the magazine of the accu

www.accu.org

{cvu}

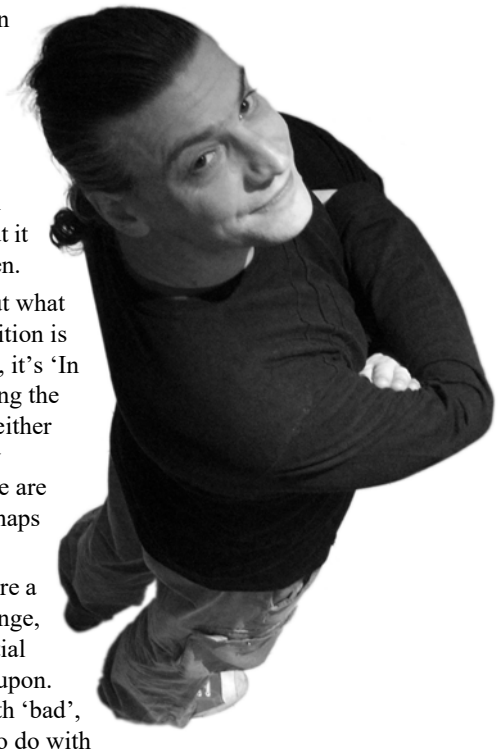
Volume 29 • Issue 2 • May 2017 • £3

Features

Regulars

{cvu}
Volume 29 Issue 2
May 2017
ISSN 1354-3164
www.accu.org

Production legacy



I expect we've all had our moments of wishing for the demise – or perhaps better yet, obliteration – of some legacy system, usually where we ourselves hadn't had a hand in its implementation, and we have our own opinions of what could have been done to make it better. Or at least, bearable. Sometimes the prospect of tearing an existing system down and rebuilding from scratch is incredibly alluring, but it is a luxury that we don't get to indulge very often.

There is much debate in technology circles about what the term 'legacy' means, but one common definition is any system that's being used for real work – i.e., it's 'In Production'. Some prefer a looser definition along the lines of having survived a version upgrade, but either way, projects either survive and become 'legacy systems', or they fail. Whether those that survive are successes, or merely not failures, is a debate perhaps for another time.

Even in a continuous delivery environment, where a deployed system undergoes almost constant change, it is still in some sense a legacy system after initial release, although the term seems to be frowned upon. This seems to be because 'legacy' is equated with 'bad', although its more general English definition is to do with handing something down as a gift. It's also associated with 'old', which carries its own disparaging connotations in technology.

Still the fact remains that most of us spend most of our time working on – fixing, upgrading, maintaining, maybe even improving – legacy systems, rather than producing new systems from scratch. After all, if the majority of work was on new endeavours, that would mean that the majority of past projects had failed.

Maybe we should start seeing the positive aspects of our 'Legacy' and strive to make our systems as welcome a gift to future generations of programmers as we are able.

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 29.3: 1st June 2017
C Vu 29.4: 1st August 2017

Overload 140: 1st July 2017
Overload 141: 1st September 2017

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

An Ode to Code

Pete Goodliffe shares his annual programming practice poem.

For the last few years at the excellent ACCU Conference, I've shared a short coding poem as one of the lightening talks. (I have to admit that the lightening talks have always been somewhat of a highlight of the conference for me not *only* because of my contributions. You never know what short nugget of wisdom or entertainment you'll hear next.)

This year I felt I couldn't possibly break from this tradition, so I present for your delight this year's entry. It's worth reproducing in *CVu* because: many of the membership are not able to attend the conference, and every time I 'perform' a poem, people come up afterwards and ask for a copy. Well, here it is…

Every year a coder here[1] *will bring a coding rhyme.*
Some might be bad, but five minutes is not too long a time.
But often these performances, whilst they might entertain
Don't do much to help us learn, to teach us, or to train.

So this year, here, I'd like to bring a far more useful thing:
Coder wisdom, earned with scars, that's set for you to sing.
Hopefully among the other conference presentations
Its memorable, through meter, humour and practical demonstration.

ACT 1

'Lay code out well' is our first theme, it's what we're often taught.
But should we care? Yes! Let me share one way that I was caught...
One time I worked upon some code of Japanese descent;
The experience was eye opening and led to this lament.

When you can't tell a single thing about the code you're reading,
The lack of comprehension makes your working quite misleading.
Without clear names and clean code shape, subtle bugs can hide.
So, syntax faults and code horrors can lurk unseen inside.

It bit us once: to ship this beast with behaviour that was dumb
caused by a missing equals sign (that should stand out like a sore thumb)
We had to spend more than four days on extended bug foray.
If the sign aligned, we'd save that time, and reduce shipping delay.

Whilst you yourself might not have code from more than one location,
The lesson from this story serves as adequate indication.
Care how code looks; the ease of reading every coding line
Impacts the speed of your code feed: you shrink your debug time.

ACT 2

'Code reviews' are our theme two, a practice often skipped,
Although without it quality is often seen to slip.
We want the highest standard code, we aim for code hygiene;
Review accountability helps to achieve this dream.

It shouldn't take much extra time if it is done correctly,
The benefits outweigh the costs; you'll reap rewards directly.
Is interesting that modern workflows agree that this is best;
It's seen most clearly used today in Github's pull request.

You don't want such an onerous scheme that it is hard to do.
You need to find the review balance that works the best for you.
And, if you don't – you let your team commit what code they wanta --
Don't be surprised if over time the code becomes a monster.

Story two that I'll tell you corroborates this claim,
And shows why code reviewing should be high among your aims.
We'll see a way a team once failed through poorly managed process,
And hopefully improve when learning from the diagnosis.

To get some software fully built the team, but one, ploughed on
To fix the final showstoppers and get a release done.
A solitary coder split to start on the next thing.
"We'll join you shortly" said the rest, believing their planning.

But problem upon problem beset the release crew;
And as quickly as they worked they found more work they had to do.
So, rather than a week or two, for months they parted ways,
And the solitary coder worked with no-one to appraise.

He built most of the next release, and built it by himself.
One man writing all code alone may not lead to code health.
His labours worked, the code it ran, so all seemed pretty cool.
But some time later they found out he was a coding fool.

After months and months of coding, the next release was nigh.
There was no time to re-write, so they used his code and sighed.
It was painful but they did it, the next release date met.
But their precious codebase was now stuck with much more technical debt.

The moral of the story told is clear for you to see:
Coding and design reviews help you work efficiently.
Even the most accomplished dev, when given things to do
May get it wrong, and code review helps stop you go askew.

EPILOGUE

So that's the end of this lecture; my five minutes are done.
I trust the lessons shared with you were useful and were fun.
Now, keep your coding clean and your process goodness showing,
Or else you'll end up starring in a conference coding poem. ■

1. Me, usually

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

Myths about 'Big Data'

Reginald Garnepudi dispels some of the hype around one popular technology.

If I could borrow the neuralyzer for just one day, I would erase the phrase "Big Data" from the memories of all people. (By the way, a neuralizer is a device about the size of a large cigar, seen in the *Men in Black* movie series that gives a bright flash and erases the memories of people.)

Since you are reading this, I assume that you've definitely heard of what Big Data is and if you are like most, you have more questions than answers. People selling Big Data solutions make a lot of (false) promises and build a lot of hype around it, so this is an attempt to shed some light on the reality.

Myth #1

Big Data is an actual thing, a tool, a piece of software that you can install or do something with it.

Myth buster #1: It is not a tool, or a platform or an API or a database. Big Data is actually a kind of a problem that specifically affects people in the data world (database admins, data warehouse admins, data governance people, IT infrastructure folks). When they have a situation where the data that their organisation is generating (or consuming) is bigger (or coming in faster) than they can efficiently and cost effectively handle, then they have a "Big Data" problem on their hands. That's all there is to it. Nothing more.

Myth #2

If you setup a Hadoop (Big Data) platform in your organisation, it will automatically spit out insights that will transform your business.

Myth buster #2: Hadoop or any other so-called Big Data platforms are software tools that solve a specific class of problems. i.e., they make it easy for you to store and manipulate data on multiple smaller servers instead of just one massive server (like traditional database servers). What this gives you (as a person in the data world) is that you can scale horizontally across multiple servers as your organisation grows. They don't have any inbuilt capabilities to spit out business insights if you throw more data at them. That's all there is to it. Nothing more.

Myth #3

Everyone has a Big Data problem and everyone at the CXO level should be worried about it.

Myth buster #3: As I said in myth buster #1, the Big Data situation is a problem in the data folks' realm. It should never be a concern at the CXO level. Unfortunately, the hype starts at the top of the organisation and is forced down the hierarchy.

Myth #4

A Big Data platform is a cheap alternative to traditional data warehouse solutions.

Myth buster #4: In the slide decks of people who sell these platforms, you will most definitely see a slide that shows that Hadoop (being open source) is so much cheaper compared to proprietary platforms. But if you add up all the infrastructure costs, cloud hosting or on-premise costs, support license costs, hiring external consultants, up-skilling your existing team, development costs, etc., the difference could be insignificant.

Myth #5

A Big Data platform (aka, Data Lake) is where you dump all your data and worry about it later.

Myth buster #5: Only insane people will even think of doing this. Why would anyone spend thousands of dollars creating a brand new data repository, only to aimlessly dump today's data so that it might be useful tomorrow? Data is the most precious commodity that an organisation has. And it has to be treated with utmost care and upfront planning. It is very common to hear statements like "just dump everything you have in a data lake, because its cheap, and you can worry about how to use the data later". I implore you to please don't fall for this theory. Always, always start with a plan about what you want to do with your data upfront.

Myth #6

Big Data is a new problem

Myth buster #6: The problem of insufficient storage and processing capacity for the amount of data you have (or want to have in the future) is not a new problem. Organisations' data needs have always been playing catch-up with the available capacity ever since computers were invented. Technologies like Hadoop make it possible to scale horizontally, assuming you are willing to spend time, effort and sweat.

Myth #7

Big Data platform will make the Data Analysts' life easy

Myth buster #7: This is probably the worst myth I always hear. There are two distinct kinds of people in an organisation…

1. Folks who are responsible for collecting, storing and managing data. (These are the people I talk about in Myth #1 above). And
2. The ones that use the data and do something with it. I'll put these people in the broad bucket called Data Analysts (aka, Data Scientists). These are the ones that take data from one or more sources, do some data wrangling, munging and massaging in order to dig out answers to business questions about "how to save money" or "how to make more money".

The number of problems that the Data Analysts solve that need huge amounts of data is so minuscule, that they hardly ever think in terms of big data. The (statistical and machine learning) algorithms that they use often don't need huge data.

If you understand these myths, you might then ask, "There seems to be so much buzz around this, so there has to be something here that we can use. What is it?"

Yes, there is something!! For an organisation, data is often the most under utilised asset. If you really want to see how data can be useful in your organisation, you have to embrace a new/different paradigm. It is the paradigm of becoming a 'Data Driven Organisation'. Big data is just a small challenge, once you start thinking about how data can be the new oil that runs your organisation. ■

REGINALD GARNEPUDI

Reginald is a passionate freelance software architect/analyst/leader/developer/geek. He enjoys solving complex data science challenges for clients and teaching programming to kids. He can be reached at reg@constantgeeks.com.

On Turnabout is Fair Play

A student responds to the Baron's latest challenge.

Last time they met, the Baron challenged Sir R----- to turn a square of twenty five coins, all but one of which the Baron had placed heads up, to tails by flipping vertically or horizontally adjacent pairs of heads.

As I explained to the Baron, although I'm not at all sure that he was following me, this is essentially the mutilated chess board puzzle and can be solved by exactly the same argument. Specifically, we need simply imagine that the game were played upon a five by five checker board

Figure 1

which we mutilate by removing the square upon which the Baron had placed the tail

Figure 2

Now, at each turn of a pair of coins Sir R----- must place tails upon both one white and one black square, and so if there are more of the one than there are of the other then he cannot possibly succeed in turning them all. The Baron could have ensured that this was the case by placing his tail upon a black square, leaving eleven heads upon the black squares and thirteen of them upon the white squares, and I should therefore have advised Sir R----- to most emphatically decline the wager! □

Acknowledgement

Courtesy of www.thusspakeak.com

A Hollywood Take on Dangling Pointers?

Silas S. Brown tells a fable about read-only variables.

A friend wrote `someString.substr(3).c_str()` and her code seemed to work but her colleagues were saying something about "going out of scope" that she didn't understand. So I sent her a film script.

The scene opens in some forgotten alleyway. Cue scary music. In walk the Silly Evil Overlord (SEO) and his Minions, all dressed in black. (Yes, I'm afraid it's only a B movie. We're not super-rich you know.)

The SEO speaks. "And now", he says, "my secret plans for dominating the world are well in progress. I tricked the book-shop chain into opening up a new book shop that contained just the books we wanted. And then", (turns to shout at his Minions) "Minion Number One! Did you accomplish your Mission for me?"

"Yes my lord. I stole a special piece of paper from the new book shop, my lord. This should tell us everything we need to know."

"Good", said the SEO. "And now, because I am Evil, and the audience need to see my Evilness, it's time for my first Evil Act of the Film. Mwa-hahaha!" (stereotypical evil laughter) "Seeing as we have now got what we want from the new book shop, we have no use for that book shop anymore. Mwahaha!" (Pulls out a small cylindrical gadget marked 'Scope of Destruction' and presses a button on it. Explosion sounds. Stock footage of pyrotechnics. Hole in the ground.)

SEO still chuckling. "And now" (pause) "for the second part of my Evil Secret Plan. The piece of paper you took from the book shop contains all the details we need to break into the bank and steal a gazillion dollars." (We're in America of course. Where else would make movies this bad? Oh, I'll probably get some flak for that.)

Scene changes to inside a tunnel. A sign says 'Bank Vault' above a myriad flashing lights and buttons. SEO and Minions arrive.

"And now" says the SEO, "Minion Number Five! Read the special piece of paper that tells us the Secret Security Key!"

Minion Number Five starts to read. "*Breaking Into Movie Banks for Dummies*. The book you require is located on shelf 451 of the Book Shop."

Minion Number Four steps forward. "My lord, I will go to the book shop and get it at once." (runs off)

Minion Number Three: "ummmm... my lord? He won't get very far my lord."

SEO thunders "WHY not?"

"Because, um, er, my lord, pardon me please, your evilness, I mean, um,"

"You blew up the book shop, my lord" says Minion Two.

SEO is indignant. "We didn't have to keep the book shop around, wasting all that space! We had no use for it anymore after we took out the one sheet of paper we really needed. Besides, imagine the inconvenience of having to keep a whole book shop on my List of Things to Blow Up Before I Die. Besides, didn't Mr Black assure us that one piece of paper from the book shop was all that would be needed?"

"With all due respect my lord" said Minion One, "perhaps what Mr Black really meant to say is, he can manage on one piece of paper as long as the book shop is still around for him to..."

SILAS S. BROWN

Silas 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

"WHY DIDN'T Mr COMPILER pick me up on this??" thundered the SEO.

Enter Mr Complier. "My lord, technically you didn't break any of my Compiler Rules. You said you needed to keep the piece of paper, and you kept it. You never said the book shop it points to has to still be around for you to check what it refers to. That's not in my department to spot. Maybe Mr Lint might have seen it, if you'd happen to catch him on a good day, but fundamentally you must understand that, although we do try to point out your mistakes as best we can, there's a certain class of error that always sneaks past us and -"

"Enough!" thunders the SEO. "I'll blow YOU up next. Suggestions! There must be Something we can do."

"My lord", chimes in another Minion, "we put a special lock on it called `const`, so it can't change. Doesn't that mean the book will have been protected from change even though you blew up the book shop?"

Laughter all round. "Wait" said the SEO, "I want to look into this. Minion Number Five! You didn't read the whole piece of paper, I know it. Read it again."

Minion Number Five was nervous. "OK", he managed, "here we go my lord: *Breaking Into Movie Banks for Dummies*. The book you require is located on shelf 451 of the Book Shop. The bearer of this Ticket is entitled to read the Book but not to write in it."

Silence. Camera pans around everyone's faces.

Minion Three pipes up. "So in other words, `const` doesn't mean it can't change, it just means they won't let US change it with that ticket. It might still change behind our backs if someone else has a better ticket. Or a…" (looks nervously at the spent detonator and breaks off)

Mr Compiler comes back in, "It means I'll moan at you if you try to write in the book" he said. "Unless of course you tell me to stop moaning with a `Cast`. In which case I'll let you write in it. And yes, of course it's no guarantee against somebody else writing in it, or moving it, or blowing it up. That's the trouble with those tickets. If you've got the ticket, it doesn't mean you've got the book, so you'd better know where the book is and who's looking after it."

"Oh" spat the SEO. "Useless Tickets. Next time I have an Evil Plan, I'll have my minions copy the whole Book, not just the Ticket!"

Gasps around all the Minions. "My lord! Tickets can be very useful for speed, much faster than copying a whole book! You just have to make sure you know what you're doing with…"

"Silence!" thundered the SEO. "Until I know what I'm doing, I'm having you all copy out whole books next time before I blow up the book shop, even if that does slow me down a bit! In fact, I'm going to find some more intelligent Minions that know when to steal a book instead of copy it. But that comes later."

A Minion dares to speak up: "Maybe we should try to get that book anyway, my lord. I mean, it might have survived the explosion. Sometimes they do, especially if nobody's got around to re-building something else on the same spot before we get there. Like when we vandalised the market stall in the Prequel Film That Shouldn't Be Made, remember? So it does sometimes work anyway. And if it doesn't, we'll just get nothing or gibberish, and we'll know. It can't be worse than that, can it?"

Minion Number Four comes back, panting and clutching a book. "My lord, guess what? Your Arch-Rival, sir, the Intelligent Evil Overlord. He persuaded the Town Planners to allocate the freed-up spot to him, and he

I Can't Think Fast Enough in a Coding Interview

Sean Corfield shares his thoughts on the technical interview process.

'Practice' is not the answer. 'Cracking the Coding Interview' is not the answer. A few people have said it – and more of us should be saying it: these 'coding monkey interviews' are stupid. They do not determine how good a programmer someone will be once they're actually hired and working. The big companies that use them have finally started to admit this. The hiring process is broken and we need to stop participating in this silly game so many companies have adopted because they're too lazy to figure out how to interview people effectively. I've refused to interview with companies that do this and I've walked out of interviews that have turned out that way. If a company really believes a quick fire 'coding monkey interview' will find them the 'best programmers' then that is a company you want to avoid – they don't understand their engineers, they don't know what makes a good team.

Remember that the interview is a two-way street. You are interviewing the company as much as they are interviewing you. An interview should be a conversation about what you enjoy in a software project, in a team, in a manager – as well as what you find problematic. It should be a chance for you to talk about how you approach problems – both technical problems and people problems – how you balance trade offs, and how you deal with things that don't go your way. You should be able to defend your choice of technology but you should also know it well enough to be able to talk about its flaws or the situations where it isn't such a good fit. You should be able to ask about the company's processes, how they manage teams, how they resolve conflict, how they reward success.

An interview should reflect the sort of collaborative process you can expect once you are an employee at that company. If an interview seems to be a confrontational process, assume that's how the company will treat you once you're hired.

I've been a hiring manager for about twenty years now. I do not do 'coding monkey interviews' because I know they do not work. And I have never hired anyone that can't do the job I've hired them for. ■

Reference

https://www.quora.com/Im-a-software-engineer-with-20-years-experience-but-I-cant-think-fast-enough-in-coding-interview-What-should-I-do/answer/Sean-Corfield?share=1587c4af#

SEAN CORFIELD

Sean Corfield used to build compilers, virtual machines, databases and telecom systems but he finally found his first love again – functional programming – and now he writes Clojure almost every day, and blogs about it at https://seancorfield.github.io/

A Hollywood Take on Dangling Pointers? (continued)

made sure we still had the access rights. And he built an exact replica of the book shop you blew up, in the exact same place, and sure enough it had the right-looking book on shelf 451. Hot off the press too! And it's a signed copy, look at this." (opens at the first page, SEO reads) "Dear SEO, I know you'll enjoy using this special replacement edition I made specially for you. I wouldn't normally say anything, but I just had to sign it this time for the audience's benefit. Mwa-hahaha. IEO." (Let's not call him Cunning Evil Overlord because that's how you get fired.)

Minion grabs book. "Isn't that kind of him? I'll start using it at once, my lord. What could possibly go wrong?" (Jarring orchestral chord. Fade to black.)

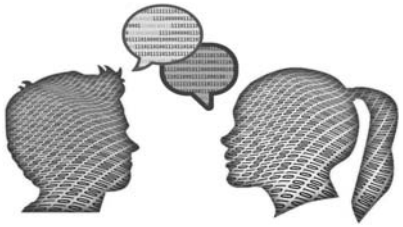
Two months later my friend was unfortunately fired from her post, saying they'd criticised her for failing to work independently, but also for failure to sufficiently communicate with colleagues.

She found this contradictory, and I must confess I would have expected a little more clarity from an 84,000-employee German enterprise.

I recommended she make a point of asking future potential employers if they practise pair programming – it sounds like that's what she needs. ■

Code Critique Competition 105

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.

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

Last issue's code

I was trying to write a simple template that gets the unique values from a range of values (rather like the Unix program uniq) but my simple test program throws up a problem.

```
test with strings
a a b b c c => a b c
test with ints
1 1 2 2 3 3 => 1 1 2 3
```

Why is the duplicate 1 not being removed?

The code is in Listing 1 (unique.h) and Listing 2 (unique.cpp).

Listing 1

```
#include <iterator>
#include <vector>
// get unique values in the range [one, two)
template <typename iterator>
std::vector<typename iterator::value_type>
unique(iterator one, iterator two)
{
  if (distance(one, two) < 2)
  {
    // no duplicates
    return {one, two};
  }
  // first one can't be a duplicate
  std::vector<typename iterator::value_type>
  result{1, *one};
  while (++one != two)
  {
    auto next = *one;
    bool is_unique =
     (*result.rbegin() != next);
    if (is_unique)
    {
       result.push_back(next);
    }
  }
  return result;
}
```

Listing 2

```
#include <iostream>
#include <string>
#include <vector>
#include "unique.h"
template <typename T>
void test(std::ostream &os,
  std::vector<T> const &vector)
{
   auto result =
     unique(vector.begin(), vector.end());
   auto out =
     std::ostream_iterator<T>(os, " ");
   copy(vector.begin(), vector.end(), out);
   os << "=> ";
   copy(result.begin(), result.end(), out);
   os << "\n";
}
int main()
{
   std::cout << "test with strings\n";
   std::vector<std::string> ptrs;
   ptrs.push_back("a");
   ptrs.push_back("a");
   ptrs.push_back("b");
   ptrs.push_back("b");
   ptrs.push_back("c");
   ptrs.push_back("c");
   test(std::cout, ptrs);

   std::cout << "test with ints\n";
   std::vector<int> ints;
   ints.push_back(1);
   ints.push_back(1);
   ints.push_back(2);
   ints.push_back(2);
   ints.push_back(3);
   ints.push_back(3);
   test(std::cout, ints);
}
```

Critique

Jon Summers <la_solutions@btconnect.com>

Header file `unique.h` initialises the result vector in this statement:

```
std::vector<typename iterator::value_type>
result{ 1, *one };
```

The initial values are passed in an initialiser list. Unfortunately, at least for this question, the initialiser list passed to a vector can have two meanings. The data can either be a list of values, each having the same type; or an integer that specifies the initial size, followed by data either of the same or a different type.

The data type of `*one` is templatised. When instantiated with a `std::string`, the initialiser list is

```
{ 1, std::string ("some string") }
```

The compiler understands that to mean: "Create a vector having one element, whose value is 'string'".

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

When instantiated with an `int`, the initialiser list is

```
{ 1, 1 }
```

The compiler understands that to mean: "Create a vector having two elements, whose values are 1 and 1."

The following loop that examines successive values for uniqueness doesn't see the first element in the `vector<int>`, because its logic assumes an initial length of 1.

Paul Floyd <paulf@free.fr>

My first reactions on reading the code were

- why is this reinventing the wheel when `std::unique` exists? (http://en.cppreference.com/w/cpp/algorithm/unique)
- which constructor is being called for '`result`' in '`unique`'?

The compiler [clang++ on Mac OS] also isn't too happy with this re-inventing of the wheel:

```
cc104.cpp:10:6: error: call to 'unique' is
ambiguous
  unique(vector.begin(), vector.end());
```

In order for it to compile, I had to rename '`unique`'.

The behaviour of this unique is different to that of `std::unique`. The version presented here copies unique elements to a new vector (without any call to reserve). It then returns this vector, which may require another copy operation – '`unique`' has more than one return path which may inhibit NRVO. `std::unique` modifies the container, moving unique elements towards the head. This minimizes the amount of allocation and copying, and still allows the user to make a copy if she doesn't want the original vector to be modified. Personally I prefer `std::unique` as it is more in the C++ philosophy that it doesn't make you pay for something that you don't necessarily want.

As to which constructor is being called for `result` in `unique`, let's look at the code:

```
result{1, *one};
```

Unfortunately, this could have two meanings. If the type of `*one` is the same as `1`, i.e., an `int`, then this can be a `std::initializer_list`, corresponding to the following vector constructor:

```
vector(std::initializer_list<T> init,
  const Allocator& alloc = Allocator());
```

If the type of `*one` isn't `int`, then the constructor is

```
vector(size_type count,
  const T& value,
  const Allocator& alloc = Allocator());
```

The second case occurs when the vector is instantiated with `std::string`, consequently `result` contains a single value, the first value of referenced by the input iterator `one`. This is probably the intent of the code.

The first case occurs when the vector is instantiated with `int`. In this case, the first `1` isn't considered to be the count, it is treated as a value in the init list. By coincidence it happens to be the same as the input vector values. So it isn't the case that the first `1` isn't being removed, rather an extra `1` is always being inserted at the head position.

To wrap up, I have a few nits. I don't like the arguments called `vector` and `iterator`. In fact I don't like any language keywords or standard library names to be used as variable names. This just makes the code harder to read and more difficult to talk about.

Lastly, the `std::vector`s in `main` could be more concisely initialized with init lists.

Raimondo Sarich <rai@sarich.co.uk>

Firstly, I could not compile the code with "`error: call to 'unique' is ambiguous`". I renamed the function to `myunique`, I assume the developer is using a more permissive compiler.

The bug is pretty easy to spot, this:

```
std::vector<typename iterator::value_type>
result{1, *one};
```

has to change to this:

```
std::vector<typename iterator::value_type>
result(1, *one);
```

This problem is covered Scott Meyers' *Effective Modern C++*, Item 7:

> If, however, one or more constructors declare a parameter of type `std::initializer_list`, calls using the braced initialization syntax strongly prefer the overloads taking `std:initalizer_lists`. Strongly. If there's any way for compilers to construe a call using a braced initializer to be a constructor taking a `std::initalizer_list`, compilers will employ that interpretation.

He goes on to explain exactly this difference between constructing a `std::vector` of numeric types with curly braces versus parentheses. Beware `std::vector`!

There seems to be little else to remark on. The code could use braced initialization of the test vectors, a scattering of `const` (the vectors, `two`, `next`, and `is_unique`), and `cbegin`/`cend`.

`std::vector::back()` would be better than `*std::vector::rbegin()`. And of course, `std::unique` should be preferred outside academic purposes.

James Holland < james.holland@babcockinternational.com>

At first glance, the result of the student's program seems baffling. How can the software produce two different results just because on one occasion the vector contains elements of type `int` and on another occasion elements of type `std::string`? The reason for this behaviour hinges on the set of constructors `std::vector` possesses and the way a particular constructor is selected.

Consider what happens when the compiler comes across the statement `std::vector<std::string> result{1, "a"}`, for example. The compiler tries to match the parameter types to one of `std::vector`'s constructors. The compiler's first choice is to select the constructor with a parameter type of `std::initializer_list<T>`. This is feasible as the vector's definition used braces in the initialisation. However, the compiler cannot use this constructor as the first value in the initialisation list is not convertible to a string. The compiler has to discard the constructor and attempt to select another. Eventually the compiler comes across a constructor that takes a numerical value and an `std::string`. The compiler will use this constructor to create an `std::vector` containing, in this case, one string of value `"a"`.

Now, consider what happens when the definition is changed to `std::vector<int> result{1, 1}`. As brace initialisation is used, the compiler first considers a constructor with `std::initializer_list<T>` as a parameter type, as before. This time the compiler does not reject the constructor as all the braced initialisation values are of type `int` (the type specified in the angle brackets of the definition). In this case the selected constructor creates an `std::vector` with two elements, both of value 1.

What we have seen is that similar looking definitions can initialise vectors with a varying number of elements. This behaviour is regrettable and probably represents a flaw in the design of `std::vector`. It is this problem that the student has unwittingly encountered.

The troublesome statement in the student's code is just after the comment "`// first one can't be a duplicate`". Using parenthesises instead of braces will result in the appropriate constructor being called and will provide the required initialisation. The student's program will now work as expected.

When designing code to fulfil some requirement, it is always beneficial to see if library functions can be used in place of writing code from scratch. In the student's code, `unique()` contains two `if` statements and a `while` loop and is quite difficult to understand and reason about. I suggest it is possible to use functions from the standard library to perform the same

function while being simpler to understand. Using such library functions will go a long way in ensuring the software performs as required. I offer the code below as a direct replacement for the student's `unique()` function.

```
template <typename iterator>
std::vector<typename iterator::value_type>
unique(iterator one, iterator two)
{
  std::vector<typename iterator::value_type>
    result{one, two};
  const auto end = std::unique(result.begin(),
    result.end());
  result.resize(std::distance(result.begin(),
    end));
  return result;
}
```

The body of the replacement function has only three statements of any significance. The first creates a vector containing the range of values, the second moves any adjacent duplicates to the end of the vector and the fourth reduces the size of the vector by discarding the duplicate values. Finally, the function returns the processed vector.

Robert Lytton <robert@xmos.com>

Elizabeth Barrett Browning once said of C++, "How do I construct thee? Let me count the ways", and let us follow her lead with a class `T`.

```
T t1; T t2(t1); T t3=t1; T t4(1,2,3);
T t5{}; T t6{t1}; T t7={t1}; T t8{1,2,3};
T t9={1,2,3};
```

As an aside, `T`'s implementation uses the non-rule "rule of zero", outsourcing ownership to smart pointers, thus the compiler produces just what we need, including rvalue constructors such as:

```
T t2b(returnsT());
```

Phew!

Regarding all this construction Liz may "love thee to the depth and breadth and height" but I am not so sure. It seems less "smiles" and more "tears, of all my life".

So "as men strive for right" can't we cut through all of this for one true construction syntax and forget about the rest?

The brace constructor syntax arrived in C++11 and at first glance looks like a uniform initializing syntax, with benefits. As intimated above, all we need to do is replace the parentheses with braces and get a few bonus constructs for free viz:

- `T t{};` is not a function declaration but constructs an object.
- `class T { int i{0}; }` is an alternative to using a member initializer list.

We can also construct containers more elegantly.

For example, in the exercise `main()` function, the container initialisation:

```
std::vector<std::string> ptrs;
ptrs.push_back("a");
ptrs.push_back("a");
ptrs.push_back("b");
...
```

can use an `initializer_list` as the constructor argument:

```
std::vector<std::string> ptrs {"a","a","b",...
```

Hang on, what does "`initializer_list` as the constructor argument" mean?

Ah, yes. A class may declare a `T(std::initializer_list<U>)` constructor along side its other constructors. Ay, there's the rub – the compiler will do all it can to use the `initializer_list` constructor, even if it needs to convert types to make it fit and one of the other constructors is an exact fit.

For example, in the exercise `unique()` function, the vector `result` is constructed using the literal integer `1` and a value of type 'dereferenced iterator' inside of braces:

```
std::vector<typename iterator::value_type>
  result{1, *one};
```

In the first test:

- the dereferenced iterator is of type `std::string`;
- the compiler can't convert an integer to a string nor a string to an integer;
- thus it can't create an `initializer_list`;
- thus it can't call the vector's `initializer_list` constructor;
- instead it calls the `vector(size_type, const T&)` constructor.

In the second test:

- the dereferenced iterator is of type `int`;
- the compiler can create an `std::initializer_list<int>`;
- it can call the vector's `initializer_list` constructor;
- thus 'result' is initialized with two values `{1,1}`.

Hence the output when the test is run.

Changing from braces to parentheses force the compiler to use `vector(size_type,const T&)` viz:

```
std::vector<typename iterator::value_type>
  result(1, *one);
```

As `auto` with braces prefers `std::initializer_list<>` too (unless it's a return or lambda type or ...) but templates don't, it seems a uniform initializing syntax is out of reach. I have to be content with, as Liz puts it, "if God choose, I shall but construct thee better after this."

Herman Pijl <herman.pijl@telenet.be>

To start with, I don't like the name `iterator` as a name for a template parameter type. The template parameter type should use a naming convention, indicating which iterator category (e.g. `std::input_iterator_tag`) is expected by the template algorithm. Concretely, I would like to see something like

```
template<typename In>
```

I would even suggest to use a `static_assert` at the beginning of the template function definition. When the `static_assert` fails, a descriptive string would explain that assumption about the iterator category is not met, e.g.

```
{
  static_assert(std::is_base_of<
    std::input_iterator_tag,
    typename std::iterator_traits<In>::
      iterator_category>(),
    "iterator category must be (derived from)"
    " std::input_iterator_tag");
  ...
}
```

I like this 'concept' ;-).

The second problem I see in the template declaration is the use of `iterator::value_type`. This assumes that the iterator is a 'complex' type, i.e. some struct or class. In other words, the iterator cannot be a simple (plain old Kernighan and Ritchie) pointer. I would like to call the template with 'legacy' code

```
int intArray[] = {1, 1, 2, 3};
auto result = unique(intArray + 0,
  intArray + sizeof(intArray)/sizeof(int));
```

In order to achieve this, replace

```
typename iterator::value_type
```

by

```
typename std::iterator_traits<In>::value_type
```

It looks like `std::iterator` is deprecated in C++17, so get used to the `std::iterator_traits` instead.

Some comments about the template definition

The implementation starts with an attempt to find out the size of the iterator range. It is clear that if the range contains 0 or 1 element, then there can be no duplicates. Unfortunately, the `distance` algorithm is a serious overhead. I would guess that the standard implementation of 'distance' will use the `iterator_traits` to have an almost immediate answer for random access iterators, but for the other categories, the overhead is enormous.

The intention was clearly to have a (premature) optimisation for small collections, more in particular the empty collection and the singleton collection, but trying to find the size of the iterator range by entirely traversing the range is far from optimal.

Traversing the whole iterator range becomes a disaster when the iterators are traversing an input stream. As the iterators are incremented, the input stream is effectively consumed!

```
std::istringstream is("1 1 2 2 3 3");
std::istream_iterator<int> isit(is), isend;
auto result = unique(isit, isend);
```

After the call to the `distance` template, the `while`-loop will not even be entered!

More critique to come.

When the `(distance < 2)` condition is met, the template function returns a braced initialiser. As the initialiser doesn't fit the `std::initializer_list<T>` constructor, the compiler tries to find some non-explicit constructors that fit and it finds a template constructor. Personally I (still) prefer to use the traditional constructor instead of an inialiser because I want to keep control. I only use the curly braces for default construction or when I want to the construct the object with the `std::initializer_list<T>` constructor.

```
return std::vector<typename
  std::iterator_traits <In>::value_type>(
    one, two);
```

As the vector has move semantics there is no overhead. You only have to type some more characters.

Now we arrive at the bug.

```
std::vector<typename In::value_type>
result {1, *one};
```

When the value type is `int`, the initialiser (curly braces) syntax causes the compiler to find two potential constructors

- ```
  vector(size_type, const T& value,
    const Allocator&alloc = Allocator());
    // not explicit since C++11
  ```
- ```
  vector(initializer_list<T> init,
    const Allocator&alloc = Allocator());
  ```

And the Standard says that the winner is: the `initializer_list`! Unfortunately this was not the one that we wanted. You can solve the ambiguity by using the parentheses instead of the curly braces.

```
std::vector<typename In::value_type>
result (1, *one);
```

When the value type is string, there is only one constructor to consider.

At first sight I liked the trick to find the previous element

```
*result.rbegin()
```

but when looking closer to the vector template there exists a back member function.

```
result.back()
```

could be slightly more performant.

I made another version `my_unique` that should do the trick. I experimented a bit with a binary predicate.

```
#include <iterator>
#include <vector>
//extra
#include <iostream>
#include <string>
#include <sstream>
#include <type_traits>
#include <functional>
template<typename In> std::vector<
  typename In::value_type>
  unique(In one, In two)
{
  // concept
  static_assert(std::is_base_of<
    std::input_iterator_tag,
    typename std::iterator_traits<In>
                ::iterator_category>(),
    "iterator category must be (derived"
    " from) std::input_iterator_tag");
  if (std::distance(one,two) < 2)
  {
    return std::vector<
      typename In::value_type>(one, two);
      //return {one, two};
  }
  std::vector<typename In::value_type>
    result (1, *one);
  while(++one != two)
  {
    auto next = *one;
    bool is_unique =
      (*result.rbegin() != next);
    if (is_unique){
      result.push_back(next);
    }
  }
  return result;
}

//template<typename In, typename BinPred =
//std::not_equal_to<typename In::value_type>>
template<typename In>
std::vector<typename
  std::iterator_traits<In>::value_type>
my_unique(In one,
  In two,
  std::function<bool(typename
    std::iterator_traits<In>::value_type
      const &,
    typename
     std::iterator_traits<In>::value_type
       const &)> binPred =
  std::not_equal_to<typename
    std::iterator_traits<In>::value_type>{})
{
  static_assert(std::is_base_of<
    std::input_iterator_tag,
    typename std::iterator_traits<In>::
      iterator_category>(),
    "iterator category must be (derived from)"
    " std::input_iterator_tag");
  std::vector<typename
    std::iterator_traits<In>::value_type>
  result;
  if (one != two)
  {
    result.push_back(*one);
    while (++one != two){
```

```
      if(binPred(*one ,result.back())){
        result.push_back(*one);
    }
    }
  }
  // post condition: one == two
  return result;
}
template<typename In>
void test(std::ostream &os, In one, In two)
{
  auto result = my_unique(one, two);
  auto out = std::ostream_iterator<typename
    std::iterator_traits<In>::value_type>(
      os, " ");
  std::copy(one, two, out);
  os << "=> ";
  std::copy(result.begin(),
    result.end(), out);
  os << "\n";
}
template<typename T>
void test(std::ostream &os,
  std::vector<T> const &vec)
{
  test(os, vec.begin(), vec.end());
}
int main()
{
  std::vector<std::string> strPtrs
  {"a", "a", "b", "b", "c", "c"};
  test(std::cout, strPtrs);
  std::vector<int> intPtrs{1, 1, 2, 2, 3, 3};
  test(std::cout, intPtrs);
  int intArray[] = {1, 1, 2, 3};
  std::cout << "intArray ";
  test(std::cout, intArray + 0,
    intArray + sizeof(intArray)/sizeof(int));
  // auto resultForArray = my_unique(intArray
  // + 0, intArray +
  // sizeof(intArray)/sizeof(int)
  std::istringstream is("1 1 2 2 3 3");
  std::istream_iterator<int> isit(is), isend;
  auto result = my_unique(isit, isend);
  std::copy(result.cbegin(),
    result.cend(),
    std::ostream_iterator<int>(std::cout,
      " "));
  std::cout << '\n';
  std::vector<int> intPtrsOrig
  {1, 1, 2, 2, 3, 3};
  auto resultOrig =
    unique(intPtrsOrig.begin(),
      intPtrsOrig.end());
  std::copy(resultOrig.cbegin(),
    resultOrig.cend(),
    std::ostream_iterator<int>(std::cout,
      " "));
  std::cout << '\n';
}
```

Commentary

This is, as a couple of people noted, a fairly straightforward critique with one main point – the troublesome construction of **std::vector** when used with braced initialisers.

The subsidiary point, which was about preferring the standard library over hand-written code, was slightly overtaken by events for those whose compilers detected the ambiguous overload with **std::unique** for them. (This occurs when some of the contents of **<algorithm>** are included indirectly by one of the headers explicitly listed.)

My own replacement for the hand-written **unique** was:

```
std::vector<T> result;
std::unique_copy(one, two,
  std::back_inserter(result));
return result;
```

In production code which approach one might take would depend on the precise context.

I was a little surprised that no-one commented on this line:

```
auto next = *one;
```

as this takes an unnecessary copy of the object: I would prefer to see **auto const &** used here. Somehow poor use of **auto** seems be easy to overlook.

Additionally, I dislike the use of the names **one** and **two** – it would be clearer, I feel, to follow the naming convention for algorithms in the standard library and name them **first** and **last**.

Finally I would like to expand on Robert's closing remark about the interaction of **auto** and **initializer_list**. The following simple program demonstrates the issue:

```
#include <initializer_list>
int main()
{
  auto i{1};
  return i; // Is this valid?
}
```

In the published wording for C++11 and C++14, **i** is deduced as a variable of type **std::initializer_list<int>** and so the **return** statement is invalid.

The good news is that this behaviour has been changed: the relevant paper N3922 was voted into the C++17 working paper in Urbana-Champaign in Nov 2014. With this change, **i** is now deduced as **int**. Even better, it was decided to treat the resolution as a **defect** for previous versions of C++.

So this example already compiles successfully with g++ 5.1, clang 3.8 and Visual Studio 2015.

The Winner of CC 104

I was pleased to see all five entrants found the problem and were all able to explain, with more or less detail, what it was and how to fix it. Rai's reference to Scott Meyers was helpful for any who wish to read further.

Several people commented that it would be preferable to make use of the standard library; it is a good habit to get into, when you are needing a fairly standard algorithm, to start by searching the standard library to see if it has already been written!

Herman pointed out a couple of places where the existing code was insufficiently general; input iterators can cause particular problems to generic algorithms (this surfaced recently in the discussions over the changes that were necessary when adding parallel versions of various algorithms into C++17).

Overall though I think Rai picked up the largest number of incidental improvements and so I have awarded him this issue's prize.

iterators with pointers was valid and did have the side-effect of removing the undefined behaviour. I did like his introduction of a simple helper struct **NameScoreEntry** to assist with reading the data in: it is very easy to create such structs in C++ and there can be significant benefits for readability with having named fields.

Overall, I think James provided the best critique, so I have awarded him the prize.

Code Critique 105

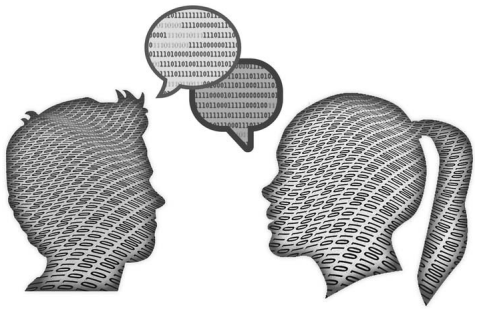
(Submissions to scc@accu.org by Jun 1st)

I was writing a simple program to analyse the programming languages known by our team. In code review I was instructed to change languages() method in the programmer class to return a const reference for performance. However, when I tried this the code broke. I've simplified the code as much as I can: can you help me understand what's wrong? It compiles without warnings with both MSVC 2015 and gcc 6. I am expecting the same output from both 'Original way' and 'New way' but I get nothing printed when using the new way.

The listings are as follows:

- Listing 3 is `programmer.h`
- Listing 4 is `team.h`
- Listing 5 is `team_test.cpp`

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 3

```
#pragma once

class programmer
{
public:
  // Add a language
  void add_language(std::string s)
  { languages_.push_back(s); }

  // the programmer's languages - original
  std::vector<std::string>
  original() const
  {
    return {languages_.begin(),
            languages_.end()};
  }

  // new - avoid copying for performance
  std::list<std::string> const &
  languages() const
  { return languages_; }

  programmer() = default;

  programmer(programmer const& rhs)
  : languages_(rhs.languages_) {}

  programmer(programmer &&tmp)
  : languages_(std::move(tmp.languages_)) {}

  ~programmer() { languages_.clear(); }
private:
  std::list<std::string> languages_;
};
```

Listing 4

```
#pragma once

#include <map>

class team
{
public:
  // Add someone to the team
  void add(std::string const &name,
    programmer const & details)
  {
    team_.emplace(
      std::make_pair(name, details));
  }

  // Get a team member's details
  auto get(std::string const &name) const
  {
    auto it = team_.find(name);
    if (it == team_.end())
      throw std::runtime_error("not found");
    return it->second;
  }

private:
  std::map<std::string, programmer> team_;
};
```

Listing 5

```
#include <iostream>
#include <list>
#include <map>
#include <string>
#include <vector>

#include "programmer.h"
#include "team.h"

int main()
{
  team t;
  programmer p;
  p.add_language("C++");
  p.add_language("Java");
  p.add_language("C#");
  p.add_language("Cobol");
  t.add("Roger", std::move(p));

  p.add_language("javascript");
  t.add("John", std::move(p));

  std::cout << "Original way:\n";
  for (auto lang : t.get("Roger").original())
  {
    std::cout << lang << '\n';
  }

  std::cout << "\nNew way:\n";
  for (auto lang : t.get("Roger").languages())
  {
    std::cout << lang << '\n';
  }
}
```

ACCU Oxford – 28 March 2017

Frances Buontempo talked about Actual Intelligence, and a few people wrote about the evening.

I really appreciated Fran travelling from London to give this thought provoking talk to this well attended ACCU Oxford monthly meeting. This focused on the philosophical issues behind AI and what intelligence is. One of the standout points for me was about how AI systems are reflecting our prejudices even though this was not the designer's intention. This is something we will have to guard against, and this will become more pressing as trust builds in AI systems e.g. in the areas of law. I can foresee governments introducing regulations regarding AI system behaviours because what is unacceptable behaviour for humans will be unacceptable for AI systems too. The challenge will be finding unprejudiced data to train the systems on. There was a very interesting observation regarding Microsoft's controversial AI chatterbot Tay which was if it had been left running would it have cleaned up its act?

Nigel Lester

I was so pleased to catch the great Fran Buontempo at ACCU Oxford, especially so as I will miss Conference.

Fran took us on a quick one hour trip around all of the significant features of the field of Artificial Intelligence, through the lens of a mathematician, scientist and coder, by asking the question, "What is Actual Intelligence?"

She explained that most of the current rash of successes for 'AI' are really just maths.

Using the fantastic Dead Salmon Study https://blogs.scientificamerican.com/scicurious-brain/ignobel-prize-in-neuroscience-the-dead-salmon-study/ she reminded us that statistics and science are hard.

The real challenge for machines and people is kindness. The fact that AI is now raising questions about what it is to be human and how to behave towards others in a way which includes non-humans is an achievement and leads to hope for benefits from the field.

The lively conversation afterwards had memorable contributions from Francis Glassborow, founding elder of ACCU, who pointed out that Human Intelligence has been responsible for much death throughout history.

The evening was rounded off with an explanation of how easy and pleasant it is to contribute to the ACCU publications from the *CVu* Editor, Steve Love.

Much thanks to Nigel Lister for organising a very well attended evening.

Tim Pizey

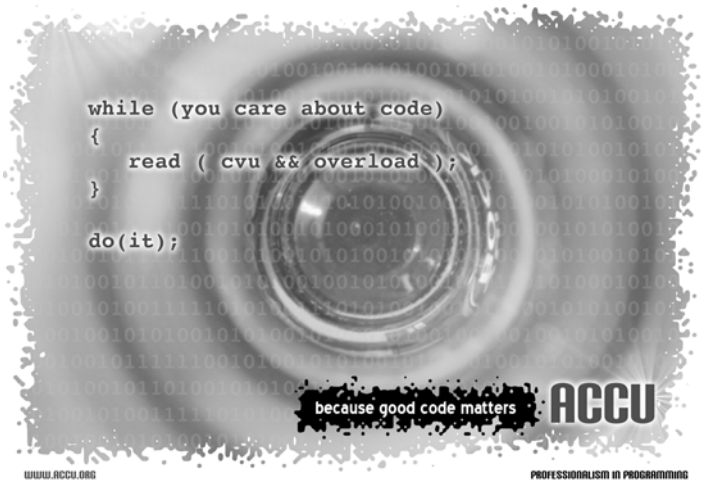
I don't often get a chance to attend the Oxford meetings of ACCU even though I live in the town because they clash with my Bridge teaching commitments. So it was a pleasure to be able to come to listen to Fran's run through of her keynote for the coming Conference. I probably had far too much to say at the end but her talk was thought provoking (as keynotes should be).

One thing I noted was that less than half the audience had read *Gödel, Escher, Bach: An Eternal Golden Braid* by Douglas R. Hofstadter. I think this book is essential reading for anyone with an interest in AI. I remember struggling as I approached halfway and wondered if it was time to put it aside and let my mind digest what I had read so far before re-reading the first part and then going on. (I find that is a great technique for dealing with technical books at the boundaries of my understanding.) However I actually managed to keep going and found the second half easier.

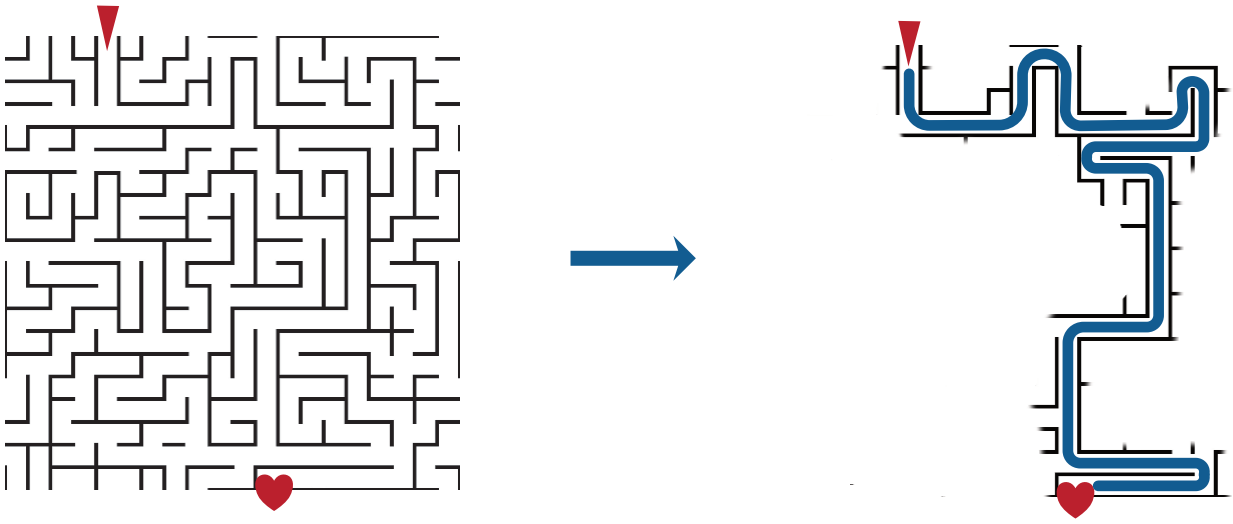
Intelligence is emergent behaviour and that whether we like it or not the current trend in computing will result in an alternative intelligence to ours. What I am also pretty sure is that we will not create an intelligent machine through our skills but through accident. I hope it is a benign intelligence because anything like our own intelligence is too dark to want (think of the mess we are making of our world). Of course that leads to the thought that a sane machine intelligence would probably want to remove at least 90% of the human race in the interests of its own self preservation.

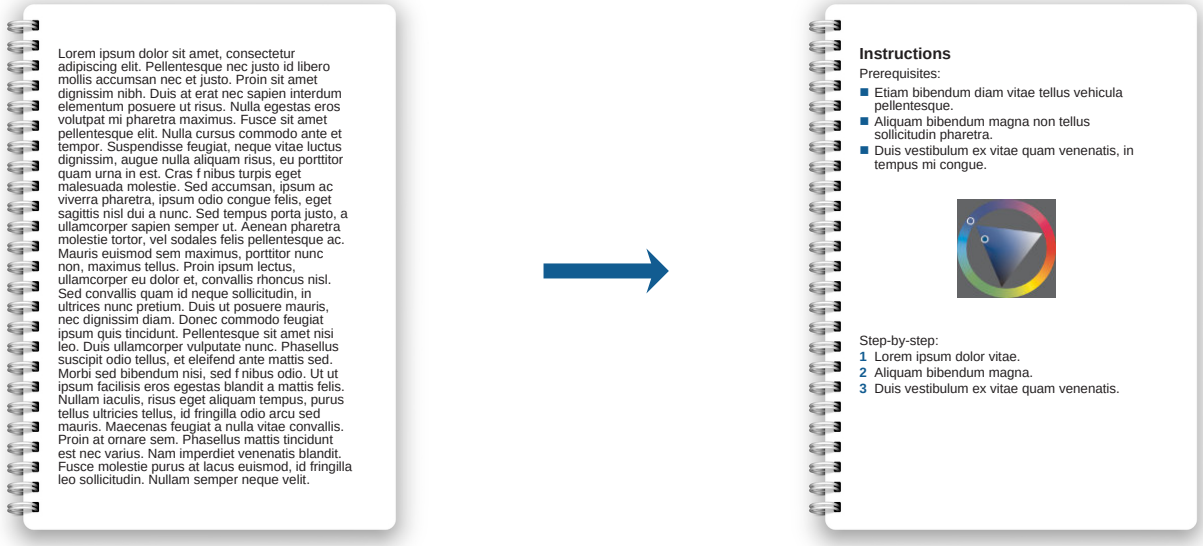
Thanks, Fran, for a thought provoking talk.

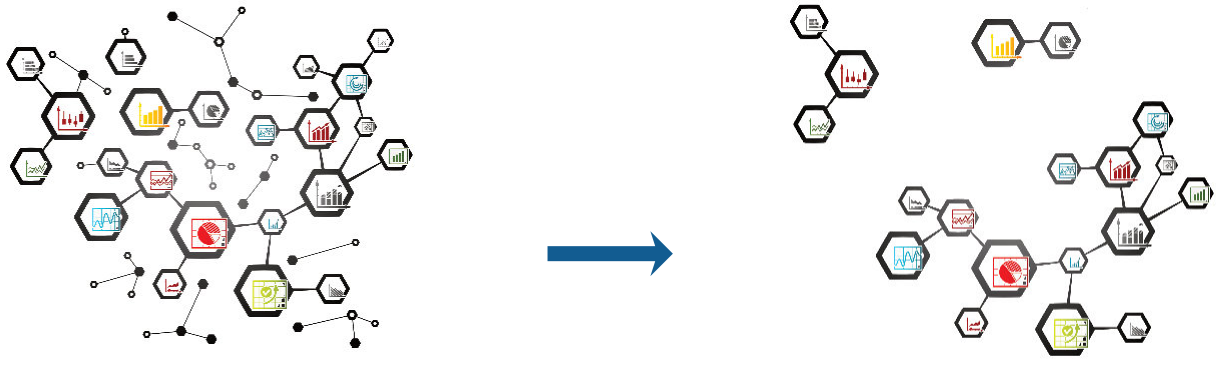
Francis Glassborow



Making the complex... ...clear







Good supporting documentation:

- Clarif es, doesn't confuse
- Provides answers, not information
- Doesn't make people hunt for solutions
- Is right for the job (text, images, video, simulations, online or printed...)

Clearly stated...

www.clearly-stated.co.uk

View from the Chair

Bob Schmidt
chair@accu.org

This instalment of the 'View' is my sixth, which means that a full year has gone by since I started as acting chair. I volunteered for the position of Acting Chair after last year's AGM, when the position remained vacant after the election. My stated main goal at that time was to "keep the organization ticking along." I will be the first to admit that, as goals go, keeping the organization ticking along is not the most inspiring, but it had the advantage of being achievable.

ACCU has had successes in the past year:

- Our website was migrated successfully to a new hosting platform, thanks to the efforts of Jim Hague.
- We published our new Diversity Statement.
- The conference web site has been reworked, and continues to improve as new features are added, thanks to our Conference Chair, Russel Winder, and his collaborators.
- Our new Code of Conduct was developed for the 2017 conference, along with guidelines for reporting and processing violations.
- Our local group affiliates have experienced strong membership growth.
- Our financial situation remains strong, with ACCU finishing 2016 with a small surplus for the fiscal year.
- Instructions on creating a Local Group were developed and published, thanks to the efforts of Nigel Lester.
- *Overload* and *CVu* continue their traditions of being high-quality publications, thanks to editors Fran Buontempo and Steve Love, and our member and non-member contributors.

We have also had our challenges:

- ACCU membership has continued its slow, gradual decline. Reversing this trend is a constant topic amongst the committee, as it has been for several years. I'm sorry to say that I have not been successful in finding a solution to this problem; I consider this my biggest failure over the past year.
- We still need to replace several pieces of our web infrastructure.
- We continue to find it difficult to recruit volunteers for committee positions. We had an acting chair and acting secretary from April 2016 until the special election in September. The Publicity, Study Groups, and Social Media positions remain vacant. The web editor position will become vacant as of the first of July.

On balance, I believe the past year has been a success, and I have every confidence that the coming year will continue the trend. This year we have a full slate of candidates for the executive members of the committee, so it is unlikely that we will start off the 2017–2018 term with a critical vacancy.

I started the role of chair without having any other experience working with the committee. (I was an auditor for one year, but that role is separate from the committee.) I'd like to thank the other committee members for their help in getting me up to speed; their comments and suggestions to make my bi-monthly Views from the Chair better and more accurate; and their commitment of time and energy to making ACCU a success.

Committee Spotlight. As I mentioned in the last *CVu*, Martin Moene is stepping down as ACCU's website editor effective the first of July. Martin has served as website editor since 2013. A quick look at the description of the role of web editor on the ACCU website (written by Martin) [1] shows how valuable his participation has been. In addition to all of that, he introduced the production of ePub versions of our two magazines.

Please join me in thanking Martin for all of his contributions and hard work. He will leave large shoes to fill.

Call for volunteers

- The ACCU web site uses Xaraya, a PHP framework that has been moribund for the last 4 years at least, and a replacement is overdue.
- We hope to recruit someone to fill the position of website editor before Martin's term concludes.
- As mentioned above, the Publicity, Study Groups, and Social Media positions have been vacant for some time.

Please contact me if you are interested in any of these positions.

Portions of this View appeared in the Annual General Meeting Information Pack as part of the Chair's Report.

References

[1] Role of Website Editor https://accu.org/index.php/members/committee/posts_and_roles#website_editor

"The magazines"

The ACCU's *C Vu* and *Overload* magazines are published every two months, and contain relevant, high quality articles written by programmers for programmers.

"The conferences"

Our respected annual developers' conference is an excellent way to learn from the industry experts, and a great opportunity to meet other programmers who care about writing good code.

"The community"

The ACCU is a unique organisation, run by members for members. There are *many* ways to get involved. Active forums flow with programmer discussion. Mentored developers projects provide a place for you to learn new skills from other programmers.

"The online forums"

Our online forums provide an excellent place for discussion, to ask questions, and to meet like minded programmers. There are job posting forums, and special interest groups.

Members also have online access to the back issue library of ACCU magazines, through the ACCU web site.

ACCU | JOIN: IN

PROFESSIONALISM IN PROGRAMMING
WWW.ACCU.ORG

Invest in your skills. Improve your code. Share your knowledge.

Join a community of people who care about code. Join the ACCU.

Use our online registration form at **www.accu.org**.

Design: Pete Goodliffe