the magazine of the accu

www.accu.org

{cvu}

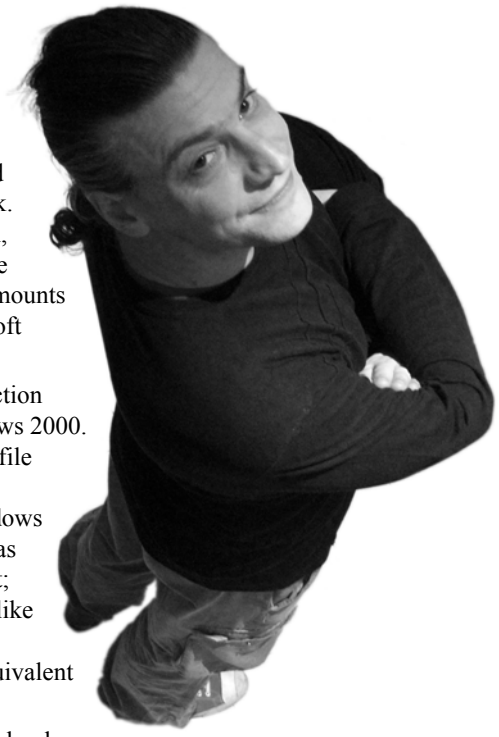
Volume 28 • Issue 3 • July 2016 • £3

Features

Regulars

{cvu}
Volume 28 Issue 3
July2016
ISSN 1354-3164
www.accu.org

Fixed fixation



I've recently been frustrated on a few occasions trying to install software for Microsoft Windows. I have a PC with an adequately sized SSD as a system disk for improved boot time, and a much larger ordinary hard disk. Most application installers give you the opportunity to choose a location for the installed software, and I routinely install to the larger disk. Some installers don't. This is frustrating enough, but some installers give you the option to choose your own directory, and then still install huge amounts of files to the system disk anyway. Yes, Microsoft Visual Studio, I'm looking at you.

The accepted fix for this seems to be to use Junction Points, a facility introduced for NTFS in Windows 2000. These behave in much the same way as Un*x's file system links (which have been around for much longer), and are a great improvement over Windows shortcuts. The trouble is, Junction points aren't as nicely integrated into the Windows environment; you cannot manage them by default in UI tools like Explorer, and the command line interface is seemingly deliberately different to the Un*x equivalent – ln.

It's not just a drive-letter issue, although this is clearly so out-dated that tools such as Junction Points and logical drive assignment have arisen to escape the origins of being a way of managing physical devices on – let's be honest – less than capable file systems. Despite the well-publicized limitations on the length of the PATH environment variable in Windows, the default installation directory for Windows is 13 characters long (19 for 32 bit apps). This, coupled with installers that don't give you the option to choose an installation location that has a shorter name (bin, for instance, is a common choice), or software that insists on residing in a specific location, leads to frustration and annoyance.

There is a wider issue of sensible defaults, especially for user-interface design, which will have to wait for another time. Until then, thank you for listening.

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 28.4 1st August 2016
C Vu 28.5: 1st October 2016

Overload 135: 1st September 2016
Overload 136: 1st November 2016

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

Testing Private

Paul Grenyer shows that testing doesn't need public access.

A friend and former colleague of mine, Chris 'Frankie' Salt, recently popped up on Facebook messenger and asked me a question:

> I wonder if you'd mind answering a Java question for me? It's more of a best practices thing. So, encapsulation vs availability of methods for testing. Splitting your code into functions makes it a lot more readable and it makes sense to make these private as they will only ever be used once. However unit testing demands access to these private methods, I know there are ways around this but I was interested in your opinion.

Me? Have an opinion about unit testing? Many stranger things have happened!

Encapsulation is all about hiding code away so that you can change it with minimal or no impact on other parts of the code base which use it indirectly. You shouldn't (ever) compromise encapsulation for the sake of testing. Every private method you write must be callable from at least one public method or via a chain of other private methods which is ultimately called from a public method. Otherwise, reflection shenanigans aside, it would never get called at all.

Does that mean reflection is the solution to testing private methods? No. It's a great tool for poking values into objects which are initialised using reflection at runtime, so that you don't have to add a public non-default constructor or public setters, but beyond that it should be avoided for testing.

Two solutions sprang to mind, triangulation and sprout classes.

Triangulation

I first encountered the term triangulation in Kent Beck's TDD [1] book, although it was a practice I had been using for some time. Triangulation is the practice of passing different values into the public methods of a class in order to test the non-public parts of the class. This is probably best demonstrated with a (contrived) example. Take a look at the **`Invoice`** class in Listing 1.

It has two private methods, **`addVat`** and **`vatMultiplier`**. **`addVat`** is used to add VAT to the total value of all of the line items. It uses **`vatMulitplyer`** to convert the VAT rate, which is passed into the invoice as a percentage, into **`value`** which can be multiplied by the gross total to get the net total.

As a conscientious developer you would want to write tests which test how **`addVat`** and **`vatMultiplier`** work in general cases, such as 20% and corner cases such as 0%. This can be achieved using triangulation and passing different VAT percentages to an invoice, adding some line items and asserting the result of calling **`netTotal`**, for example as in Listing 2.

In a simple example like the **`Invoice`** class, testing using triangulation is probably sufficient. However, it leaves me with an uncomfortable feeling that the calculation of VAT probably isn't really the responsibility of the **`Invoice`** class and any more than two test methods for VAT in the **`Invoice`** test class feels wrong. Of course you can write another test class which just tests the VAT parts of the **`Invoice`** class, but that doesn't feel right either.

Sprout classes

I first encountered sprout classes in Michael Feathers' *Working Effectively with Legacy Code* [2] book, although it was a practice I had been using for some time. A similar technique, called *Extract Class*, is described by Martin Fowler in his *Refactoring* [3] book. The basic idea is that you extract parts of one class and put them into a new class which is instantiated and used from the first class. Let's take a look at this in the context of the **`Invoice`** class (see Listing 3).

Listing 1

```
public class Invoice
{
  private final List<LineItem> lineItems
    = new ArrayList<LineItem>();
  private final double vatPc;

  public Invoice(double vatPc)
  {
    this.vatPc = vatPc;
  }

  public void add(LineItem lineItem)
  {
    lineItems.add(lineItem);
  }

  public double grossTotal()
  {
    double total = 0;
    for(LineItem lineItem : lineItems)
      total += lineItem.getValue();
    return total;
  }

  public double netTotal()
  {
    return addVat( grossTotal() );
  }

  private double addVat(double value)
  {
    return vatMultiplier() * value;
  }

  private double vatMultiplier()
  {
    return (100 + vatPc) / 100;
  }
}
```

`addVat` and **`vatMultiplier`** have been removed and replaced with the instantiation of the **`Vat`** class and a call to its **`add`** method. The **`Vat`** class looks like Listing 4.

The responsibility for calculating VAT has been moved away from the **`Invoice`** class and into the **`Vat`** class. Having a separate **`Vat`** sprout class with a public **`add`** method also means that it can be tested in isolation away from the **`Invoice`** class. What was previously the private **`addVat`** method on the **`Invoice`** class is now a public method on the **`Vat`** class which can be tested directly. The **`Vat`** class still has a private method, **`multiplier`**, but it can be easily tested using triangulation. Writing more tests also feels more comfortable (see Listing 5).

PAUL GRENYER

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

Listing 2

```
@Test
public void totalNetWithLineItems()
{
  Invoice invoice = new Invoice(20);
  invoice.add(new LineItem(50));
  invoice.add(new LineItem(20));
  invoice.add(new LineItem(20));
  invoice.add(new LineItem(10));
  assertEquals(120, invoice.netTotal(), 0.01);
}

@Test
public void totalNetWithLineItemsAndZeroVat()
{
  Invoice invoice = new Invoice(0);
  invoice.add(new LineItem(50));
  invoice.add(new LineItem(20));
  invoice.add(new LineItem(20));
  invoice.add(new LineItem(10));
  assertEquals(100, invoice.netTotal(), 0.01);
}
```

Listing 4

```
public class Vat
{
  private final double rate;

  public Vat(double rate)
  {
    this.rate = rate;
  }

  public double add(double value)
  {
    return multiplier() * value;
  }

  private double multiplier()
  {
    return (100 + rate) / 100;
  }
}
```

Finally

By using triangulation or sprout classes or a combination of the two you can fully and easily test private methods without the need to compromise encapsulation or the design of public interfaces. Which to use depends on the complexity of the private methods being tested and/or the number of different tests that need to be written. When you have simple private methods which require few test cases, triangulation can be sufficient. As the complexity and number of test cases increases, sprout classes become a better solution.

Regardless of how you decide to test private methods, keep your interfaces clean, don't change them for the sake of testing and measure your test coverage.

The example code is available here: https://bitbucket.org/pjgrenyer/testing-private

References

[1] *Test Driven Development* by Kent Beck. ISBN-13: 978-0321146533

[2] *Working Effectively with Legacy Code* by Michael Feathers. ISBN-13: 978-0131177055

[3] *Refactoring: Improving the Design of Existing Code* by Martin Fowler. ISBN-13: 978-0201485677

Listing 3

```
public class Invoice
{
  private final List<LineItem> lineItems
    = new ArrayList<LineItem>();
  private final double vatPc;

  public Invoice(double vatPc)
  {
    this.vatPc = vatPc;
  }

  public void add(LineItem lineItem)
  {
    lineItems.add(lineItem);
  }

  public double grossTotal()
  {
    double total = 0;
    for(LineItem lineItem : lineItems)
      total += lineItem.getValue();
    return total;
  }

  public double netTotal()
  {
    return new Vat(vatPc).add( grossTotal() );
  }
}
```

Listing 5

```
@Test
public void vatAt20pc()
{
  assertEquals(120, new Vat(20).add(100), 1);
}

@Test
public void vatAt15pc()
{
  assertEquals(115, new Vat(15).add(100), 1);
}

@Test
public void vatAt17_5pc()
{
  assertEquals(117.5, new Vat(17.5).add(100), 1);
}

@Test
public void vatAt0pc()
{
  assertEquals(100, new Vat(0).add(100), 1);
}

@Test
public void vatAt100pc()
{
  assertEquals(200, new Vat(100).add(100), 1);
}
```

The Codealow

Pete Goodliffe presents a new software soliloquy.

The parents among the *C Vu* readership will no doubt be aware of *The Gruffalo* [1]. Many of us have read this book over and over (and over) again to excited children who enjoy the meter of the writing, the humour, and the cunning triumph of the humble protagonist in the face of larger aggressors.

If only life was like that. Especially the life of the humble jobbing programmer.

Who says life can't imitate art? To prove this, here is a short code parable, presented for your enjoyment, with apologies to Julia Donaldson. I first presented this at the ACCU 2016 conference in Bristol.

The Codealow

A DEVELOPER'S BEDTIME STORY

A dev started work on some deep, dark code.
A bug saw the Dev, and the Dev looked good.
Come and waste time in my erroneous ways;
With the fun I can give, You'll be stuck here for days.

That's terribly kind of you, bug, but no;
I'm off to fix a codealow.

A codealow? What's a codealow?
A codealow! Why didn't you know?
It has terrible branches, executed at will
And disastrous uses of do/while/until.

I'd not heard of that, and it sounds very scary.
I'd best run and hide or I'll look ordinary!
And, saying that, the bug turned and resigned,
Leaving code behind that worked just as designed.

Silly old bug, doesn't he know
There's no such thing as codealow?

On worked the Dev, with a satisfied smirk.
A team lead saw our Dev, thought he needed more work.
Put down that editor, don't write to that log!
Come take some tasks from my growing backlog.

I'd love more to do. I'd gladly help, though
I've plenty of work with this huge codealow.

A codealow? What's a codealow?
A codealow! Why didn't you know?
It has multi-thread access prone to data race,
And foul unused methods all over the place.

"No time to refactor!" The team lead then cursed
And went to find other devs' lives to make worse.
As he scuttled off our dev chortled with glee;
The cunning ruse worked, he was management-free.

Silly old manager, doesn't he know
There's no such thing as codealow?

On worked the Dev, having fun with his code
As an ominous feeling appeared and then growed.
Things that seemed perfect had gradually soured.
Strange bits of logic made our hero turn coward.

What is this weird code... how now to keep coping?
Who wrote this drivel, and what were they smoking?

It has functions so lengthy you'd get lost for days,
And logic that weaves in remarkable ways.
It has surprising behaviour that no one predicts,
And dense, turgid code, that no one tries to fix.

It has legacy parts that are old as the hills,
Whose evil required both malice and skill.
The cohesion and coupling have become a disaster;
There's no way to get at the clean code I'm after.

The sad thing the source control points out quite clearly:
There's only one person who spent time in here, and he
Ought feel shame-faced, he ought to feel guilty.
The nefarious culprit who made this mess is: me!

Oh help! Oh no! I wrote a codealow!

At this monstrous mistake the dev ran in despair
And learnt that next time he must code with more care.

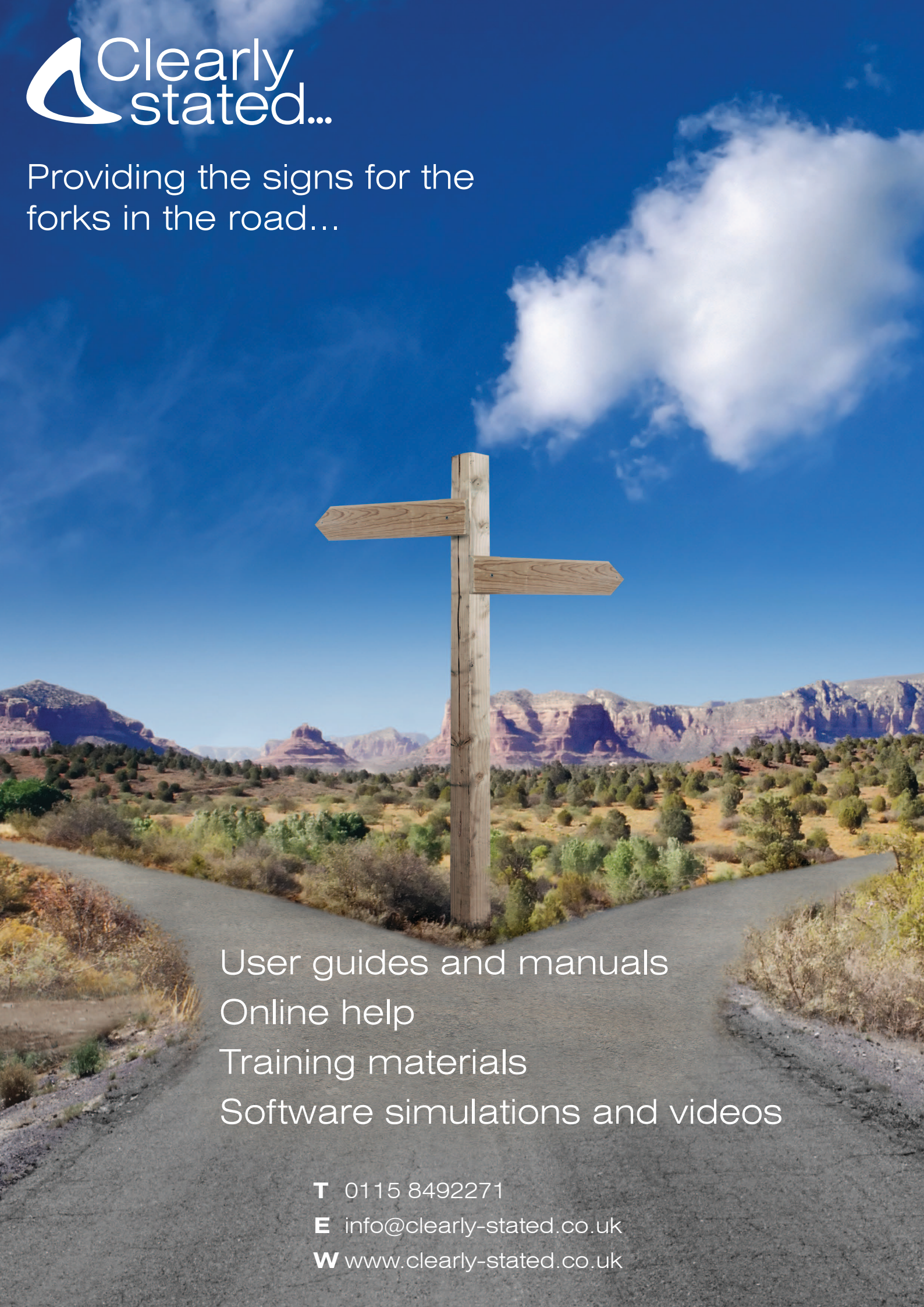
A dev started work on some brand new code.
The dev wrote a test, and the test was good.

Reference

[1] *The Gruffalo*. Julia Donaldson, Axel Scheffler
ISBN: 9781509804757

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


Clearly stated...
Providing the signs for the forks in the road...
User guides and manuals
Online help
Training materials
Software simulations and videos
T 0115 8492271
E info@clearly-stated.co.uk
W www.clearly-stated.co.uk

Whiteboards

Chris Oldwood makes a case for collaboration over technology.

They say a picture is worth a thousand words and in software development there are plenty of words to be said about a great many things from what we're building to how we're building it. The old adage 'a stitch in time saves nine' is a cornerstone of the agile methodology as we struggle to ensure that the team (customer, developers, testers, etc.) are all on the same page before we start laying down code.

In the modern world of software development where there is a bias towards 'working software over comprehensive documentation' the written word is starting to give way to higher bandwidth conversations, most notably just talking to each other face-to-face when the need arises. But words alone are not enough as not everyone can understand a concept just through verbal communication, you can't just talk slower, and louder, and hope that the other people will just 'get it'. The moment a conversation begins to deteriorate is (if you've not already done so) probably a great time to double the lines of communication and open a channel to the visual senses.

In an earlier instalment of this column [1] I touched briefly on drawing diagrams, but from the perspective of using my notebook as the canvas. Whilst it has some benefits, for example the permanency, I've found that I would much prefer the larger, more transitory nature of the whiteboard if it was on offer. What I've come to discover is that I was resorting to using my notebook due to the absence of a better tool (the whiteboard), and because it was a much better tool than the alternative (Visio & SharePoint) for the nature of the message I wanted to convey.

Digital optimisation

One of downsides of working in The Enterprise is that large organisations are obsessed with exploiting economies of scale and this is very noticeable when you look at the workspace allocated to teams. Open plan offices with banks of desks ensure the maximum density of people and can feel somewhat like a battery farming approach to software delivery. In a continued drive to replace anything physical with its virtual equivalent there is no longer any room for luxuries like whiteboards or Kanban boards. Presumably information persisted as 'electronic bits' are so much more versatile than physical media, so why would you want to accept anything less?

The rationale for digitising everything in the office is I suspect predicated on a false assumption that doing anything more than once is waste and must therefore be eliminated. The fact that you have not understood what it is that needs doing, and will therefore build the wrong thing, is secondary to the needs of appearing to make progress. By removing the physical tools we use to collaborate we remove the impetus to do so and the whiteboard is one of the largest and most accessible reminders we have available. I can think of no other tool that draws (no pun intended) us in quite like a whiteboard covered in ad hoc sketches and notes.

Whilst the notion of hot-desking might be great in theory, in practice a team that is going to spend any considerable amount of time together needs a place to call home. This spot needs a certain amount of permanence, ideally closely located to its dependencies, with some surfaces on which the team can draw and attach its information radiators. If you're unlucky enough to be sitting in one of the rat runs that develop when you have a naïve desk layout then you can use the board as a barrier at one end to dissuade through traffic. If you have noisy neighbours you might be able to use it as a baffle too. Just be mindful that any physical barrier you put up radiates information outwards to ensure you don't appear sectioned off and hostile towards collaboration with outsiders.

Dumb and smart boards

Like most things these days, even the humble whiteboard comes in every form from the simplest to the all-singing, all-dancing, Internet Enabled tm version. Your basic large board on wheels is probably the most common breed sighted in the wild as it's easily moveable, and therefore shareable and large enough to be usable, i.e. hold more than one scribble at a time. Some offices have re-discovered the benefits of such a medium and gone all out and turned every wall into a writeable surface. At one company I worked up you could never find anyone at their desk because they were always at a break-out room or huddled around some random patch of wall chatting and drawing pictures.

For those offices that prefer zero-clutter (which usually means a clean desk policy and nothing on the walls except art or a picture of the CEO) there is the temporary solution – drive-by whiteboards. These are rolls of white plastic film which are statically charged and can therefore be hung on most surfaces, e.g. painted walls and windows. They can stay up for a surprisingly long time and also easily be taken down and re-hung elsewhere. Rather than capture their content in another form they are so cheap that you could just keep the original sketches, at least until you know for sure it's worth transcribing them formally elsewhere. Apart from a few surprising surfaces where they don't work the main thing to watch out for is ensuring that you don't disturb some other team's peace and tranquillity by pitching up on some free window space near them for an ad hoc design meeting. Whilst you might enjoy attempting to subvert the organisation by getting stuff done, the complaints from the wrong neighbours may perversely strengthen the argument for virtual collaboration.

One of the earliest electronic whiteboards I got to use was essentially a whiteboard with a scanner and printer welded on the side. You wrote on the board as normal and then when you were done you pushed a button and the white plastic sheet rotated under the scanner/printer combo like a giant roll of fax paper to produce your 'hard copy'. Since then the humble board has gotten much smarter, so much so that it's no longer a physical canvas but a virtual one. Now you draw fake lines, boxes, circles and arrows with (coloured) fake pens and create your sketch in the alternate reality. As if that wasn't enough your scribble can be added to your SharePoint archive under the guise of a OneNote document where you can relive this masterpiece time and time again.

Only no one does. In the same way that JIRA brings your product development into the electronic age with a promise of taking away all the fuss and nonsense of shuffling around post-it notes, so the electronic whiteboard comes with similar dreams and aspirations. And yet there is still so much friction to working with computer-based tooling that it's easier to just do it the old fashioned way. Just as the daily stand-up becomes dominated by the time it takes to access the electronic task board on a shared screen, so a quick design meeting becomes bogged down in bizarre networking problems and pens that feel unresponsive.

Don't get me wrong they are very clever pieces of kit and they can do some pretty neat tricks around transcribing text and recognising shapes. For example where you might previously have had to rub out and redraw

CHRIS OLDWOOD

Chris is a freelance programmer who started out as a bedroom coder in the 80's writing assembler on 8-bit micros. These days it's enterprise grade technology in plush corporate offices. He can be easily distracted via gort@cix.co.uk or @chrisoldwood

parts of a diagram on a physical board to make room you can just select an area and shrink or move it about. If you think your database should have been on the left, or drawn in red ink just change it. And if you think your sketch is destined for a greater part in the world then it has tools to clean it up and allow you to pretend that you drew it in Visio all along.

Of course, in the meantime whilst some companies have been developing smarter boards, others have been developing smarter apps to help give your physical doodles some permanence. For some, just using the camera software on your phone will be more than enough, whilst others might want to clean it up slightly (e.g. fix the white balance) before emailing or uploading to their team's Slack channel or wiki. As you might expect there are a number of free apps for modern phones that are optimised for taking photos of whiteboards which helps reduce the friction a little more, e.g. Office Lens. Just bear in mind that it might be quicker to draw it again when the need arises than shave the yak.

Notation

It's easy to get hung up on notation, but don't. Back in the early '90s during The Notation Wars the three amigos (Booch, Jackobson & Rumbaugh) called a truce and worked together to product the one notation to rule them all – the Unified Modelling Language (UML). Whilst this caters for a whole slew of different diagram types each with its own sets of symbols, you'll rarely need this in practice to sketch out a concept; in fact most of the time you can get away with nothing more than some badly drawn squares and a few arrows.

there are a number of free apps for modern phones that are optimised for taking photos of whiteboards

Being a graphic artist is not a prerequisite for being able to communicate with simple sketches, yes you might have to redraw bits if they go really wonky, but that's the beauty of a whiteboard – rub it off and try again. If you're drawing some kind of rough architecture diagram and you want to be more adventurous you still only really need three symbols: a rectangle, a tall cylinder (for a database) and a rotated cylinder (for a message queue). You could even throw in a dotted line if you have a need to distinguish more than one relationship. The one thing I do think is important is that every arrow means the same thing – don't mix-and-match dependencies with data flows as it's confusing (the arrows usually just point in opposite directions).

A couple of different coloured pens will naturally allow you a little more freedom too, such as when you have namespaces and containers to group things, but as a rule if you find yourself needing lots of different colours (or trying to add hatching to a diagram) you've probably gone over the top. The main thing to watch out for though is to keep your pens well segregated! When you're working with both a whiteboard and post-it notes, it's all too easy for a permanent marker to infiltrate the whiteboard pen drawer and then you've got a problem. I think it's worth sticking with well-known brands, e.g. buy Sharpies for your stickies and Staedtler pens for your whiteboard, as you'll instantly know if it's in the wrong pile. Also once whiteboards become popular again make sure you have a drawer with a seriously heavy-duty lock to keep them and your board rubber safe from those less well stocked in the stationery department.

As with any technical diagram you should really stick to the same level of abstraction throughout, e.g. don't try and mix classes into a high-level architecture sketch. Simon Brown [2] has a really nice concept that he calls C4 which stands for Context, Container, Component and Class. I've found this is a useful aide memoire when thinking about what it is I want to draw. The static structure of a system is fairly easy to draw using only rudimentary skills and basic shapes, but when it comes to something like sequence diagramming the swim-lanes approach of UML is as good a notation as any. UML may be much derided for leading many down the rabbit hole of Big Design Up Front, but the U in UML (unified) does at least mean that should you choose to embrace a richer notation, it's one most people should be likely to know.

Words and pictures

Although I've focused heavily on the more pictorial uses for whiteboards we shouldn't forget that they are more useful than that – you can write words on them too. Okay so they're not necessarily the best thing for writing lengthy prose, not least due to the unsightly stain you'll probably acquire on the edge of your palm, but there's nothing wrong with using the board to help you create lists of stuff. Or you could use the top corner to write up a point of focus (aka sprint goal) for the next few days to continuously remind the team about what over-arching business event or theme they should be collectively working towards.

Even though you might be decomposing your work into epics, stories and tasks, there is still a need to flesh out details at the point of implementation. For instance you may have defined some loose conditions of satisfaction [3] at an earlier planning stage, but now that you're ready to work on the story in earnest it would be useful to lay down a more formal set of acceptance criteria, perhaps with the other 3 amigos (BA, developer and tester). Naturally the board is a good place to work out a list of happy and sad paths that need to be catered for. Or maybe you need to sketch out something around the UI, perhaps the developer and tester need to factor some kind of test API into the design, or someone from ops needs to see how this new feature affects the monitoring of the system. A story is just a placeholder for this conversation.

Whilst index cards are nice for arranging on a desk, they don't stick very well to walls, unlike Post-it notes. I've known places though where you aren't allowed to stick Post-it notes on windows and their walls are painted with some kind of anti-sticky-note paint, so the board may have to double up as both a task board and design tool. Personally I've always been fonder of arranging things on a vertical surface than horizontal, but I'm sure mileage varies greatly here.

The one thing I never use the board for is writing code. Apart from an interview where I'm being specifically asked to write code on a board, I'd always do it at a terminal using a tool custom built for the job – the text editor.

Write on / wipe off

The humble whiteboard is becoming an endangered species in the corporate world. It's a sad moment seeing one sat in the corner of an office with an architecture diagram from years ago almost burnt into it because the pen ink has had so long to dry out. Almost every day for a board should be different as people come and go, stopping only briefly to chat, sketch, write, arrange, sip coffee and reflect before moving on. These bursts of activity will be interspersed with periods of serenity, before once again the slate is swiped clean and the hullabaloo of collaboration punctuates the air once again. Whiteboards are for living. ■

References

[1] *C Vu* 25-4, http://www.chrisoldwood.com/articles/in-the-toolbox-pen-and-paper.html

[2] http://www.codingthearchitecture.com/2014/08/24/c4_model_poster.html

[3] http://stackoverflow.com/questions/3697466/can-you-clarify-the-differences-between-conditions-of-satisfaction-cos-and-acc

How to Block Russia From Your Website (and why you might want to)

Silas S. Brown takes a stand against indiscriminate legislation.

It's 2025. ACCU publishes a negative review of a poor-quality programming book, a book that happens to be popular in Russia. The book's publishers are big and powerful; they say the review is unlawful, because it is criticising popular opinion, thus causing disunity. That now counts as 'extremism'. *C Vu* and *Overload* are added to the Federal List of Extremist Materials, ACCU is banned as an extremist organisation, and all ACCU members in Russia are imprisoned for 20 years due to their affiliation with this supposed terrorist threat.

That sounds ridiculous, but it's legally possible. According to the Parliamentary Assembly of the Council of Europe (2012), Russia's 2006 amendment to its 2002 Federal Law on Counteracting Extremist Activity (originally enacted in response to the 9-11 attack) removed the need for a group to be involved in 'violence or calls to violence' before it can be counted as extremist. It's now extremist simply to promote any kind of 'discord', which is vaguely defined.

By the time this issue of *C Vu* goes to press, Russia will likely have finished banning a couple of minority Christian groups whose literature includes negative reviews of the Russian Orthodox Church. But there's nothing in the legislation that restricts such bans to religious differences. Russia has basically outlawed the bad review, and anybody who recommends any journal that publishes bad reviews about anything popular will, according to the vague wording of the law, be subject to criminal conviction as a terrorist.

I'm reminded of Niemoller's 'First They Came' poem [1]. The most prominent group they're going for at the moment seems to be Jehovah's Witnesses [2], but if you feel smug because you don't like JWs anyway, how are you going to feel when the big software companies start talking to the Prosecutor's Office in the same way that the Russian Orthodox Church talks to them today?

But Russia is a lot bigger than I am, and there doesn't seem to be much I can do about it, especially as I don't want to become a politician or anything like that. But I do have a website with free software on it – it may be a very small thing, but I decided to block Russia and display a message saying I won't share my software with them because of this.

Now, when I say 'block', I don't really mean it in the sense they do. I don't really want to stop ordinary Russian citizens from using my software. I just wanted to make them have to jump through hoops to get it, in order to raise a little awareness. So I implemented the block as a piece of Javascript that plonks a big black box over the page. You can remove it by disabling Javascript, or by using browser developer tools to edit the DOM and delete that node, or by using a proxy, or by disabling its display in a user stylesheet, or any number of other ways. Furthermore the text underneath is still findable on Russian search engines (I'm hoping they won't figure out how to interpret the box as a kind of 'cloaking' and down-rank the page for it).

The basic change is a surprisingly simple addition to an Apache htaccess file, requiring no support for server-side scripting:

```
<Files typography.js>
  ErrorDocument 403 http:// <url-of-russia-version>
  .js
  Order allow,deny
  Allow from all
  Deny from ru
</Files>
```

where `typography.js` [3] is a piece of Javascript that gets included at the bottom of all my pages, basically to replace straight ASCII quotes and dashes with their nicer typographical equivalents if and only if the browser is known to support such (some old mobile and terminal-mode browsers still don't, and my site still 'gracefully degrades' to ASCII, at least on its English pages; I wasn't so worried on my Chinese pages because I've never seen a system that displays Chinese but not curly quotes).

The htaccess rule works by telling Apache to deny access to this one file for any domain ending .ru, and instead send the browser to another URL, which hosts a different version of the script that also includes the big black box (a `div` element with style `"position:fixed"` and `"height:100%; margin:5em; left:0px; top:0px; z-index:9"`; colour as you see fit). The easiest way to 'hack' it is:

```
if(document.createElement) {
  d=document.createElement('span');
  d.innerHTML='<div style="..."> ... </div>';
  document.body.appendChild(d)
}
```

It's important that the `ErrorDocument` line specify an absolute URL (even if it's served from the same place); if instead it just specifies a file, Apache will return that file with HTTP status code 403, and browsers like Chrome won't execute the Javascript body if the HTTP status code indicates an error. Since I'm not in Russia, I had to test this by temporarily placing my own address on the `Deny` line.

For this to work, Apache has to do a reverse DNS lookup on the IP address of anybody trying to retrieve that `typography.js` file, and this must complete before the page is served. It could take some time if the user's ISP is one of the increasingly rare beasts that doesn't set reverse DNS entries for its IP addresses. That's not a major issue for the user if the only consequence is your straight quotes don't become curved quotes for a while (just make sure that script is the last thing to load, in case they're using HTTP pipelining). But it will hold up one of the Apache server threads, which could become an administrative issue if many such requests are received at the same time. (This won't be a consideration if your web pages are served from an institution that uses a modern lightweight single-threaded server such as nginx instead of Apache, although its configuration would of course be different.)

Also it's entirely possible that some ISPs in Russia have IP addresses that don't resolve to .ru domains. I wasn't worried about that; if you want to do a more thorough job, it's possible to get a list of known Russian IP address blocks, but your Apache configuration will end up being quite large and probably slow things down for everybody, plus you'd have to periodically update that list.

Of course not every bad review is correct, and many reviews published by minority groups are biased to say the least. But intelligent readers should be able to decide for themselves about such things; outlawing non-violent negative reviews as 'extremist', and prosecuting anyone who mentions them, could have far too many unintended consequences. Besides raising

SILAS S. BROWN

Silas S. Brown is a partially-sighted Computer Science post-doc in Cambridge who currently works in part-time assistant tuition. He has been an ACCU member since 1994 and can be contacted at ssb22@cam.ac.uk

Debugging – What Has Changed in the Last Decade?

Neil Horlock travels through time in search of bugs.

Ten years ago I read an interesting article in *CVu* entitled 'A review of debugging tools'. It introduced me to the concept of reversible debugging, and to a company called Undo Software, whose founders Greg Law and Julian Smith co-authored the piece. The idea of reversible debugging was understood well enough but the tooling was not where it needed to be. But my interest had been piqued and I kept track of the technology's progress.

Obviously, a lot has happened in the last decade when it comes to software development. Therefore I thought it would be a good time to review what has changed in debugging, particularly reversible debugging, over that time, and how it relates to people developing and deploying large systems in the real world.

To start with, let's look at what software life was like ten years ago. I was working for the same international bank as I am now, and I had similar issues to many other people in my position. The software my team developed was mission-critical and highly complex. Therefore, stability was and remains a key tenet of our production platforms and as such having good tools and strong practices is part of our bank culture. Then, standard operational practice in case of any issues was of course to review the ubiquitous Log files. These gave a glimpse of what was happening, hopefully the right glimpse, but certainly not the full picture, making it difficult to see exactly what had been happening before a crash occurred.

Again, like a lot of my peers, the sheer size and complexity of the environment made finding bugs more difficult. As an international bank we had a highly dispersed team, with development taking place in various centres around the globe and often being remote from the actual deployment of the product. Everyone would be using different machines with varying specifications, adding another challenge when trying to recreate bugs from machine to machine in order to track them down. Anyone who has debugged code will fully understand that a lot of time and resource can be spent running code that has crashed once, waiting for it to repeat itself to get a better understanding of what had caused the issue.

This is what first got me interested in reversible debugging. For those that haven't come across it, the concept is simple. Reversible debuggers enable developers to record all program activities (every memory access, every computation, and every call to the operating system) and then rewind and replay to inspect the program state. This colossal amount of data is presented via a powerful metaphor: the ability to travel backward in time (and forward again) and inspect the program state. This makes it much simpler to pinpoint what was happening in the run up to a bug striking, and hence fix problems faster.

Fast forward to 2016

So what has changed in the last decade? The issues of complexity, time zones, and different machines for development/deployment remain, and have been joined (and exacerbated) by five others.

1 Greater requirements

Back in 2006, a European bank would be typically trading with 10–15 different markets across the continent. That figure has grown to 80 or more in the intervening decade. The roughly eight fold increase in the number of systems we have to interface with has led to a growth in the resources we have at our disposal – however, it has increased pressure on productivity and introduced a need to automate many more activities to make the best use of our resources. Everyone one will no doubt recognise the need to find ways to do much more with similar or lower headcounts and budgets.

2 Higher staff turnover

Software developers are highly prized, and competition to recruit and retain them is fierce. Inevitably this means that these days team turnover is higher than it might have been in the past. The knock-on effect of this is that a person debugging the code may be less familiar with the software itself and its place in the scheme of things. Therefore they can't rely on innate knowledge to understand 'how something happened' – this is much harder without the deeper experience that comes with time spent working with particular systems.

3 Software development has become more mature

Ten years ago concepts such as agile and unit testing were buzzwords that were beginning to gain traction. They are now a core part of what we do, with all our processes built around standard practice. This is obviously a major step forward for the industry, as we now rely on more mature engineering practices. However, it also puts the spotlight even more firmly on debugging across the whole build. Software is much more complex, meaning that a series of interactions can trigger a bug. You therefore need to have a record of the complete chain of events if you want to speed up solving the issue.

NEIL HORLOCK

Neil Horlock is a C++ programmer with an interest in high-performance computing. He is a member of the BSI panel for C++ and member of the SG14 low latency study group for ISO C++.

How to Block Russia From Your Website ... (continued)

awareness of this, I hope this article is also useful for any programmer who needs to place a temporary message for a particular region on a webserver without needing server-side scripting (which not all administrators enable). ■

References

[1] http://en.wikipedia.org/wiki/First_they_came_...
[2] http://www.jw.org/en/news/legal/by-region/russia/jw-religious-freedom-threatened/
[3] http://people.ds.cam.ac.uk/ssb22/typography.js

reversible debugging has really gained momentum

4 Moore's Law – for good and bad

Thanks to Moore's Law, our hardware has changed out of all recognition. Whereas before our development and deployment systems ran on top spec Solaris servers with 64 CPUs, we can now get similar compute power on a $5k Xeon machine. However, we still potentially have differences between machines – with a production or user test machine being higher specification and larger scale than one used by developers. This makes recreating bugs more difficult, but on the flipside the increase in performance also makes using powerful debugging tools much more practical, as any overheads are dramatically reduced.

5 Regulatory change

The banking industry has seen significant regulatory change since 2007, so not only are we dealing with many more trading markets, but there is also a need to demonstrate compliance in new ways for each of them, leading to even more complexity. We therefore have a greater duty of care about how we carry out engineering and ensure releases are problem-free. This means we have to be able to trace all code changes as part of our compliance efforts.

Where reversible debugging helps

As I've said a significant part of any debugging exercise is recreating the problem in a repeatable fashion. Reversible debugging eliminates many of these needs, at least from the perspective of observing the bug, though the scenario still needs to be reproduced to adequately test the fix.

In the same way that software development has moved on in the last decade, so has reversible debugging, meaning there are many more products now in the space, each providing their own way of solving problems. These include solutions from Record and Replay (rr), Chronon and Time Machine for .NET, as well as Undo. There are also many improved ways of instrumenting live running code, from detailed tracing tools to products that can record live running processes without a debugger attached. In these cases, once the recording is saved, it can be shared with your development team for offline analysis. To illustrate this it would mean that if we found bugs when testing code in London we could simply send the recording back to the developers for them to run. No matter where they are, or what machines they have available, they can precisely reconstruct the program's original behaviour and step backwards as well as forwards in the code to find the root cause of the bug. This overcomes one of the main issues of first generation reversible debuggers – the difficulties of recreating bugs on the often different machines used in development, QA testing and production.

Using reversible debugging helps in four main ways:

- The ability to record and rewind gives much deeper insight into what caused a crash.
- The ability to collect evidence on one machine and replay an exact copy of it on another helps overcome the different machines/configurations used in development/testing/deployment. This allows faster tracking of small issues.
- It also works well in more structured testing, ensuring larger bugs are found.
- In User Acceptance Testing (UAT) systems. These are one step away from production, so are much more complex, with multiple connections to other systems. However, we're looking at how reversible debugging can help us in the future as the technology develops and performance further improves.

Looking to the future

While the potential of reversible debugging and the use of live recordings offers great potential productivity gains today I expect the technology to improve greatly in the coming years. Looking forward, I expect the technology to further improve in two key areas:

1. **Performance**

   Since first trying reversible debugging ten years ago, we've seen a better than tenfold reduction in the performance overhead. This is through a combination of development of the software and the greater processing power we now possess. If this trend continues to move significantly downwards, it opens up new possibilities in terms of the types of systems and software where we can use reversible debugging. The holy grail here would be to have recording always on.

2. **Multi-language**

   The days when organisations developed in a mandated, single language are increasingly in the past. This is particularly true as we move to a mobile-first world and developers adopt the best language for individual applications or needs. For example, currently Undo supports C++ on Linux and Android, and I expect this range to expand, driven by customer and market demand.

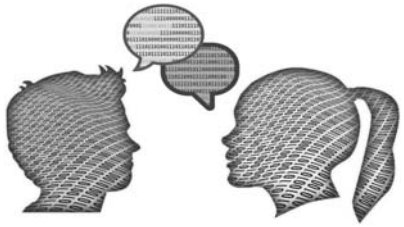
As I said, it is now ten years since I read that article in *CVu*, and came across reversible debugging and Undo. Since then I've watched the market change as reversible debugging has really gained momentum. There are now more entrants in the market, new ideas appearing and more importantly the innovation has been recognised by a number of other people.

For example my bank is a strong supporter of Accenture's FinTech Innovation Lab programme, a twelve week initiative that facilitates introductions for tech companies to stakeholders in leading financial institutions. Demonstrating the importance of software quality to the world's leading banks, Undo beat hundreds of other entrants to win the programme, giving it the chance to showcase its technology to more than 300 industry leaders. To me this shows the value of disruptive technologies such as reversible debugging in an era where code quality and security have never been more important. It endorses my original interest, and shows how banks such as mine can support and help innovation to grow and flourish.

Ten years is a long time in software development, so I won't risk predicting what things will look like in 2026. However, it is fair to say that we'll be developing and deploying in ever-more complex environments, with software central to the operations of all organisations, whatever sector they are in. Bugs will still be with us – and are likely to be more difficult to track down than ever. Expect more innovation from the likes of Undo to make debugging easier and to increase productivity – here's to the next 10 years. ■

Code Critique Competition 100

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



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.

Note: If you would rather not have your critique visible online, please inform me. (Email addresses are not publicly visible.)

The 100th critique

This issue contains the 100th in the code critique series which was started by Francis Glassborow (and a regular feature by the time I joined ACCU) and then continued under the oversight of David A. Caabeiro.

The code critique comes under the heading of 'Dialogue' and has attracted entries from a wide range of members over the years. It's a relatively easy place for people to contribute to the magazine, to learn something from the act of critiquing the code, and perhaps be inspired to try their hand at an article in the future.

I continue to enjoy setting them – but I'm still no nearer understanding why some issues attract many critiques and others attract only one! C/C++ example code generally is the most popular, but I do occasional produce problems written in other languages. I'm always keen to hear from people who have written (or come across) code that might form the basis of a critique – anonymity is provided if desired (!)

Last issue's code

I wanted to learn a bit about C++ threading so I tried writing a thread pool example. But it sometimes crashes – I've managed to get it down to a small example.

Sometimes I get what I expected as output, for example:

```
Worker done
Worker done
Ending thread #2
Ending thread #0
Worker done
Ending thread #1
Worker done
Ending thread #3
Worker done
All done
```

But other times I get a failure, for example:

```
Worker done
Ending thread #0
Worker done
Awaiting thread #1
Worker done
W
<crash>
```

I'm not sure what to do next – can you help?

The program is in Listing 1 (right and overleaf).

Listing 1

```
#include <algorithm>
using namespace std;
#include <array>
#include <chrono>
using namespace chrono;
#include <cstdlib>
#include <iostream>
#include <thread>

static const int POOL_SIZE = 4;

// Allow up to 4 active threads
array<thread, POOL_SIZE> pool;

// Example 'worker' -- would in practice
// perform some, potentially slow, calculation
void worker()
{
  this_thread::sleep_for(
    milliseconds(rand() % 1000));

  cout << "Worker done\n";
}

// Launch the thread functoid 't' in a new
// thread, if there's room for one
template <typename T>
bool launch(T t)
{
  auto it = find_if(pool.begin(), pool.end(),
    [](thread const &thr)
    { return thr.get_id() == thread::id(); }
  );
  if (it == pool.end())
  {
    // everyone is busy
    return false;
  }

  *it = thread([=]()
  {
    t();
    thread self;
    swap(*it, self);
    self.detach();
    cout << "Ending thread #"
      << (it - pool.begin()) << "\n";
  });
  return true;
}

int main()
{
  while (launch(worker))
  {}
  // And finally run one in this thread as an
  // example of what we do when the pool is full
  worker();
```

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

Listing 1 (cont'd)

```
  for (auto & it : pool)
  {
    thread thread;
    swap(thread, it);
    if (thread.joinable())
    {
      cout << "Awaiting thread #"
        << (&it - &*pool.begin()) << "\n";
      thread.join();
    }
  }
  cout << "All done\n";
}
```

Critiques

Felix Petriconi <felix@petriconi.net>

I personally would first include all necessary headers and then add a global using `namespace std`.

Since `chrono` is only used in the `worker()` function, I personally wouldn't use a namespace inclusion for a single occurrence. Alternatively one could simply add `using std::chrono::milliseconds` at the beginning of the `worker` function.

Instead of a fixed size of threads I would go for the native number of available cores of the used platform with:

```
std::thread::hardware_concurrency()
```

There is also no need to make `POOL_SIZE` a static variable. A simple constant would be fine for such a purpose.

Good that `std::array` is used here, but `hardware_concurrency()` is not a constant value during compile-time so one should probably use a vector.

For this example the usage of `rand()` is probably OK. But in general one should avoid using it, because the distribution is bad. See Stephan T. Lavavej's talk at the Going Native 2013 conference.

Using `std::cout` itself is thread safe, but the output is not synchronized. So I recommend creating a `log` function that synchronizes the output to `std::cout`.

Inside `launch()`, I would never pass an iterator into a thread. It is very dangerous in general to pass an iterator to something other than an algorithm-like function. I only use iterators locally within the scope of the current function or method. In this example the access to the thread objects happens without synchronization between the main thread and the spawned thread which results in undefined behaviour. So in the main thread a free slot is searched and in the spawn thread a) swapping the objects and b) detaching it happens. This must be done synchronized, e.g. with a mutex. As well there is again the problem with using `std::cout`. Please see above.

In the `main` function the loop variable is named `it`. This is misleading for a reader, because it is not an iterator, but a reference to the array objects. One sees the problem at the statement

```
&it -&*pool.begin()
```

From my point of view this looks somehow strange. Either I would encapsulate a `std::thread` object into some `struct`, that has an additional thread number attribute, or I would have let the pool be:

```
vector<std::pair<int, std::thread>>
pool(thread::hardware_concurrency());
```

so that the thread number is stored in the first element of the pair, or more simply I would use a classic `for` loop with an index like

```
for (size_t i = 0; i < pool.size(); ++i)
```

and then use the `operator[]` to access the individually thread elements. Again there is a race condition, because the thread function detaches the thread potentially at the same time as the main thread tries to do it.

In the code the array variable is named `pool`. But there is no pooling of threads. Here they are started with each new function and then it finishes. Normally one tries to reuse a thread for all the tasks, because spawning a new thread is very expensive.

I know that the code was written for getting to know threading. I experienced myself that threading is not easy and that are there are many traps that one can step into. So if I would need a thread pool for production code I would always try to look for a ready tested solution, before I try to build something by myself.

John Roden <john.roden@iol.ie>

Besides the `main` routine, the program starts four other threads whose administrative information is stored in an array of thread objects (the pool). This array is manipulated in both the main routine and each thread (by `swap()` functions). The result is that the main routine is not seeing the original thread objects when it awaits on threads to complete. The `swap()` functions aren't necessary in this case.

The program also detaches each thread with the result that the thread is no longer joinable. Tests reveal that the wait loop at the end sometimes sees a thread as joinable but by the time it tries to join it, the thread has detached and is no longer joinable – the program hangs in my test (it could legitimately crash!). There is no reason to detach the threads in this case.

My solution to the program's problems is to remove the `swap()` and `detach()` lines and access the original thread objects in every case.

These are the lines (commented-out) which need to be disabled in the code:

```
40        t();
41         //thread self;
42         //swap(*it, self);
43         //it->detach();
44         cout << "Ending thread #"
45           << (it - pool.begin()) << "\n";

60        //thread thread;
61        //swap(thread, it);
62        if (it.joinable())
```

In this example the threads do not need to change any shared information; if they did, a mutex of some sort would be needed but that complexity can be avoided here. The `cout` function is guaranteed to remain uncorrupted when used by multiple threads (C++11) but it is possible to get the output from different threads interleaved. When this is a problem, a mutex lock guard would be needed for each `cout`.

James Holland <James.Holland@babcockinternational.com>

Compiling and running the program revealed the crashing problems the student was complaining about. If the program contains threads and crashes intermittently, the problem stands a good chance of being caused by race conditions.

An inspection of the student's code reveals plenty of opportunities for race conditions. For example, the main thread calls `launch()`, that creates a thread object in one of the elements of the thread pool array, and starts to execute the worker function via a lambda. At more or less the same time, the lambda function calls the `detach()` function of the object just stored in the thread pool. I said more or less, because it is not possible to know which event will happen first; creating the thread object in the pool or calling the detach function. If the detach function is called first, the object will be the original default constructed thread. Calling `detach()` on a default constructed thread results in `std::system_error` being thrown. This is probably not what was intended. There is also a potential race condition with `swap()` within the lambda function.

Incidentally, I am not quite sure what the student had in mind when creating a default constructed thread and swapping it with the one in the pool. Perhaps it is something leftover from the student's main project. In any case it is a potential source of trouble. In fact any harmful race condition should be removed from the code.

It should be noted that the last line of the lambda uses the iterator (`it`) that refers to the thread object in the pool. This use of `it` does not cause a race condition because the code refers directly to the iterator and not what the iterator is pointing at (namely, the thread object). This just goes to show that making sure multi-threaded code is free of harmful race conditions is a tricky business. Because of this, it is best to keep the program design as simple as possible.

The code below runs the worker function in four threads (stored in the thread pool) and then waits for all the threads to finish executing before exiting. I believe it to be free of harmful race conditions but it still may not execute reliably. If `thread()` is unable to start a new thread, it will throw an exception of type `std::system_error`. To guard against this, no more threads should be created than given by `std::thread::hardware_concurrency()`. This function returns the number of hardware threads available. Unfortunately, the number returned is not guaranteed to be accurate, it is just an estimate. Because of this, the call of the `std::thread` constructor should be enclosed within a try-catch block.

Finally, I have made use of a `mutex` to ensure that a thread writes a complete line of text on the console without interruption from another thread. This simply improves the display formatting.

I have been able to describe only a few features of `std::thread`s. For a comprehensive treatment, I would recommend the book *C++ Concurrency in Action* by Anthony Williams, ISBN 9781933988771. Also recommended is Scott Meyers' book, *Effective Modern C++*, ISBN 9871491903995. Scott has a chapter devoted to concurrency and gives many useful hints and tips.

Finally, although not perfect, I provided an amended version of the student's code with race conditions removed.

```
#include <algorithm>
#include <array>
#include <chrono>
#include <cstdlib>
#include <iostream>

using namespace std;
static const int POOL_SIZE = 4;
array<std::thread, POOL_SIZE> pool;
std::mutex m;
void worker()
{
  std::this_thread::sleep_for(
    std::chrono::milliseconds(rand() % 1000));
  std::lock_guard<std::mutex> guard(m);
  cout << "Worker done\n";
}
template <typename T>
bool launch(T t)
{
  auto it = find_if(pool.begin(), pool.end(),
    [](std::thread const & thr)
    {
      return thr.get_id() ==
             std::thread::id();
    });
  if (it == pool.end())
  {
    return false;
  }
  *it = std::thread([=]()
    {
      t();
      std::lock_guard<std::mutex> guard(m);
      cout << "Ending thread #" <<
        (it - pool.begin()) << "\n";
    });
  return true;
}
int main()
{
  while (launch(worker))
  {}
  worker();
  for (auto & it : pool)
  {
    it.join();
  }
  std::lock_guard<std::mutex> guard(m);
  cout << "All done\n";
}
```

Commentary

There has been a fair bit of discussion over the years since C++11 was completed about the pros and cons of exposing direct access to threads in the C++ library. The original plan had been to get the basic mechanism into the standard and the to introduce higher level abstractions, implemented using the basic functionality, in subsequent versions of C++.

Unfortunately the higher level abstractions have been somewhat slower to appear than hoped, although there should be some parallel algorithms coming in C++17, with the result that much code is still being written using 'raw' `std::thread`. (While outside the scope of this critique, there are various libraries, not in the C++ standard, that do provide some higher level building blocks.)

One difficulty with writing threaded code is that it can be hard to prove the code is correct and any bugs often only appear intermittently. It is, as Felix and James variously pointed out, good to have a clear program design and to make use of existing, debugged, libraries or idioms for threading.

This example is purporting to demonstrate a thread pool but, as Felix said "there is no pooling of threads". The programmer might have done better to have taken some examples from, for instance, Anthony's book and to work through those rather than designing their own 'pool'-like system.

Multi-threaded code involves two levels of correctness (at least). The first level is that of *thread-safety*, i.e. that the various threads in the program manipulate data structures in a way that avoids undefined behaviour or crashes. The second level is ensuring that the behaviour of the program is meaningful and consistent, with whatever execution order the various threads involved end up using.

In this program, as each critique mentioned, the use of `cout` is thread-safe (because the standard guarantees there will be no data races when used like it is here) but the output can be interleaved and therefore inconsistent. However, the use of `pool` is not even thread-safe as multiple threads modify the array using `swap()` and iteration without any mechanism to make the access correct.

There are two main directions for making the use of `pool` correct. One way is to use explicit synchronisation (probably using a mutex) and the other way is to ensure clear ownership of the object so that only **one** thread accesses the object (John's solution takes this route).

The Winner of CC 99

The three critiques seemed to cover the main problems with the code between them. I'm not sure I'd want to go quite as far as Felix and *never* allow an iterator to be passed to a thread, but it is definitely a bit of a 'code smell' and would need to be designed carefully to ensure the access remains valid whatever the subsequent execution pattern.

There was some discussion in the critiques about the best way to size the number of threads to use for the machine executing the program rather than using a fixed value (`POOL_SIZE`) – possibly involving `std::thread::hardware_concurrency()`. This function returns the 'number of hardware thread contexts' (but this should only be considered as a hint.) You can normally *create* many more threads than this, but only this number of threads can actually execute simultaneously. (This is additionally subject to other platform-specific constraints, such as thread affinity masks.)

Overall I think Felix provided the fullest critique so I have awarded him this issue's prize.

Code critique 100

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

I wanted to do something slightly different for this, the 100th code critique column, so I have based this issue's critique on the 'left-pad' function that was part of npm but was withdrawn by the author, breaking a large number of components on the Internet. See

http://blog.npmjs.org/post/141577284765/kik-left-pad-and-npm

for some more official information about the issue and this blog:

http://www.haneycodes.net/npm-left-pad-have-we-forgotten-how-to-program/

for some discussion about some of the issues it raises.

(The left-pad story was briefly discussed on accu-general during the 'NIH syndrome' thread.)

Please feel free to comment on the Javascript code itself, or on the wider issues raised by the story.

Listing 2 contains `leftpad.js`, and a trivial test page is provided in Listing 3 if you want to play with the function in a browser.

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://accu.org/index.php/journal). This particularly helps overseas members who typically get the magazine much later than members in the UK and Europe.

Listing 2

```
function leftpad (str, len, ch) {
  str = String(str);

  var i = -1;

  if (!ch && ch !== 0) ch = ' ';

  len = len - str.length;

  while (++i < len) {
    str = ch + str;
  }

  return str;
}
```

Listing 3

```
<!DOCTYPE html>
<html>
<head><title>CC100</title></head>
<body>

<h1>Code Critique 100</h1>

<script src="leftpad.js"></script>

<script>
function testLeftpad() {
  result.innerHTML = '"' +
  leftpad(text.value, len.value, pad.value) +
  '"';
}
</script>

<table>
<tr><td>Text to pad</td><td>
  <input type="text" id="text" value="1234">
</td></tr>
<tr><td>Length</td><td>
  <input type="text" id="len" value="10">
</td></tr>
<tr><td>Pad char</td><td>
  <input type="text" id="pad" value="0">
</td></tr>
</table>

<br/><button onclick="testLeftpad()">
Try out leftpad
</button>

<pre id="result"></pre>

</body>
</html>
```

Write for us!

C Vu and Overload rely on article contributions from members. That's you! Without articles there are no magazines. We need articles at all levels of software development experience; you don't have to write about rocket science or brain surgery.

What do you have to contribute?

- What are you doing right now?
- What technology are you using?
- What did you just explain to someone?
- What techniques and idioms are you using?

For further information, contact the editors: cvu@accu.org or overload@accu.org

View from the (Acting) Chair

Bob Schmidt
chair@accu.org

Please allow me to introduce myself. I am Bob Schmidt, and I have volunteered to be the ACCU Chair for the 2016–2017 term. I am based in Albuquerque, New Mexico, USA, making me the first non-UK based Chair of the organization (but please don't hold that against me). I am president of Sandia Control Systems, Inc. and have been a consultant and contractor since 1994. I am a licensed Professional Engineer in the U.S. state of Texas. I've been a member of ACCU since 2011.

Currently I'm acting chair – I decided to volunteer after the AGM in April, when the position remained vacant. I wanted to ask a past holder of the position about the duties and time commitment before volunteering, and I wasn't able to do that until after the meeting. I'd like to express my personal thanks to past Chairs Ewan Milne and Alan Griffiths for answering my questions about the role. They were instrumental in helping me decide to volunteer.

Anna-Jayne Metcalfe presented a thought-provoking keynote on Comfort Zones, shortly after I volunteered. As she was talking about stepping outside of one's comfort zone, I thought to myself that I hadn't so much stepped out of mine, but rather had jumped from a fourth floor without benefit of a net. I spent a considerable amount of time that evening trying to get my logical brain to convince my lizard brain I had done the right thing. (My wife just laughed when I told her.)

My main goal for ACCU in the coming year is to keep the organization ticking along. Based on what I heard during the AGM and the following ACCU conference session, we have just one pressing issue – our web site and web hosting. Tim Pushman, who has been hosting the web site for a number of years, had informed the committee he was no longer able to continue in the role. Our immediate need is to find a new hosting platform, probably one that will allow us to run the existing platform. Our longer term need is to replace that older platform with a current one. That platform will need to support the existing site functionality.

By the time this gets published we should have a new host platform, and hopefully our existing software will be transferred to it. We are still looking for one or more volunteers who would like to help upgrade our platforms and improve the site. This will not be a long-term commitment. If you are interested please let me know.

ACCU will need an additional auditor for 2017, as I will be unable to continue in that role. (Auditors are independent of the committee.) The role of auditor is not an onerous one, requiring only a few hours of effort in the month prior to the AGM. Qualifications include being able to balance a cheque-book and a minimum familiarity with Microsoft Excel. Rob Pauer, our treasurer, was very helpful answering the questions I had this year. Guy Davidson is starting his two year term as auditor. If you are willing to join Guy and accept the role for the one year remaining on my term, please send me an email.

The 2016 ACCU Conference was a great success, and a tribute to the hard work of our Conference Chair, Russell Winder. It was a shame that Russell was unable to attend in person to see the fruits of his labour, but I'm told he was following some of the sessions as they were posted on YouTube. Russell is continuing in the role as Conference Chair for 2017. Please join me in thanking Russell for a job well done, and wishing him a continued and speedy recovery. Thanks also to Jon Jagger for stepping in as master of ceremonies.

During the ACCU conference session we heard reports from Fran Buontempo and Steve Love on the challenges they face as editors of *Overload* and *C Vu* (respectively). Their main concern is a lack of a backlog of articles waiting to be published. I'm sure you have seen their requests for material on the accu-general mailing list. Currently they are operating in 'just-in-time' mode, and are considering reprinting articles to help with content.

I had not worked with Steve prior to this report, but I have written several articles for Fran. *Overload* is a peer-reviewed journal; I can tell you from experience that Fran and her fine group of reviewers are excellent at bringing out the best in you and your material. It may not be true that everyone has a book in them (and if so it should probably stay there), but we all have something to share. Please consider working with Steve or Fran and sharing your experiences and expertise with all of us.

One of the action items I inherited with the position is to develop a Diversity Statement for the organization. I have a draft that I have shared with the committee, whose consensus was that we need more input from a more diverse group of members. If you have strong feelings about what a diversity statement should say (or not say), please contact me and I will add you to our (hopefully) growing list of participants in the process.

As I stated in the opening paragraph, currently I'm just the acting chair. In addition, Malcolm Noyes also volunteered to stay in the role of Secretary after the AGM. We need to hold a special general meeting in order to vote on Malcolm's and my formal nominations for these roles. Due to the steps and timeline required by the constitution the vote probably will be held sometime in September. Watch this space and accu-general for more information as it becomes available.

JOIN THE ACCU!

You've read the magazine, now join the association dedicated to improving your coding skills.

The ACCU is a worldwide non-profit organisation run by programmers for programmers.

With full ACCU membership you get:

- 6 copies of *C Vu* a year
- 6 copies of *Overload* a year
- The ACCU handbook
- Reduced rates at our acclaimed annual developers' conference
- Access to back issues of ACCU periodicals via our web site
- Access to the *mentored developers projects*: a chance for developers at all levels to improve their skills
- Mailing lists ranging from general developer discussion, through programming language use, to job posting information
- The chance to participate: write articles, comment on what you read, ask questions, and learn from your peers.

Basic membership entitles you to the above benefits, but without Overload.

Corporate members receive five copies of each journal, and reduced conference rates for all employees.

How to join

You can join the ACCU using our online registration form. Go to **www.accu.org** and follow the instructions there.

Also available

You can now also purchase exclusive ACCU T-shirts and polo shirts. See the web site for details.

Design: Pete Goodliffe

PERSONAL MEMBERSHIP
CORPORATE MEMBERSHIP
STUDENT MEMBERSHIP

PROFESSIONALISM IN PROGRAMMING
WWW.ACCU.ORG