At definition of `l::load`, `x` is used and fits description of bullets 2) and 3). So this program is ill-formed, but the compiler is allowed to not even tell about it.

The answer to 'what it would print' is thus kind of moot. A better quality compiler would still issue the diagnostic and not produce anything to execute. For the rest, as the behaviour is only defined for well-formed programs, anything can happen, including printing something or nothing, or even posting the correct answer to this the SCC entry.

Listing 2

```
#include <iostream>

int x;

struct i
{
  i() { x = 0; }
  i(int i) { x = i; }
};

class l
{
public:
  l(int i) : x(i) {}
  void load() {
    i(x);
  }
private:
  int x;
};

int main()
{
  l l(42);
  l.load();
  std::cout << x << std::endl;
}
```

The 'how to deal' part is simple: just stop lookup abuse for good. Actually the main issue is trivial to mitigate with some naming convention, like name global/namespace variables with **g_** prefix and forbid that form for anything else. Having that we can still construct problematic case with anonymous namespace or clashing global types against data members, carefully choosing expressions that are still work (maybe adding **decltype()** to the mix), but the problem potential will get way too low to worry about.

[Ed: I'm not persuaded 3.3.7 applies to **i(x)** since when the body of load is evaluated the meaning of **x** is unambiguous, whatever the ordering of the member declarations. However Pal's general direction about avoiding ambiguity and the potential dangers of ill-formed programs still stand.]

Code critique 84

(Submissions to scc@accu.org by Dec 1st)

I want to use a data structure as a key in a map, but it isn't working as I expect. I've simplified the code down to a shorter example that shows the problem – can you help explain why?

I'm expecting to get output of:

```
Testing k1 {c,42,y} => c42y
Testing k2 {p,57,x} => p57x
```

But I (sometimes!) get output like this instead:

```
Testing k1 {c,42,y} => c42y
Testing k2 {p,57,x} =>
```

but I don't understand why. I've added some extra output at the end but I still don't see it.

The code is in Listing 3.

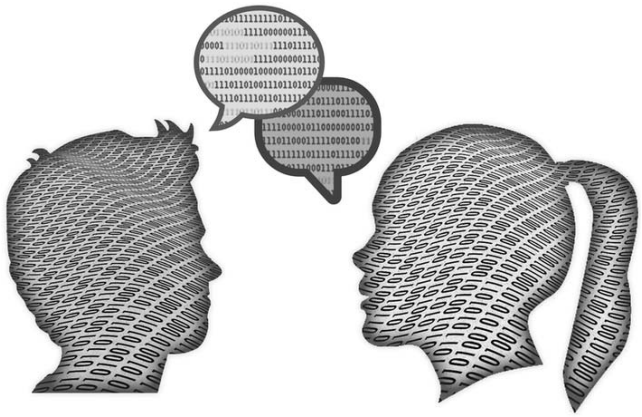
Listing 3

```
#include <iostream>
#include <map>
#include <memory.h>
#include <sstream>
#include <string>

// This is the Key -- it is "plain ole data"
struct Key
{
  char ch;
  int i;
  char ch2;
};

typedef std::map<Key, std::string> Map;

std::ostream & operator<<(std::ostream & os,
  Key const & key)
{
  std::cout << "{" << key.ch << ","
    << key.i << "," << key.ch2 << "}";
  return os;
}

bool operator<(Key lhs, Key rhs)
{
  return memcmp(&lhs, &rhs, sizeof(Key)) < 0;
}

// Add an item to the map
void add(Map &map, Key key)
{
  std::ostringstream oss;
  oss << key.ch << key.i << key.ch2;
  map[key] = oss.str();
}
```

Listing 3 (cont'd)

```
// Add an item to the map
void add(Map &map, char ch, int i, char ch2)
{
  Key key;
  key.ch = ch;
  key.i = i;
  key.ch2 = ch2;

  add(map, key);
}

int main()
{
  Map map;

  Key k1 = {'c', 42, 'y'};
  add(map, k1);

  add(map, 'p', 57, 'x');
  Key k2 = {'p', 57, 'x'};

  std::cout << "Testing k1 " << k1 <<
    " => " << map[k1] << std::endl;
  std::cout << "Testing k2 " << k2 <<
    " => " << map[k2] << std::endl;

  // Testing the first element
  Key k = map.begin()->first;
  std::cout << "First: " << k <<
    " => " << map[k] << std::endl;

  k = map.rbegin()->first;
  std::cout << "Last: " << k <<
    " => " << map[k] << std::endl;

  std::cout << "Contents of the map:\n";
  for (Map::iterator it = map.begin();
    it != map.end(); ++it)
  {
    std::cout << it->first <<
      " => " << it->second << std::endl;
  }
}
```

You can also get the current problem from the accu-general mail list (next entry is posted around the last issue's deadline) or from the ACCU website (http://www.accu.org/journals/). This particularly helps overseas members who typically get the magazine much later than members in the UK and Europe.



Norfolk Developers Conference

Paul Grenyer presents the Norfolk Developers Conference.

Naked Element Ltd. is proud to present NorDevCon, the Norfolk Developers one day Agile and tech conference. The conference will take place on Friday 28th February 2014 at the Kings Centre in Norwich. Building on the hugely successful Agile and technical tracks from this year's SyncConf, NorDevCon will also feature a cloud and big data track, a workshop track and a combined local speaker and SyncDevelopHER (bringing together women in IT) track. The hugely successful conference dinner will be back as well as a reception hosted by Virgin Wines.

To be kept up-to-date with NorDevCon, please join the mailing list:

http://eepurl.com/BZ-an

and follow the conference on twitter: @nordevcon

The speakers

NorDevCon has a fantastic speaker line up in 2014. Some of the most popular speakers from this year's SyncConf will be returning along with plenty of new speakers from around the country and from the local area. We're still working on the opening keynote speaker and hoping to secure a popular figure from the software craftsmanship community.

Nat Pryce and Steve Freeman, the authors of *Growing Object Oriented Software Guided by Tests*, will be closing the conference. Their book took the software community by storm with its outside-in approach to automated testing.

After speaking at SyncConf, Kevlin Henney, Phil Nash, Ian Robinson (Neo Technology) and Norwich favorite Liz Keogh will all be returning to Norwich for NorDevCon. Local speakers include Danielle Ashley, Dom Davis (Virgin Wines), Janet Randall (Aviva) and Pete Roome (ex Pandr).

The workshop track will see the return to Norwich of Jon Jagger's CyberDojo, an F# workshop from recent Norfolk Developers speaker Phil Trelford (Trayport), who will also be doing a presentation session on F#, and Russel Winder who will be providing hands on experience of the Spock testing framework.

Chris O'Dell (7digital) will be speaking about continuous delivery, Anthony Saxby (Microsoft) will be speaking in the new cloud and big data track, Rachel Davies (Unruly Media) will be returning to Norwich to tell us about Agile at Unruly Media and Jon Skeet (Google) will be speaking about C#.

More speakers will be announced as they are confirmed.

The venue

The King's Centre is a high quality conference venue in the centre of Norwich with 14 different rooms to suit individual requirements. The centre offers conference rooms, meeting rooms, breakout rooms and a 650 seater auditorium. The auditorium is pictured at the top of the page.

The conference dinner

The conference dinner will be held at the venue in the evening following the conference. In this unique experience the speakers remain seated while the conference attendees move round between courses. This is your opportunity to speak to your favorite speakers of the day. The price

includes three courses and two glasses of wine per person. There will also be a bar.

Last year the conference dinner was one of the highlights of the conference and sold out! Please make sure you purchase your dinner ticket at the same time as your conference ticket.

The Virgin Wines reception

Virgin Wines will be hosting a reception at the venue between the end of the conference and the start of the conference dinner. As well as a glass of wine courtesy of Virgin Wines there will also be a bar. Places are limited so please make sure you get your free Virgin Wines reception ticket at the same time as your conference ticket.

Sponsors

Once again we are getting a lot of support from technical organisations in the form of sponsorship. Confirmed sponsors for NorDevCon include:

Virgin Wines

Neo Technology

Norfolk Tech Journal

There are currently still some sponsorship packages available:

- Associate £250

  Logo on slide and mention during intro

  Tweets & Mentions from @NorDevCon and @NorfolkDev

  Logo on website

PAUL GRENYER

Paul Grenyer is a husband, father, software consultant, author, testing and agile evangelist. He can be contacted at paul.grenyer@gmail.com

View from the Chair

Alan Griffiths
chair@accu.org

It was a different millennium when I joined the ACCU. In fact it wasn't even called the ACCU at that time. While the internet existed and some people had internet we were a long way from having instant access to the wisdom and knowledge that today we can get from a quick search on a phone.

The primary sources of information were manuals, books, magazines and journals but I found that many of the articles available were useless to me – they were targeted at an audience that were happy to have a program work at all. I was past that point. There were a very few books that attempted to address the question of 'how to do it well' (as opposed to 'how to do it at all') but they too were hard to find as they were probably not stocked at the local book-store (and, unlike now, there were no online reviews).

What I discovered with the ACCU was unique: other people who were interested in developing their craft and sharing what they discovered. Techniques, cool hacks, recommended books and tools poured through this channel of communication. Firstly through the C Vu Journal, then through email, Overload, the conference and the Mentored Developers programme.

Another thing that I gained was to make associates and friends with many of these people with whom I shared the journey. This was a great benefit to me (as the people I worked with lacked the same joy I found in finding better ways to do things).

Now most of the journals and magazines have disappeared and in their place are websites. I can find new techniques and cool hacks if they are described in terms that occur to me when using a search engine. Books too are easily found on the internet – along with recommendations and even previews. Tools can be downloaded at the click of a button.

If we wish to simply to share the journey of becoming a better developer then that too can be found on the internet via, for example, blogs.

What is left for the ACCU?

Static Code Analysis (continued)

- Partner £1500
  Logo on slide and mention during intro
  Tweets & Mentions from @NorDevCon and @NorfolkDev
  Logo on website
  Banner
  Exhibit

- Elite £3000
  Logo on slide and mention during intro
  Tweets & Mentions from @NorDevCon and @NorfolkDev
  Logo on website
  Banner
  Exhibit
  Speaking Slot (to all conference attendees)

If you are interested in sponsoring NorDevCon, please email nordevcon@nakedelement.co.uk.

Naked Element Ltd

Naked Element Ltd. is a made to measure software development services provider. Whether you need complex, enterprise-level software integrations, a made to measure web application or a mobile app, we have the experience and skills to meet your needs.

We work with clients at all organisational levels, to advise on strategies and to implement solutions.

Tickets now available

NorDevCon promises to be even bigger and better than this year's SyncConf. I hope you're looking forward to it as much as we are! Tickets went on sale on 1 November 2013 at:

http://nordevcon2014.eventbrite.co.uk/

Web: nordevcon.com

Twitter: @nordevcon

Mailing list: http://eepurl.com/BZ-an

Norfolk Developers

We are very lucky in Norwich to have a thriving and highly active tech community.

Complementing the existing tech community, Norfolk Developers is peeling back the high level and going straight to the heart of software development practices and processes. It has already brought a number of national and international speakers to Norwich and there are plans for local speakers and workshops in the near future.

You might want to try something new or wish to take your software development up to the next level. Norfolk Developers is jumping depth-first into the detail and bringing you practical value. We make it easy for you to access knowledge, progress and prosper in the highly specialised and valuable field of software engineering.

Norfolk Developers was founded by Paul Grenyer (Director at Naked Element Ltd, SyncNorwich co-founder and Norfolk Tech Journal founder), Dom Davis (lead developer at Virgin Wines) and Ben Taylor (CEO at Rainbird Technologies Ltd.). Meetups usually take place on the first Wednesday of the month and are usually held in the offices of sponsors Virgin Wines. Each meetup boasts two 60 minute presentations.

nor(DEV):