the magazine of the accu

www.accu.org

# {cvu}

Volume 27 • Issue 3 • July 2015 • £3

## Features

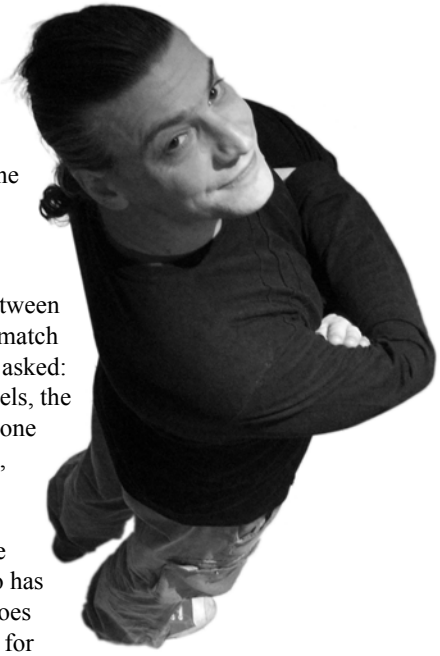


## Regulars

{cvu}
Volume 27 Issue 3
July 2015
ISSN 1354-3164
www.accu.org



# What is a user interface?



I recently bought a new amplifier for my guitar. It's great! Modern technology means it comes with a variety of different ways to disturb the peace of the neighbourhood. The three channels built-in to the box – clean, dirty, and solo which can be 'programmed' independently of the other two – all sound fantastic, but having to take your hands off of the guitar to switch between them is very inconvenient when you've been overcome by the urge to improvise wildly. So I bought a foot-switch.

It has two pedals – ideal, you'd think, for choosing between three channels. My instinct on how to use it seems to match the expectations of other players I've rant^H^H^H^H asked: one pedal to switch between the clean and dirty channels, the other switch to toggle the solo channel. My pedal has one switch which alternates between clean and dirty (tick), and the second switch to **turn on** the solo channel. Pressing it when the solo channel is already selected...does nothing at all. To switch away from the solo channel, I must press the other switch, which also has the effect of changing the main channel selection. It does have the facility to set the polarity for each pedal, and for each switch to be latched or not...but I'll never make use of those features.

In days of yore and valve amplifiers, the foot switch was often a mechanical one which enabled a boost in the power amp, making solos a little louder. All that's really different in my pedal is the extra complexity of also having clean and dirty channels as options. I was a little surprised that the foot switch requires a battery (perhaps to power the LED?), but I'm left with the nagging suspicion that the manufacturer has taken a simple requirement, and come up with something much more complicated than it needs to be, and doesn't behave in quite the way I expect.

There is something familiar about that, but I can't quite put my finger on it...

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 27.4** 1st August 2015
**C Vu 27.5:** 1st October 2015

**Overload 129:** 1st September 2015
**Overload 130:** 1st November 2015

# 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

# Dictionary and Thesaurus

Chris Oldwood finds surprising similarities in prose and code.

One of the reasons I reckon I got into computer programming was to avoid having to write – not code, but actual English prose. I hated it. At school, when I was 16, I had to take exams in both English Language and English Literature and I did phenomenally bad – I got a 'U' in both. For those unfamiliar with the English education system circa 1986 a 'U' means Unclassified. It means I did so badly that they can't even give me a grade – even an 'F', the lowest grade, would be too good for me. Of course, I had to take it again as a 'C' grade was mandatory to get anywhere, such as University, and I got it, eventually.

Computer programming on the other hand was great. Back in the '80s, I was a bedroom coder bashing out nuggets of assembler on various 8-bit micros. The best thing about assembler was that there weren't many mnemonics to learn and they weren't spelled properly anyway (e.g. MOV, SUB, JP, etc.) or they were acronyms (e.g. LEA – Load Effective Address). If you did misspell anything the assembler would probably pick you up on it right away as the code wouldn't compile (should that be "assemble"?). Transposition-style errors were possible but it was more likely that you got the logic wrong than the 'spelling' of the code.

## The writing's on the wall

Wind the clock forward another 10 years and we're into the realms of Object-Oriented languages and silly limits like 8-character long identifiers have become a relic of the past. The idea of Design Patterns has sprung up to give us a common vocabulary with which to discuss recurring design problems. Also we're now talking about domain modelling so that we can represent both abstract and concrete forms of business concepts in our code as classes and functions to solve much bigger problems. The world I was residing in at that moment in time was still in the territory of 'office automation', but the problems we were solving were more abstract as we tried to make sense of how the Internet was adding a technical spin around historically mechanical processes. Suddenly I discover that real language is everywhere in my job.

The wake-up call for me that my weak natural language skills were a genuine problem was seeing how others had started to work around my limitations. In a number of cases, one developer had created `typedef`s (aliases, for non-C++ programmers) as a way of correcting the spelling of some of my class names. Back then there was no decent IntelliSense to guide you, and on a large codebase hitting the compiler only to find that you'd not misspelled a type consistently costs you a non-trivial amount of time. And these classes were in the core library. Another stand-out moment was around my confused use of the words 'license' and 'licence' such that the noun and the verb were misused badly enough to create configuration and code hell for those that knew the difference.

If that wasn't bad enough, working with a colleague who was using Whole Tomato's Visual Assist product, which not only spell checks your comments and string literals, but also your class and method names (including compound words!), was a real eye opener. What I selfishly thought was me just being a bit quirky, perhaps even slightly endearing, was fast becoming an embarrassment. Everywhere I looked now there were little red squiggly lines under what I was writing – be it code or prose.

## Turning the tables

In what could probably be viewed as a poacher-turned-gamekeeper reversal of roles I now find myself on the other side of that fence. And the situation is more unpleasant than I ever suspected. In an even more ironic twist of fate, I find myself writing more documentation now that I'm 'doing agile' than I ever did before.

It was over 10 years before I even saw a technical or functional spec. and I only added a single paragraph to the one I did come across. No, the way I focused my efforts (initially) on writing was to put together lengthy diatribes about the state of the codebase and the lack of good engineering practices being used, i.e. go all passive-aggressive. Whilst this ended up having little-to-no impact on changing the attitudes within the team, it did give me an outlet with which to practise writing proper prose. I coupled this with putting together a developer (rather than support) focused wiki to document some of the more common gotchas, such as how to merge integration branches efficiently in ClearCase, or diff for all changes between two labels. I also started to follow the advice of Record Your Rationale from the book *97 Things Every Software Architect Should Know* [1]. My desire to write short, but more importantly clear, documentation was also largely prompted by working in a multi-cultural team for the first time.

Naturally when you start paying closer attention to such matters in your own code you also begin to notice those deficiencies in other people's output too. I've had to fix 'eclipsing' issues in ClearCase due to random capitalization, e.g. Counterparty vs. CounterParty, missed diagnostic clues due to misspelled words not matching log file greps, and hunting longer in source code repos when doing some software archaeology [2] due to badly written commit messages. Having non-native English speakers working on the codebase too was also an eye opener as I struggled to explain why some of the code they wrote just didn't read particularly well. Up to that point I hadn't realised it was even possible to write 'grammatically incorrect' code and I felt more than a little churlish about bringing it up.

And so finally we come to two very old fashioned 'tools' that I've since grown to depend on...

## The dictionary

Once upon a time a dictionary was a cutting-edge feature, even for a word processor or desktop publisher, and a stand-alone dictionary application would cost serious money. These days they are often built into FOSS tools like Notepad++, and the commit dialog for the TortoiseXxx VCS extensions has one too so there is no excuse for misspelling a commit message unless your language is unsupported. Chrome was the first browser I used which had dictionary support for text boxes that made writing blobs of text like blog post comments easier as you no longer had to paste the text into Notepad++ or Word just run the spell checker over it.

I mentioned earlier that spell checking wasn't restricted to just prose either, it could also be performed on code. And not just comments and string literals, but also on class, method and variable names too by tools like Visual Assist. In the intervening years I've seen a few free plug-ins to the popular IDEs that will handle comments and string literals, but sadly they won't have a stab at identifiers which I think is a shame. Perhaps too many developers still write code with overly terse names? For code running on the .Net platform the FxCop tool is one of the few exceptions, it also allows you to add your own domain-specific acronyms and terms to a custom dictionary to minimise the false positives.

**CHRIS OLDWOOD**

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

# Coding Dinosaurs

Pete Goodliffe aims to outlive the jurassic coding age.

In a recent interview for a well-known website, I was asked a number of practical questions about the art and craft of programming. One of the questions struck a chord with me, and seems particularly interesting for the programmer who cares about 'becoming better'.

> The technologies we work with are constantly changing and it's all too easy to find yourself becoming something of a 'coding dinosaur'. What can programmers do to ensure that they keep learning and developing their skills?

I found this a particularly interesting question, since right now I'm learning new stuff that isn't necessarily strictly about coding. I'm learning how to manage teams and projects, and make sound higher-level technical decisions. This, like learning the gritty details of code, is fun. But it's a different type of fun.

More akin to the type of fun you can have herding stubborn cats.

My current focus very specifically takes me away from the learning of new code techniques, the stuff I love. I can still wield my familiar coding tools well, and I use them regularly. But I don't have the same time to invest in learning about them.

Am I in danger of becoming that dinosaur?

As a programmer, as in any other field, in order to avoid stagnation you have to take conscious, deliberate action. You will not avoid stagnation by doing nothing. Doing nothing is the very act of stagnating.

> In order to improve, you must make a conscious effort. you won't learn by accident.

**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

## Dictionary and Thesaurus (continued)

As for spell checking files using a command-line based tool, there is Ispell, Aspell and more recently Hunspell. Whilst these are suitable for checking prose, I'm only aware of Aspell as having support for checking code (and even then only C/C++ comments and literals).

When it comes to checking the spelling of a word on my smartphone, which has a reasonable but not extensive dictionary, I end up reaching for Google. By default just searching for a single word will suggest an alternative if spelled incorrectly and the first hit is usually the definition, although you can Google "define <word>" to ensure the first hit is the definition. However, that always feels like the proverbial 'sledgehammer to crack a nut' but I'm too much of a cheapskate to cough up for a decent dictionary app (assuming I can find room on the device for it, but that's another story).

### The thesaurus

My wife's English teacher once told her never to use the words get, put or nice. In his opinion there are so many better synonyms in the English language that a writer should never feel the need to use them. From a software development perspective, I can't say the word 'nice' has been much of a problem but the other two seem to crop up with alarming regularity.

When it comes to naming properties of a class (in programming languages where properties are not a first class concept) you could be forgiven for thinking that the `getValue`/`setValue` pair is mandatory. As a consequence of this there is also a school of thought that any method which is prefixed with `get` is therefore a property and so likely comes with a similar performance guarantee too.

In the online thesaurus I looked at, there were pages and pages of synonyms and related words for 'get'. Off the top of my head I can think of a number of really common alternatives that I use regularly, such as: create, make, build, acquire, fetch, locate, find, retrieve, request, derive and calculate. You can probably already see a few patterns here as the initial ones revolve around the creation of objects, the middle few with looking up things and the final couple for doing some form of processing.

Although there are no formal guidelines about what semantics any of these words might suggest (despite at least one attempt by yours truly [3]) I would hope that they provide at least a little more insight than the weaker 'get'. For example the words fetch, request and retrieve all hint at some form of more complex operation than simply returning the value of a member variable. Hopefully the level of complexity hinted also suggests that failure is probably on the cards and so may need to be factored in.

Whilst code may not be prose, little variation in the verbs used in method names makes code read very monotonously. Test names that follow a classic given/when/then format will often also end up being very mechanical in nature. You may have a bit more leeway with test names, but that doesn't absolve you of being economical with the language to succinctly convey the behaviour under test.

These days a thesaurus is always within easy reach if you have an internet connection and it only takes a few moments to search and find something appropriate for the task in hand. However, sadly they are far less accessible than a dictionary within text editors and browsers. Hence I often have a copy of Word lying around in the background so that I can switch to it, type the word I'm looking to replace and hit shift-F7 to get a bunch of alternatives within seconds. Being a native application it also takes up far less resources than another Chrome tab.

### Epilogue

My journey as a programmer took an unexpected turn just over a decade ago as I finally found myself unable to avoid the similarities between writing readable code and prose. Instead of being a chore, I have actually found the experience of 'raising my game' quite liberating. In fact where once I found the idea of learning about natural languages quite unappealing, I now find the subject far more attractive exactly because there are many parallels with programming. And whilst I may not write code, tests or documentation to rival a best-selling novel, I hope that the extra attention to detail adds precision and clarity that ultimately benefits the reader. ■

### References

[1] http://97things.oreilly.com/wiki/index.php/Record_your_rationale

[2] 'In The Toolbox: Software Archaeology', *C Vu* 26-1

[3] http://chrisoldwood.blogspot.co.uk/2009/11/standard-method-name-verb-semantics.html

# Are we nearly there yet? Refactoring C++

Alan Griffiths evaluates two tools for developers with some simple use-cases.

## Refactoring

The idea of invariance under transformation is ancient and the fundamental concept behind the mathematical representation of symmetry. It has applications in many disciplines: physics employs the symmetries of eleven dimensions for 'string theory'; music employs various temporal and tonal symmetries; even politics references the symmetry of 'right' and 'left'.

In software development there are various symmetries, but one important class of transformations are the refactorings of code. I've used these over many years but hadn't recognised their significance until I encountered Martin Fowler's book of that name [1] around the turn of the millennium.

Shortly after that I was working in Java and encountered the first attempts at providing automation of refactoring transformations of code in development tools: first in Eclipse and then in JetBrain's IntelliJ IDEA. The experience was a revelation to me about how I'd been avoiding making transformations to the sourcecode simply because it was too much effort to keep track of things.

To take a simple example: extracting a piece of a function into a new, more focussed, function. I'm sure you've been there too: a function starts out with a clear purpose but gradually grows over time until the first part of it does one thing and the rest something that follows. Then some code doesn't need the first part done that way and a conditional branch is added.

If we were starting from scratch, the two parts belong in separate functions but it requires work to split them out: one needs to work out the parameters that need passing, track the scope of local variables, create function prototypes, and move the code. None of that is hard, but the 'housekeeping' takes headspace and the effort can interrupt the more urgent task at hand.

The important thing to know about refactorings is the invariant: they don't change the meaning of code. It behaves exactly the same before and after the refactoring – only the design changes. I mentioned the 'Extract Method' refactoring above – there's a corresponding refactoring 'Inline Method' that replaces each call of a method with the method body. Refactoring isn't a goal in itself: you use a refactoring to reliably change the code in a way that makes it more useful for something the code doesn't currently support.

What refactoring tools gave me when working in Java (and later in C# and Python) was the ability to select the transformation I needed and let the

**ALAN GRIFFITHS**

Alan Griffiths has been developing software through many fashions in development processes, technologies and programming languages. During that time, he's delivered working software and development processes to a range of organizations, written for a number of magazines, spoken at several conferences, and made many friends. He can be contacted at alan@octopull.co.uk

# Coding Dinosaurs (continued)

If you care about improving your skills, if you care about becoming better, then you must specifically set aside time to learn, and time to perform deliberate practice, the kind of directed, specific learning that is more thoughtful and more productive than mere 'playing with stuff'. Good examples are to invest time learning new technologies, and trying them out, and to performing code katas that reinforce your existing knowledge and help you build up your skills.

If you do not have a specific plan of action to do this, you may very well never get around to learning! If you don't intend to invest time, you may quickly find yourself swept up in the details of what you already know, working your day job, whilst the techie world moves on around you.

For me personally, I currently have a plan in place to:

- Read a couple of new techie books I've purchased, one on software archaeology, and one on learning new languages. I chose these two topics specifically as things that will focus me on learning important new ideas. (I love carrying a physical book around with me to dip into when I'm on a train.)
- Subscribe to a podcast about C++, to keep abreast of new ideas in the language I am most familiar with.
- Spend an hour or two at least one evening a week playing with an entirely new language, just to begin to learn the way it works. This isn't reading about it, this is physically writing some code. Of course, in such a time I'll never learn the intricacies of the language. I'll barely even scratch the surface. But it will inevitably expose me to new ideas that will teach me.

That's my specific plan. It's not too taxing, but it's also enough to encourage me to keep learning.

What's your plan?

> Do you have a specific plan for learning new coding skills? How do you ensure that you stick to this plan? If you don't have a learning plan, consider creating one right now!

Ultimately, the very desire to avoid becoming a 'coding dinosaur' is the key thing. You must stoke your passion for coding. As soon as you feel it's boring or old-hat, you'll have no desire to improve, and no motivation to learn more.

That's where stagnation happens.

I find that my passion wanes when I am no longer learning, and operate out of what I already know.

The techie world doesn't move so fast that your skills will lose relevance overnight. However, if you don't pay attention to continuously learning, and invest directed effort, you can easily begin to slide out of usefulness. ■

## Questions

1. How much attention do you pay to directed, specific learning?
2. Are you particularly worried about becoming a coding dinosaur?
3. Does your daily programming job require you to learn new stuff constantly? If not, why not? And how might you weave more learning into your tasks?


Pete's new book – *Becoming a Better Programmer* – has just been released. Carefully inscribed on dead trees, and in arrangements of electrons, it's published by O'Reilly. Find out more from http://oreil.ly/1xVp8rw

editor/IDE to do the job. When you can do that with the same assurance that you can use automated indentation or 'auto complete' you don't have to worry about breaking the code or interrupt your thinking about the code.

## But what about C++?

With C++ being the *lingua franca* of ACCU I shouldn't need mention that C++ is hard – not just for developers, but also for the writers of development tools. There's a reason that C++ compilers are orders of magnitude slower than Java compilers at processing lines of sourcecode: the language is more complex. (I don't think anyone has even dreamt of implementing Eratosthenes Sieve at compile time in Java.) And the preprocessor provides lots of opportunity for confusing tools.

So, although over a decade has passed since I used refactoring tools in Java, whenever I start a C++ job I have to revert to making these transformations the hard way.

## Eclipse CDT and Cevelop

Of course, C++ hasn't been ignored completely for a decade and a half and work (notably that led by a well known ACCUer: Peter Somerland) has continued on implementing C++ support as Eclipse plugins. CDT [2] is the original project and Cevelop [3] is based on CDT as packaged by Peter's group at the Institute for Software.

I'll give some examples below, but while the support this gives is better than nothing it isn't the reliable refactoring support needed to avoid breaking the flow. In practice I find that I need to commit the state of the code before any refactoring to avoid losing work and build and run the test suite afterwards. Some very strange things can happen and, if problems occur during the refactoring, the editor can revert the code to an old (or even broken) state losing recent changes.

In spite of these problems I have made a lot of use of Eclipse CDT and Cevelop over the last few years. (Not that it has entirely separated me from vim – I use different tools for different tasks.)

## JetBrains CLion

I mentioned JetBrains above: they are the company behind some popular refactoring IDEs and plugins for Java, C# and Python. Last year they announced a beta program for their C++ refactoring tool – and based on my past good experience with their tools I signed up. However, I didn't use their IDE much during the beta program mostly because it was very slow – at one point taking tens of seconds to respond to each key-press.

But CLion was recently released and I downloaded the evaluation version. I still had some problems getting it to work but as they were exhibiting at the ACCU conference and took an interest in the problems I was seeing: I got to a point where I could evaluate it. There are still some performance problems (mostly a 5 minute start-up time on my current project) but it proved usable.

As with Cevelop the automated refactoring can go badly wrong (again I'll give some examples), but so far I've always been able to fix or just revert the refactoring without problems.

Listing 1

```
$ bzr diff
=== modified file 'playground/demo-shell/window_manager.cpp'
--- playground/demo-shell/window_manager.cpp 2015-05-13 07:23:52 +0000
+++ playground/demo-shell/window_manager.cpp 2015-05-17 14:55:49 +0000
@@ -196,6 +196,12 @@
   surf.resize({right-left, bottom-top});
 }

+void select_next_session() {
+  focus_controller->focus_next_session();
+  if (const auto surface = focus_controller->focused_surface())
+    focus_controller->raise( { surface });
+}
+
 bool me::WindowManager::handle_key_event(MirKeyboardEvent const* kev)
 {
   // TODO: Fix android configuration and remove static hack ~racarr
@@ -210,9 +216,7 @@
   {
-    focus_controller->focus_next_session();
-    if (auto const surface = focus_controller->focused_surface())
-        focus_controller->raise({surface});
+      select_next_session();
     return true;
   }
   else if (modifiers & mir_input_event_modifier_alt &&
```

Listing 2

```
$ bzr diff
=== modified file 'playground/demo-shell/window_manager.cpp'
--- playground/demo-shell/window_manager.cpp2015-05-13 07:23:52 +0000
+++ playground/demo-shell/window_manager.cpp2015-05-17 15:11:15 +0000
@@ -196,7 +196,11 @@
   surf.resize({right-left, bottom-top});
 }

-bool me::WindowManager::handle_key_event(MirKeyboardEvent const* kev)
+bool void WindowManager::select_next_session() const {
+  focus_controller->focus_next_session();
+  if (auto const surface = focus_controller->focused_surface())
+    focus_controller->raise({surface});
+} me::WindowManager::handle_key_event(MirKeyboardEvent const* kev)
 {
   // TODO: Fix android configuration and remove static hack ~racarr
   static bool display_off = false;
@@ -214,7 +214,7 @@
   {
-    focus_controller->focus_next_session();
-    if (auto const surface = focus_controller->focused_surface())
-      focus_controller->raise({surface});
+    select_next_session();
     return true;
   }
   else if (modifiers & mir_input_event_modifier_alt &&
```

I'm going to concentrate on Cevelop in what follows, but a lot of what I say about it will also apply to Eclipse-CDT.

## My current project: Mir

I'm currently working on an open-source C++ project – which means that you can download the code [4] and experiment for yourself. It is part of Canonical's 'Ubuntu Touch' phone operating system and intended to form part of a future 'converged' desktop and phone version of Linux.

The Mir code is mostly C++11 with some of the more recent code using bits of C++14. There are also a few C99 files around as we provide a C API. It isn't an enormous project, but larger than many: about 250KLOC.

## Starting off

There's an immediate difference between the startup of CLion and Cevelop as CLion recognises the project's build system (CMake) and sets up an out-of-tree build environment. That's nice, but it is a happy coincidence for me: if I needed to work on a project using an unsupported build system then I'd have to look elsewhere.

Cevelop in contrast gives the user the freedom to specify the build commands and the responsibility to set up the build environment.

Both then start scanning the codebase for symbols and until that finishes a lot of features are unavailable. There is a difference if the IDE is closed down and reopened as Cevelop appears to cache its 'index' whereas there is the same delay every time CLion is started.

## Refactoring

I tried a bit of 'Extract Method' in one of the messier functions in the codebase and both IDEs failed horribly.

Listing 1 shows it in Cevelop. For reasons that are not obvious to me the new function isn't a member function. (Even though it references the `focus_controller` member variable!)

Listing 2 shows the same in CLion. Not impressive either!

In neither case does the 'refactored' code even compile. In one sense that is good, as it flags up the problem; in another sense it is bad as for automated refactoring to be useful it is essential that it preserves the existing functionality. I tried a number of other examples with equally odd results.

Listing 3

```
$ bzr diff
=== modified file 'examples/server_example_canonical_window_manager.cpp'
--- examples/server_example_canonical_window_manager.cpp 2015-05-14 12:29:13 +0000
+++ examples/server_example_canonical_window_manager.cpp 2015-05-17 15:51:29 +0000
@@ -113,7 +113,8 @@
 -> ms::SurfaceCreationParameters
 {
    auto parameters = request_parameters;
-  parameters.size.height = parameters.size.height + DeltaY{title_bar_height};
+  auto title_bar_delta = DeltaY{title_bar_height};
+  parameters.size.height = parameters.size.height + title_bar_delta;

    auto const active_display = tools->active_display();

@@ -198,8 +199,8 @@
          parameters.top_left.y = display_area.top_left.y;
    }

-  parameters.top_left.y = parameters.top_left.y + DeltaY{title_bar_height};
-  parameters.size.height = parameters.size.height - DeltaY{title_bar_height};
+  parameters.top_left.y = parameters.top_left.y + title_bar_delta;
+  parameters.size.height = parameters.size.height - title_bar_delta;
    return parameters;
 }
```

Listing 4

```
$ bzr diff
=== modified file 'examples/server_example_canonical_window_manager.cpp'
--- examples/server_example_canonical_window_manager.cpp 2015-05-14 12:29:13 +0000
+++ examples/server_example_canonical_window_manager.cpp 2015-05-17 16:01:41 +0000
@@ -197,8 +197,9 @@
       if (parameters.top_left.y < display_area.top_left.y)
         parameters.top_left.y = display_area.top_left.y;
    }
-
-  parameters.top_left.y = parameters.top_left.y + DeltaY{title_bar_height};
+  mir::geometry::DeltaY title_bar_delta = DeltaY
+  { title_bar_height };
+  parameters.top_left.y = parameters.top_left.y + title_bar_delta;
    parameters.size.height = parameters.size.height - DeltaY{title_bar_height};
    return parameters;
 }
```

I had better luck with 'Extract Variable', first CLion (Listing 3).

While I'd rather see `auto const title_bar_delta` that's not bad – although I selected the second instance of `DeltaY{ title_bar_height}` it replaced all three correctly. Not so good with Cevelop (Listing 4).

This only replaced the selected use of the expression and offers up some eccentric formatting.

I won't show any more excerpts, the story is that sometimes the 'refactoring' truly preserves the meaning of the code and sometimes it breaks it. It can be used in both tools but you need to think about it and check what the tool has done.

I started with a question: 'Are we nearly there yet?' the answer is a disappointing 'nearly'. ■

## References


[1] Fowler, Martin (1999) *Refactoring: Improving the Design of Existing Code*, Addison Wesley

[2] http://eclipse.org/cdt/

[3] https://www.cevelop.com/

[4] http://unity.ubuntu.com/mir/

# Golang programming on AppEngine

## Silas S. Brown tries his hand at writing native code for the Cloud.

Go [1] is Google's new C-like programming language which compiles to machine code but with various built-in safety mechanisms. Their 'AppEngine' public cloud servers currently support Python, Java, PHP and Go, so Go is the only option if you want the speed of machine code (albeit with some added safety checks) and don't want to set up a full virtual machine. (Google will let you set up a full virtual machine but at an extra cost, and RedHat's OpenShift offering allows you to run arbitrary binaries, but of course the rules could change at any time.)

If using AppEngine, you can write code with different modules in different programming languages, which is useful if you already have most of it in Python or Java and just want to optimise a small part of it into Go. But different modules have different `.yaml` files and handle different URL patterns, so the communication between modules will likely involve URL fetching, which might contribute to your quota use depending on how exactly they calculate it (which could change).

Go's syntax is very much like C, the most immediately obvious differences being:

1. Type names are put after variables and functions, not before as in C. So the C code: `int myFunc(int a, int b)` would be written in Go as: `func myFunc(a int, b int) int` and the C++11 statement: `auto c = my_expression();` would be written in Go as simply: `c := my_expression()`
2. Semicolons are automatically added at the end of lines. This means you have to be careful where you do and don't put newlines. For example, if writing `} else {` it must be all on one line, which is contrary to the C styles of some, although I for one tend to write that all on one line anyway.
3. Braces are compulsory, even when there is only one statement in the block. You can't write `if (a) b();` – you have to put braces around `b()`. On the other hand, the parentheses around the `a` are optional.
4. There is no `while` keyword, but writing `for` with one argument effectively turns it into a `while`.
5. Unsurprisingly for a 'safe' language, there is no pointer arithmetic. Most things you did in C with pointer arithmetic have to be done in Go with arrays, but thankfully there is much added support for array operations. 'Slices' in Go can expand as needed, but I/O functions tend to take their current size as a cue for how much data you want to read, so one caveat is not to accidentally pass an empty array to a read function and expect it to read anything! If you need to read a string as an array of bytes, you probably have to copy it like this (unless there's a better way I don't know about):

```
myBytes := make([]byte, len(myString))
copy (myBytes, myString)
```

6. `++` and `--` are statements, not expressions, so (unlike in C) you can't write them in the middle of a longer expression. You can still write `myVar++`, but this now has to be a statement on its own.

**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

Go's 'packaging' system works as follows: Every `.go` source file provides a package (the main package is `main`), and can 'import' other packages from other `.go` files. One package can be split across more than one `.go` file. The compiler automatically searches the current directory for all `.go` files (you don't have to tell it which ones to look at). You're not allowed to mix files from different packages in the same directory, so non-standard packages you wish to import must be placed in subdirectories. There are also facilities to automatically download packages from various public source-control systems, but versioning can be an issue so you might prefer to make local copies. At any rate the resulting binary will always be statically linked (Go doesn't 'do' shared libraries), which, while making for larger binaries, might at least mean you don't have to worry so much about installing different versions of shared libraries on other people's systems.

Go's standard library is strong on Internet data processing and includes support for concurrency and synchronization. Mark Summerfield wrote a more in-depth article about concurrent programming in Go in *Overload* 3 years ago [2], but if you're just writing a simple server for AppEngine then perhaps all you really need to know is that multiple threads of control might be running through your code so it's best to avoid mutable shared resources (or use locking as a last resort).

If you are writing for AppEngine then I'd suggest you download AppEngine's version of Go, which has the AppEngine-related packages already included, but you could also install the 'golang' packages in your GNU/Linux package manager, and there are versions for the Raspberry Pi. The golang-doc package provides a web server for offline browsing of the documentation; this is also included in AppEngine's Go download.

Compilation speed is quite fast, but you do need enough RAM: while experimenting with a Go back-end for my Annotator Generator [3], I found the compiler can take between 400 and 500 times the size of its input program in RAM. Because Go is a garbage-collected language, it's possible to manage on less RAM at the expense of more CPU by running the GC more frequently, but the problem is most operating systems don't have a standard way of asking 'are we running out of RAM yet': if you allocate too much, then typically you will either take the huge speed penalty of being extensively swapped out to virtual memory without being told (unless VM has been disabled), or else the OS would sooner kill your process entirely than tell you about the memory shortage. The standard 'fix' for this (which is also used by the GCC compiler) is to query the system's specification for total physical RAM size, and set your internal limits accordingly. This can go wrong if too much of that physical RAM is in active use by other programs or is otherwise unavailable, and in this case it can be necessary to manually trick the compiler or other program into reading an artificially reduced value, otherwise it will sprawl itself out into swap space without knowing. ■

## References and notes

[1] www.golang.org. Some prefer to call it Golang (Go Language), as I have done in the title of this article because 'go programming' seems more open to misunderstanding.

[2] Mark Summerfield: Concurrent Programming with Go. *Overload* #106 (December 2011)

[3] 'Web Annotation with Modified-Yarowsky and Other Algorithms', *Overload* #112 (December 2012); code is at http://people.ds.cam.ac.uk/ssb22/adjuster/annogen.html

# EuroLLVM Conference 2015

## Ralph McArdell reports on his experience of the LLVM Conference.

A week or so before the 2015 ACCU conference, I attended the 2015 EuroLLVM conference. LLVM [1] and associated projects such as Clang [2] are all about computer language translation infrastructure with LLVM itself and many of the associated projects being open source. The conference was held over two days in the Hall Building of Goldsmith College in New Cross, London, UK on Monday and Tuesday the 13th and 14th April 2015. This was my first time at an LLVM conference and I attended because I am interested in LLVM and Clang and wanted to know more. The conference was reasonably priced at £60+VAT for the two days, and it was held conveniently close to me. As someone wanting to know more about LLVM, Clang, *et al.* my main interests were in overview and tutorial sessions rather than the hard technical sessions.

### Monday

Conference registration started at 09:00, with refreshments provided from 09:30. The morning was taken up by what was called a 'Hackers Lab' – which presumably was for hard core LLVM, Clang and related projects' developers. Not being such a developer and having registered, attached my name badge and grabbed a coffee and a pastry snack, I joined many other delegates in hanging around in the common area. There were a number of posters detailing LLVM related endeavours by various organisations which had been put up in the common area. I started reading one titled 'LLVM for Deeply Embedded Systems' by people from Embecosm [3] and Myre Laboratories [4]. I knew the name Embecosm and recognised one of the authors – Jeremy Bennett – from the Open Source Hardware User Group (OSHUG) [5] and the Parallella [6] SDK forums. I made a comment to a guy standing next to me and we got chatting – he was from Germany and was interested in the static analysis tools Clang provides as he had an unfamiliar large ball-of-mud code base to maintain.

After lunch the conference proper started with a keynote given by Francesco Zappa Nardelli on the trickiness of concurrency in C and C++ even post C11 and C++11. Having revised the memory model, atomic operations and memory orderings that came in with C11 and C++11, we were reminded that certain sorts of compiler optimisations can produce incorrect code in concurrent contexts and told that these sorts of compiler bugs cannot be caught with the current state of compiler testing. Francesco then went on to assert that the problem can be reduced to searching for transforms of sequential code that are not sound for concurrent code and checking for changes to runtime events. A tool has been produced – CppMem [7] (I think) – that can check for these sorts of problems. The talk closed with the take-away that the formalisation of the C and C++ memory model has enabled compiler concurrency testing, and correctness of memory order mapping. However, there is a need to find out what compilers implement and programmers rely on – and please would we take the survey (I did not remember to).

Following right on from the key note were the first of the sessions – with three parallel streams. I ran up stairs to the room where Eric Christopher and David Blaikie were giving a debug info tutorial where I discovered that DWARF [8] is the primary debug information format used by Clang – the C languages front end that uses LLVM as a compilation back end, and as it is a permissive standard – meaning there are many variants – applications that consume DWARF information such as debuggers are not generalised but tend to be tied to the tool that generated the DWARF information. The main point of the tutorial was to introduce us to the LLVM `DIBuilder` class that eases the pain of adding debug information to a program's compilation output, with useful hints such as build source location information into the design from the get go as it is difficult to retrofit.

None of the sessions following the mid-afternoon refreshment break seemed to be introductory or tutorial in nature, so I went to a talk given by Mattias Holm on T-EMU 2 [9] – billed as the next generation of LLVM based microprocessor emulator. T-EMU 2 uses C++11 and, unsurprisingly, the LLVM toolchain throughout. Currently T-EMU 2 only supports SPARC processors and like many tools and projects based on LLVM, is library based and provides a command line interface. While the interpreted instruction implementation only yields around 10 MIPS performance this can be raised to around 90 MIPS by using a threaded and optimised approach. It is hoped to raise performance to an estimated 300 MIPS by moving to binary translation.

The main points we were supposed to take on board were that using the LLVM TableGen tool to emulate cores coupled with the use of LLVM intermediate representation (IR) led to rapid emulator development. As a LLVM neophyte I also took away the notion that TableGen – which I had seen used during LLVM builds – seemed like something worth looking into further [10]. Mattias ended by noting that TableGen is not fully documented causing people to resort to reading the code, and that LLVM IR assembler is hard to debug.

For the final session of the day I chose Zoltan Porkolab's talk on Templight [11] (in fact Templight 2) – a Clang extension for debugging and profiling C++ template metaprograms – which sounded pretty much my sort of talk! The Templight developers have patched Clang to add options for Templight. Compiling C++ code with the Templight options active causes a trace file in XML format to be produced that can be used as input to front end analysis tools. The current tools have been developed using Graphviz and Qt and allow template instantiations to be displayed and analysed in a step by step fashion or instantiation timings and memory usage to be analysed. The Metashell project [12] uses Templight to provide an interactive template meta programming REP and there is a Templight rewrite by Mikael Persson available on GitHub [13].

In the evening there were drinks and dinner at the London Bridge Hilton hotel. As there were a couple of hours to spend before the drinks I adjourned to a local pub with a couple of people I had met for a chat and a drink. At dinner I sat next to Andrew Ayers from Microsoft's .NET team – who was giving a talk the following day on CoreCLR garbage collection support in the LLVM MSIL compiler – a talk I would have liked to go to if it did not clash with another talk I wanted to catch. I remember chatting a bit about C++ templates and C# / .NET generics.

### Tuesday

The second and final day of the conference got off to a start at 09:00 with a keynote given by Ivan Goddard from Mill Computing [14] on using the Clang and LLVM toolchain for their 'truly alien' Mill CPU architecture and the problems they have encountered. Ivan started by asking how many people had heard of the Mill CPU and on finding most people had not, spent the first part of the talk on a quick tour of the Mill CPU architecture (for those interested check out the documentation section on

**RALPH MCARDELL**

Ralph McArdell has been programming for more than 30 years with around 20 spent as a freelance developer predominantly in C++. He does not ever want or expect to stop learning or improving his skills.

the Mill Computing web site [15]). Next Ivan went through how their compiler team were using LLVM and Clang and the problems they had encountered – including LLVM not liking the Mill's large, very regular instruction set; that LLVM and Clang lose 'pointerhood' as pointers tend to devolve to integers which is not good for the Mill as it uses a specific 64-bit pointer type; that LLVM cannot cope with the high level of function call support the Mill provides; and refactoring LLVM code on the trunk breaks other targets. The Mill macro assembler is interesting in that assembler instructions are C++ functions and C++ is the assembler macro language. First you compile the assembler C++ program, and then run the resultant executable which generates the assembler code. To end, Ivan offered some code they use to automatically produce specific Mill family member instruction sets from specifications as an example of an alternative to the LLVM TableGen tool, which, it appears, is in need of having something done about it – but no one knows what. Finally, Ivan appealed to the LLVM and Clang community for help fixing the problems Mill Computing had experienced.

**we were reminded that certain sorts of compiler optimisations can produce incorrect code in concurrent contexts**

The first of Tuesday's sessions I attended was given by Liam Fitzpatrick and Marco Roodzant about LLVM-TURBO [16] which turned out to be a commercial product aimed at those needing to create code generators for their embedded processors and who do not wish to get their hands dirty with Clang and LLVM directly. The selling point is that LLVM-TURBO requires less time and people with the example of using vanilla LLVM requiring 10 people over 2 years while using LLVM-TURBO required 3 people over 4 months. LLVM-TURBO uses what appears to be their own CoSy compiler development system and bridges between LLVM and CoSy formats.

To take us up to lunch were a set of short 5 minute lightning talks – a familiar concept to those who have attended an ACCU conference in recent years. Arnaud de Grandmaison started the proceedings by informing us that using vectorisation to speed up computations such as colour space conversion and matrix multiplication can give a two times speed increase. Dmitry Borisenkov reported on an LLVM based ahead of time Javascript compiler that although in an alpha state can be up to two times faster than the Google V8 engine. Tilmann Scheller gave us some tips on building Clang and LLVM as quickly as possible and also spoke about the new 2.0 version of the OpenCL SPIR intermediate representation [17] saying that unlike the original it is no longer a subset of LLVM IR but can easily be mapped to LLVM using a small decoder. Next Frej Drejhammar and Lars Rasmusson presented their proposal for LLVM extensions allowing the generation of patch points while Jiangning Liu, Pablo Barrio and Kevin Qin explained their patch that uses heuristics to improve the LLVM inliner's performance. Alberto Magni introduced Symengine that analyses CPU↔GPU transactions in order to optimise data transfers between CPU and GPU. Edward Jones explained how a patch to DejaGnu [18], used for regression testing of GCC, allows it to be used to regression test Clang which is especially useful for embedded systems as they can use the remote execution feature. Hao Liu, James Molloy and Jiangning Liu presented a method of vectorising interleaved memory accesses. Pablo Barrio. Chandler Carruth and James Molloy meanwhile returned to inlining with a description of their attempts to allow inlining of recursive functions, and how the final attempt using a stack to remove recursion in fact ended up producing slower code for a pathological Fibonacci series test case. Russell Gallop presented a method of verifying that code generation is unaffected by compiler options such as `-g` (generate debug information) and `-S` (preprocess and compile but do not assemble or link) by compiling with and without the option(s) of interest and comparing the generated output — doing so can help locate subtle bugs across the compiler code base. Kevin Funk explained how moving the KDevelop IDE's C and C++ editor language support to libclang provided full C and C++ language parsing and they got ObjectiveC parsing for free! Finally, Alexander Richardson and David Chisnall introduced a Clang extension that optimised memory allocation for objects using the C++ PImpl idiom [19] by combining allocation in a similar way to `std::make_shared`. If I understood correctly the extension, through the use of a custom attribute, would also create the wrapper class that wraps the implementation class instance pointer.

After lunch I decided to go to Deepak Panickal and Ewan Crawford's session on why one might want to use LLDB [20], the LLVM debugger. It is designed with a clean and maintainable plugin architecture, works on all major platforms — with the caveat that there is more work to do on MS Windows support, maintains up to date language support by using libclang, has both C++ and Python APIs to add LLDB support to applications and automate repetitive tasks, has an internal Python interpreter, allowing scripts to be run from breakpoints, has a GDB compatible machine interface and that it should be easy to switch to from GDB. Got that? Good.

Following the LLDB session I went straight into Daniel Krupp, Gyorgy Orban, Gabor Horvath and Bence Babati's talk on their industrial experiences of using the Clang static analysis tool. It seems that while Clang and its associated tools can form an impressive checker framework they do have their usability problems which makes their use quite fiddly. To mitigate these problems the authors' team at Ericsson created a project build workflow together with viewer tools to smooth over the rough edges. There are plans to open source the tools and submit the code to the community, providing their employer has no objections.

After the mid afternoon refreshment break I went to JF Bastien's talk on using C++ on the web without getting users pwned. It seems JF works for Google and the talk concerned the Chrome Native Client (NaCl) [21] and Portable Native Client (PNaCl) and the security measures they use. As I had never seen (P)NaCl in action the most impressive thing about this talk were the demonstrations of things like bash shells running in Chrome along with applications like Emacs and Vim. It seems that the native client built in to recent Chrome browsers, and I think Chromebooks, provides native code — currently written in C and/or C++ — to execute in a sandboxed Linux like OS environment. PNaCl executables use the LLVM toolchain to compile down to LLVM IR that is then compiled to native code when downloaded as part of a web page load. Of course you have to be paranoid about running native code so other than the sandboxed pseudo-OS environment they use other techniques such as random instruction and register selection when compiling and using fuzzing to help check for bugs.

Next I went straight to my final session of the conference given by Siva Chandra Reddy on using LLDB for debugging. Siva started by giving a report on the LLDB project status. Encouragingly the project now has 11 developers, has support for Linux and Android, with Windows support under active development. Remote debugging support has recently been checked in, is documented and now uses a remote debug server. X86 and X86-64 support is available now, with ARM and ARM64 support under development. On Windows Win32 support is mostly complete with Win64 coming along. Next some details on using remote debugging were given first when both debugger and debuggee are the same platform and then the more complex case where they are different as is common in the embedded world. Finally Siva covered debugging and testing the debugger, mentioning that LLDB has very good logging facilities as well as special command line arguments and environment variables. As for testing, because LLDB is very interactive and platform dependent they use a Python test framework in which each Python test case has an accompanying C/C++/ObjectiveC file that is used as the thing that is to be debugged.

**using a stack to remove recursion in fact ended up producing slower code**

There was only the conference close session left and while waiting for it to start in the lecture theater it was to be held in I caught the end of the previous session on a Fortran front end for LLVM and my ears picked up the name 'Flang'. ■

# Code Club

Steve Folly shares his experiences with volunteering and teaching children coding.

There has recently been a proliferation of organisations set up to get children and young people interested in 'coding'. And not just in the UK – throughout the world as well, including Coder Dojo [1], Hour of Code [2], and many others.

There are most likely good reasons for this. In the UK, the number of students taking computer science subject in universities fell by about 29% between 2003 and 2012 [3]. This is not surprising when you understand that teaching IT in schools used to be all about learning how to use applications like Word and Excel; not how to write software.

I'm happy in my job as a software developer, I'm getting paid, why should these figures concern me? Well, I am concerned and I do care. Who's going to be there in the future to continue our good work?

**I believe passionately it's up to us to get the next generation inspired by technology**

Over the past few years I found I've been helping and mentoring colleagues, and I always had this feeling that I wanted to do more. I have in the past volunteered with a local community centre; giving my time to help with IT related topics. It was very fulfilling and I was really pleased that I could help others with my knowledge.

If any readers were at the ACCU Conference in 2014 they will remember Bill Liao's keynote talk about Coder Dojo [1]. I was there and it really hit home with me: inspiring children – that's it.

So, there's the idea. How do I implement it? The thought of just approaching a school and asking "Hey, I'd like to set up a computer club for the children, how about it?" was a bit daunting. I already had in my mind the kinds of responses I might get, and that put me off a bit!

I was in conversation with a friend one day and he mentioned an organisation called Code Club Pro [4]. His brother is a teacher and Code Club Pro have an amazing group of volunteers to teach computing skills to teachers. Code Club Pro is a part of the Code Club family; there is also Code Club [5] which is aimed at teaching coding skills to 9 to 11 year old children.

The reason this has all kicked off in the UK is that the National Curriculum was changed a couple of years ago to put more emphasis on the fundamentals of coding skills (including understanding algorithms, decomposition, debugging).

This seemed to be the perfect opportunity right there: a UK-wide organisation where volunteers can register an interest to run a club, and schools and organisations can also register an interest to want to run a club. Having the backing of an organisation like this will make things much easier especially as you won't be contacting the school out of the blue. Code Club's website is really easy to use. It took me just 10 minutes to sign up and they have a checklist of the important things to do to help you get your club up and running.

If you can't find an organisation at a suitable location for you, they also have a full list of all schools so you are able to contact a school even if they're not actively looking to start a club. It's up to you then to convince the school they need to start a club!

There are a few important – and legal – things to do before you are able to start your club. The most important one is to apply for a DBS (Disclosure and Barring Service) certificate. This is what used to be called the CRB (Criminal Records Bureau). Basically, is it safe for you to work with children? Each organisation you work with may require you to apply for a DBS certificate. As I found out, you can't just forward a copy of your certificate to the next organisation. The non-optimal route is to apply for a DBS certificate for every organisation you are working with. I have two now, and then I found out about the Update Service. I highly recommend you opt-in to this when you get your first certificate. This is the official route that allows other organisations to officially confirm that you have a valid DBS certificate.

You will also need public liability insurance. If you're setting up a club on your own this can be rather expensive.

The school or organisation may be happy to help you get your DBS certificate, but some may not want to get involved with insurance. There

**STEVE FOLLY**

Steve has been a software developer since the early 1990s, working on defence projects such as radars, and on ticketing systems for public transport. Steve was a Sinclair child, starting out on a ZX81. He can be contacted at steve@spfweb.co.uk

## EuroLLVM Conference 2015 (continued)

### References

[1] LLVM, http://llvm.org/
[2] Clang, http://clang.llvm.org/
[3] Embecosm, http://www.embecosm.com/
[4] Myre Laboritories, http://myrelabs.com/Myre
[5] Open Source Hardware User Group, http://oshug.org/
[6] Parallella, http://www.parallella.org/
[7] CppMem, http://svr-pes20-cppmem.cl.cam.ac.uk/cppmem/help.html
[8] DWARF standard, http://www.dwarfstd.org/
[9] T-EMU 2, http://t-emu.terma.com/
[10] TableGen documentation, http://llvm.org/docs/TableGen/
[11] Templight, http://plc.inf.elte.hu/templight/
[12] Metashell, https://github.com/sabel83/metashell
[13] Mikael Persson's Templight re-write https://github.com/mikael-s-persson/templight
[14] Mill Computing Inc., http://millcomputing.com/
[15] Mill Computing documentation, http://millcomputing.com/docs/
[16] LLVM-TURBO, http://www.ace.nl/LLVM-TURBO
[17] SPIR, https://www.khronos.org/spir
[18] DejaGnu project, http://www.gnu.org/software/dejagnu/
[19] Pointer To Implementation (pImpl) http://en.wikibooks.org/wiki/C%2B%2B_Programming/Idioms
[20] LLDB project, http://lldb.llvm.org/
[21] Google Chrome Native Client https://developer.chrome.com/native-client

is an alternative – and Code Club recommend that you arrange your DBS certificate and insurance via STEMNet.

STEMNet is a UK wide independent charity which receives funding from the UK government, the Scottish Government and the Gatsby Charitable Foundation and relies on thousands of volunteers to inspire young people in STEM subjects: Science, Technology, Engineering and Maths. And thanks to a change in the national curriculum a few years ago, there is now more than ever a high demand for volunteers for computing.

So if you sign up to become a STEMNet Ambassador, they will organise a DBS check for you and cover you for public liability insurance as well. You will need to attend an induction session where you will learn about how to work with children – e.g. what to do if they tell you something they want to be kept private, etc.

As well as this very useful induction session, Code Club also have a number of training videos to help you get settled in to your club. If you've never worked with children before they are very useful and cover topics such as keeping children safe, helping children learn and volunteering in schools.

One of the conditions of STEMNet's insurance is that a teacher must be present when you run your club. This is certainly a good thing, because in my experience they will be there for crowd control!

I was lucky enough to discover that Dormansland Primary School were looking for a volunteer to help run a club. The school is just a couple of miles from the office where I work. Communication with school was simple, a few email exchanges, initiated via Code Club, and an introductory meeting at the school was arranged. The teacher was really enthusiastic about the club and was pleased to have found a volunteer. The timing was just spot on – the meeting was at the end of November, and the school was planning to start the Code Club in January for the spring term.

I was really looking forward to it, albeit with a little apprehension. I had a naïve preconception about how the club would be run – calmly and orderly. It wasn't like that at all, and in hindsight, that's not a bad thing. That's not to say it was wildly chaotic either! The thing that pleased me most was to see that the children were helping each other when they got stuck or had questions.

Code Club have a wide range of pre-prepared projects that you can use in your club – teaching coding in Scratch, Python and learning about HTML and CSS. The projects are also on Github and they are open to contributions.

Scratch [6] is a fantastic environment to teach young children about coding. It's very visual and colourful, and code can be created by simple dragging and dropping of code blocks that snap to together.

I chose to teach the children Scratch as I found out from the initial discussions with the school that the children were starting to use Scratch in their lessons, and their computers have Scratch already installed.

Which brings me on to a few hints and tips – when you're planning your sessions, it would help to find out exactly what facilities the school or organisation have. In particular, which versions of software do they have? The Scratch lessons on the Code Club web site are written for Scratch version 2.0. My club started using version 1.4. It's not a show-stopper, since I managed to adapt the lessons for 1.4. Also, I found out that some of the computers didn't have audio configured properly – that was a bit unfortunate to discover half way through a lesson writing a project called 'Rock Band'! We switched to Scratch 2.0 online half way through the term.

Are the computer facilities networked? Do they have an internet connection? If you are using Scratch 2.0 you might want to consider whether the children could use the online version and create online accounts so they can show off their work outside of school. Obviously, you will need to discuss this with the school first, permission would have to be sought. In my case, some of the children already had an online account so when we switched to Scratch 2.0 they were able to save their stuff online.

Does the school have a projector? My school does have a projector, so I took the approach to work through the lessons on a projector so we could all work at the same pace. Perhaps one disadvantage is that you'll be working at the slowest pace, so some of the more able children may get bored. An alternative is to print exercise packs for each child to work through at their own pace. Each method has its advantages and disadvantages; there is no right way so you'll have to see what works for you. I think I'll try a mixed approach next time.

I overestimated how much work we could do in a one hour lesson. We did tend to rush a bit towards the end of each session. I'm getting better at that now, but I'm still very pleased with how much we managed to get through.

On the last day of the club just before Easter I gave the children free rein to create whatever project they wanted to do and was pleased to see them come up with creations that I would never have expected. It's all about igniting that spark of creativity and imagination in the children.

My second club is now running over the summer term. Initially, there wasn't enough interest in the club to warrant running it. That was disappointing considering the good feedback I had from the first club. However, I was assured this was typical of all after-school clubs during the summer term. But after I suggested coming to the school to give a 5-minute pitch during an assembly about what Code Club is – including making a piano out of fruit (using the MaKey MaKey kit) – the looks of confusion on the children's faces was just what I was after – and, disappointingly, not being able to launch rockets from the school playground – I was informed the club was now oversubscribed so unfortunately a few children had to be turned down. It's slightly different to the spring term club – we have a wider age group this time, so I had to deal with a wider range of abilities. Still, I managed to keep them all engaged and interested and enthusiastic about coding which is the most important thing.

I haven't mentioned yet that, being an after-school club means that it runs between 3.15pm and 4.15pm. It certainly helps to have an understanding employer who is willing to let you have the time away from the office. This was most definitely not a problem in my case; Dave, the owner of the company I work at is also passionate about 'giving back to the community' so he didn't need much convincing.

I briefly mentioned earlier about Code Club Pro. If volunteering with children is not for you but still want to help in some way, consider Code Club Pro. With the change in the national curriculum, there is a big learning curve for teachers as well. Code Club Pro and other similar organisations, such as Computing At School [7], part of the BCS, exist to help teachers get up to speed with the technology they are expected to teach to children.

I believe passionately it's up to us to get the next generation inspired by technology. For so long, education in the UK has neglected the importance of technology.

I hope you saw my Lightning Talk at the ACCU 2015 conference and that it's motivated you to do something. I'm not sure which was more nerve-racking for me – giving a 5-minute talk to my peers about Code Club, or actually running a club for children. The lightning talk went well, and thank you to everyone who gave me feedback – especially Frances Buontempo who suggested I write this article.

And it's not just the children that are gaining out of this – it's been a great learning experience for me as well.

It's up to us to inspire the next generation for our profession. Do something about it! ■

## References

[1] http://www.infoq.com/presentations/coderdojo
[2] https://hourofcode.com/
[3] http://www.universitiesuk.ac.uk/highereducation/Documents/2013/PatternsAndTrendsinUKHigherEducation2013.pdf
[4] http://codeclubpro.org/
[5] http://codeclub.org.uk/
[6] http://scratch.mit.edu/
[7] http://www.computingatschool.org.uk/

# Split and Merge Revisited

## Vassili Kaplan makes improvements to the Expression Parser.

Silas S. Brown (ssb22@cam.ac.uk) spotted a defect in the Merge part of the Split and Merge algorithm published in *CVu* [1] – you can see Silas's letter on page 21. We fix this defect here and also provide a slightly different C++ implementation of the Split and Merge algorithm, using the 'Virtual Constructor' idiom [2].

### Right associativity defect

As Silas Brown pointed out, "the Split and Merge algorithm, as described in the article, does not consistently treat operators as right-associative either. If the operator is preceded by another of the same or lower precedence level, then it will be left-associative: 3-12-1 would get -10. But if the operator is preceded by one of a higher precedence level, it will become right-associative with the operators before that: 3-2*6-1 would get -8."

To fix this, we need to break out of the merge loop and go back to where we were after any successful merge.

### Split and Merge revisited

The split part remains the same: the result of the first step of our algorithm is a list of structures. Each structure consists of two elements – a real number and an action, e.g.

Splitting ("3 - 2 * 6 - 1") → [3, '-'], [2, '*'], [6, '-'], [1, ')']

In the merge step the actual calculation is done by merging the elements of the list of structures created in the first step. We attempt to merge each structure in the list with the next, taking into account the priority of actions.

Merging can be done if and only if the priority of the action in the first structure is not lower than the priority of the second structure. If this is the case merging consists in applying the action of the first structure to the numbers of both structures, thus creating a new structure, e.g. merging [5, '-'] and [4, '+'] will produce [5 - 4, '+'] = [1, '+'].

What if the second structure priority is higher than the first one?

Then we merge the next (second) structure with the structure next to it, and so on, recursively. We break out of the merge loop and go back where we were after any successful merge.

Continuing with our example of "3 - 2 * 6 - 1":

Merging ([3, '-'], [2, '*'], [6, '-'], [1, ')']) →

Merging ([3, '-'], Merging ([2, '*'], [6, '-']), [1, ')']) →

Merging ([3, '-'], [12, '-'], [1, ')']) →

Merging ([-9, '-'], [1, ')']) → [-9 - 1, ')'] = [-10, ')'].

Therefore, 3 - 2 * 6 - 1 = -10, which is now correct.

Listing 1 contains the implementation of the fixed Merge part in C++ (the parts different from the earlier version [1] are in bold).

### Implementation using the 'Virtual Constructor' idiom

The implementation of the Split and Merge algorithm in [1] was using pointers to functions. As James Coplien points out, a better, more object-oriented way, would be using the 'Virtual Constructor' idiom [2]:

> "The virtual constructor idiom is used when the type of an object needs to be determined from the context in which the object is constructed."

In our case these objects are actual functions to be called (e.g. sine, cosine, logarithm, etc.). They are determined only at run time. The advantage of this idiom is that a new function can be added easily to the framework without modifying the main algorithm implementation code.

Listing 2 is how the base class of such functions would look.

Listing 1

```
double EZParser::merge(Cell& current,
  size_t& index, vector<Cell>& listToMerge,
  bool mergeOneOnly)
{
  if (index >= listToMerge.size())
  {
    return current.value;
  }

  while (index < listToMerge.size())
  {
    Cell& next = listToMerge.at(index++);
    while (!canMergeCells(current, next))
    { // If we cannot merge cells yet, go to the
      // next cell and merge next cells first.
      // E.g. if we have 1+2*3, we first merge
      // next cells, i.e. 2*3, getting 6, and
      // then we can merge 1+6.
      merge(next, index, listToMerge, true
        /* mergeOneOnly */);
    }

    mergeCells(current, next);
    if (mergeOneOnly) {
      return current.value;
    }
  }

  return current.value;
}
```

Listing 2

```
class EZParserFunction
{
public:
  static double calculate(const string& data,
      size_t& from, const string& item, char ch)
{
    EZParserFunction func(data, from, item, ch);
    return func.getValue(data, from);
  }
  static void addFunction(const string& name,
     EZParserFunction* function) {
    m_functions_[name] = function;
  }

protected:
  EZParserFunction() : impl_(0) {}
  virtual ~EZParserFunction() {}
  virtual double getValue(const string& data,
     size_t& from) {
    return impl_->getValue(data, from);
  }
```

**VASSILI KAPLAN**

Vassili Kaplan has been a Software Developer for almost 15 years, working in different countries and different languages (including C++, C#, and Python). He currently resides in Switzerland and can be contacted at vassilik@gmail.com.

Listing 2 (cont'd)

```
  double loadArg(const string& data,
     size_t& from)
     {
       return EZParser::loadAndCalculate(data,
         from, ')');
     }

private:
  EZParserFunction(const string& data,
    size_t& from, const string& item, char ch);

  EZParserFunction* impl_;
  static map<string,
    EZParserFunction*> m_functions_;
};
```

Here is what a derived class for a sine function would look like:

```
class SinFunction : public EZParserFunction
{
  public:
    double getValue(const string& data,
    size_t& from)
  {
    return ::sin(loadArg(data, from));
  }
}
```

The functions derived from the **EZParserFunction** class are used in the first, splitting process. The split process itself will remain the same, just the call to process the last extracted token will be now

```
double value = EZParserFunction::calculate(data,
   from, item, ch);
```

instead of

```
double value = m_allFunctions.getValue(data,
   from, item, ch);
```

Listing 3 is the (virtual) constructor of the base class.

As we can see the two functions, **StrtodFunction** and **IdentityFunction**, have a special meaning and must be implemented (see Listing 4).

Listing 3

```
EZParserFunction::EZParserFunction
   (const string& data, size_t& from,
    const string& item, char ch)
{
  if (item.empty() && ch == '(')
  {
    impl_ = identityFunction;
    return;
  }
  map<string, EZParserFunction*>::
  const_iterator it
  = m_functions_.find(item);

  if (it == m_functions_.end())
  {
    strtodFunction->setItem(item);
    impl_ = strtodFunction;
    return;
  }

  impl_ = it->second;
}
```

Listing 4

```
class StrtodFunction : public EZParserFunction
{
public:
  double getValue(const string& data,
    size_t& from)
  {
    char* x;
    double num = ::strtod(item_.c_str(), &x);
    if (::strlen(x) > 0)
    {
      throw CalculationException("Could not parse
         token [" + item_ + "]");
    }
    return num;
  }
  void setItem(const string& item) {
    item_ = item; }

private:
  string item_;
};

class IdentityFunction : public EZParserFunction
{
public:
  double getValue(const string& data,
  size_t& from)
  {
    return loadArg(data, from);
  }
};

static StrtodFunction*
  strtodFunction = new StrtodFunction();

static IdentityFunction*
  identityFunction = new IdentityFunction();
```

The rest of the functions do not have to be implemented for the algorithm to work and can be added ad hoc (e.g. **log**, **exp**, **pi**, etc.)

Listing 5 is how the user can add functions that she implemented somewhere in her program initialization code. ■

## References


[1] Vassili Kaplan, 'Split and Merge – another Algorithm for Parsing Mathematical Expressions', *CVu*, Volume 27, Issue 2, May 2015.

[2] James Coplien, *Advanced C++ Programming Styles and Idioms* (p. 140), Addison-Wesley, 1992.


Listing 5

```
EZParserFunction::addFunction("exp",
  new ExpFunction());
EZParserFunction::addFunction("log",
  new LogFunction());
EZParserFunction::addFunction("sin",
  new SinFunction());
EZParserFunction::addFunction("cos",
  new CosFunction());
EZParserFunction::addFunction("fabs",
  new FabsFunction());
EZParserFunction::addFunction("pi",
  new ExpFunction());
EZParserFunction::addFunction("sqrt",
  new SqrtFunction());
```

# Anthony Williams: An Interview

## Emyr Williams continues the series of interviews with people from the world of programming.

Anthony Williams is the author of *C++ Concurrency in Action*, and Director of Just Software Solutions Ltd. He has extensive experience developing a wide variety of applications for a wide range of systems, including embedded systems, Windows applications and web-based applications, which he draws on when advising clients or developing bespoke software for them. A strong believer in the benefits of TDD and agile methodologies, Anthony is always striving to find ways to minimize the code that cannot be tested.

**How did you get started in computer programming? Was it a sudden interest in computing? Or was it a gradual process?**

I started when I was about 7. We had a ZX81 at home, and BBC Micros at school. I used to program them in BASIC. I became hooked rather quickly. Often I would write programs on pieces of paper for typing in later because I didn't have access to the computers as much as I'd like. Later on we had an Amstrad CPC6128 at home, and I used to program that in BASIC and assembly language (hand-assembled on paper). What really got me interested was the idea that I could make the computer do anything I wanted, if only I took the time to work out how. Seeing software others had written I would want to know how they did it.

**What was the first program you ever wrote? And what language did you write it in?**

The very first program I ever wrote was something along the lines of

```
10 PRINT "Hello"
```

in BASIC. It's amazing the feeling that such a simple thing can give you: I can type instructions and the computer will do what I say! I then wrote other simple BASIC programs, such as the classic animal-vegetable-mineral game.

**What would you say is the best piece of advice you've ever been given as a programmer?**

"Write copious unit tests". It's simple advice, but I didn't really get it for a long time. Of course you have to test your code, but the importance of unit tests didn't really sink in until I started reading about TDD and refactoring. With a decent set of unit tests you can be sure that your code is working now, and when you change something you can immediately see if you unintentionally changed the behaviour of something else. Combined with source control so you can easily get back to previous versions that's incredibly powerful.

**If you were to start your career again now, what would you do differently? Or if you could go back in time and meet yourself when you were starting out as a programmer, what would you tell yourself to focus on?**

I could probably have done with a book on algorithms and data structures when I was starting out. Though it's fun working things out for yourself, it can get in the way of achieving the end goal.

**What was the biggest "ah ha" moment or surprise you've experienced when chasing down a bug?**

I don't recall any particularly momentous "ah ha" moments or surprises when chasing bugs. Often enough it's more of a "D'Oh!" moment when you realise that you used the wrong variable in an expression.

**A lot is said about elegant code these days. What would you say is the most elegant code you've seen? And how do you define what elegant code is?**

Elegant code is clear, simple and concise. You can read it and easily understand what it does without needing any comments, but there's nothing extraneous. Sadly, it's often really hard to write elegant code! To a large extent, it's the use of suitable abstractions that enables elegant code. Code written using the `jss::actor` class from Just::Thread Pro can be remarkably elegant, as can code written using just the STL, especially when combined with the C++11 range for loops, and lambda functions.

**With technology moving so fast these days, where do you think the next big shift in computer programming is going to be?**

I think Functional Programming is going to become increasingly mainstream, especially as parallelism grows, since it greatly simplifies parallel programming if you have side-effect-free functions. However, I don't think it's going to supplant imperative programming; it's just going to be another tool that good programmers reach for when appropriate.

**One of the things you were involved with was writing the threading library that was included in Boost, which has now become a part of the C++ standard from C++ 11 onwards. How did that come about?**

Bill Kempf wrote the original version of Boost.Thread. In 2005, there was a drive to get the whole of Boost covered under a single license – the new Boost Software License. Unfortunately, at the time Bill could not be contacted to change the license, so a couple of us began a rewrite project, and rebuilt the whole library from the ground up. As the work on the new C++ standard was underway, I then got involved with the proposals for the thread library portion of the new standard, and the 'new' Boost implementation and the draft standard then evolved together. I've since handed over maintenance of Boost.Thread to Vicente Botet, as my work on concurrency libraries has been focused on my own Just::Thread and Just::Thread Pro libraries.

**What advice would you give a programmer who'd like to get involved in an open source project or contribute to something like Boost?**

Become an active contributor on the project mailing list, so people know who you are when you submit patches. Ask the active developers which areas they need help with, or just submit patches that fix bugs in the bug tracker. Most developers will be glad of the help.

Contributing a whole library to Boost is a lot of work. Getting it reviewed and accepted can take years, though they are trying to address that, and once your library has been accepted then you need to handle bug reports and fixes, often for platforms you have no access to.

**Do you mentor other developers? Or did you ever have a mentor when you started programming?**

I am not currently mentoring anyone, though I have done in the past, and I enjoyed the responsibility of helping someone else expand their skills. I didn't have a 'mentor' when I started, though you could probably say that my Dad took on that role.

**EMYR WILLIAMS**

Emyr Williams is a C++ developer who is on a mission to become a better programmer. His blog can be found at www.becomingbetter.co.uk

# ACCU London Review

## One of the attendees at a recent meet shares their experience.

Having been quiet for a while, due to people moving away, we have finally managed to get a few regular meetings going again. In May we just had a social, and in June we had some mini-talks. Instead of insisting on a time capped 5 minute lightning slot, people were allowed to talk for a little longer if they wanted. The talks varied between about 5 minutes and nearer half an hour. This format allows people who may not be brave another to give a presentation for an hour to do something slightly less scary. One thing the ACCU has consistently managed is allowing people to step outside their comfort zone and thereby getting better, be that at coding, talking, writing or whatever.

We had five speakers:

- Jaimen lathia – 'Computer Science vs Software Engineering'

  Jaimen said this was the first time he'd spoken outside of work, and took a look at some of the terms we use to talk about ourselves.
- Frances Buontempo – 'ABC your way out of a paper bag'

  Frances swarmed her way out of a paper bag, using an abstract bee colony.
- Guilherme Candido Hartmann – 'New developments on Neural Networks and Deep Learning'

  Guildherme shared some really powerful neural network techniques for computer vision.
- Schalk Cronjé – 'Asciidoctor'

  Schalk showed us how easy Asciidoctor is to use and the various formats it can automatically generate.
- Patrick M Martin – 'schadenfreude security "adjustment"'

  Patrick showed us how to do evil things with Windows dlls.

We then retired to a local tavern and caused a porter draught after a little effort.

Most of speakers have kindly shared links to techniques and tools they mentioned on the meetup page: http://www.meetup.com/ACCULondon/events/222888479/


# 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

## Anthony Williams: An Interview (continued)

What inspired you to start your own business? And how did you find the experience? And any advice for anyone thinking of going it alone?

I started my own business because I wanted to move to Cornwall, and I couldn't find a decent programming job in Cornwall at the time. From a practical point of view it's straightforward – fill in some forms, pay a bit of money. The scary part is ensuring that you have enough jobs coming in or product sales to pay your bills. Thankfully, I had existing contacts who were more than happy to subcontract development to my company, whilst I made new contacts and built up the sales of Just::Thread.

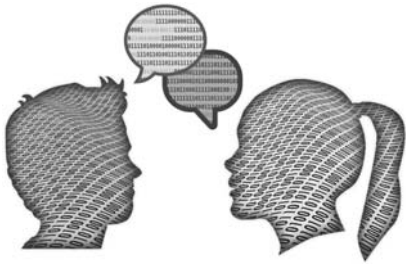
For anyone thinking of setting up their own business, my advice is to sound out existing contacts for work before you make the leap. It's also worth reading the advice of people like Andy Brice (http://successfulsoftware.net) and Patrick McKenzie (http://www.kalzumeus.com/), especially if you want to sell a product rather than take on development work for other companies.

Finally, do you have any advice for any kids or adults who are looking to start out as a programmer?

There's a lot of options for someone starting out today, with many books and tutorials available, as well as sites like http://www.codecademy.com/ which provides interactive lessons, and http://www.codewars.com/ which provides exercises of various levels to help you practice what you've learnt. When I was learning, I greatly appreciated the immediacy of BASIC: there was no compile step, and you could just type BASIC commands at the prompt. To get that today you need something with a REPL, like Python, or node.js for Javascript. My son got on well with 'Invent your own computer games with Python'. Once you've learnt enough from the tutorials, pick a simple project and work through it.

# Code Critique Competition 94

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. (We will remove email addresses!)

## Last issue's code

> I'm trying to write a simple program to demonstrate the Mandelbrot set and I'm hoping to get example output like this Figure 1, but I just get 6 lines of stars and the rest blank. Can you help me get this program working?

Can you find out what is wrong and help this programmer to fix (and improve) their simple test program? The code is in Listing 1.

Listing 1

```
#include <array>
#include <complex>
#include <iostream>
using row = std::array<bool, 60>;
using matrix = std::array<row, 40>;
std::ostream& operator<<(std::ostream &os,
  row const &rhs)
{
  for (auto const &v : rhs)
    os << "* "[v];
  return os;
}
std::ostream& operator<<(std::ostream &os,
  matrix const &rhs)
{
  for (auto const &v : rhs)
    os << v << '\n';
  return os;
}

class Mandelbrot
{
  matrix data = {};
  std::complex<double> origin = (-0.5);
  double scale = 20.0;
public:
  matrix const & operator()()
  {
    for (int y(0); y != data.size(); ++y)
    {
      for (int x(0); x != data[y].size(); ++x)
      {
        std::complex<double>
          c = (xcoord(x), ycoord(y)), z = c;
        int k = 0;
        for (; k < 200; ++k)
        {
          z = z*z + c;
          if (abs(z) >= 2)
          {
            data[y][x] = true;
            break;
          }
```

Listing 1 (cont'd)

```
        }
      }
    }
    return data;
  }
private:
  double xcoord(int xpos)
  {
     return origin.real() + (xpos -
       data[0].size()/2) / scale;
  }
  double ycoord(int ypos)
  {
     return origin.imag() + (data.size()/2 -
       ypos) / scale;
  }
};
int main()
{
  Mandelbrot set;
  std::cout << set() << std::endl;
}
```

## Critiques

### Paul Floyd <paulf@free.fr>

It took some debugging to get to the causes of the errors.

First impressions. I didn't like the uncommented use of indexing a string literal in the row `operator<<`. I understood it, but not all developers would. As it's a relatively rare idiom, it needs an explanatory comment. Just to make sure that there was no problem with the output, I changed the literal to `*.` so that the 'true' values printed something more visible. And indeed there was no problem there. I had a quick check that the unqualified call to `abs` was indeed calling the complex version rather than the `int` one. Lastly I didn't like the gratuitous use of `operator()`, defined in the class body to boot. I'd prefer a named function like `calculate`. If `calculate` is always called after construction, then I'd call it from the constructor instead.

Next, compiling the code. I tried with

- Solaris Studio 12.4 – no warnings
- clang++ 3.2 – complained about sign comparison in the `x` and `y` for loops
- g++ 4.8.2 – same `x` and `y` for loop sign comparison, and also about missing field initializers for `Mandelbrot::data`.

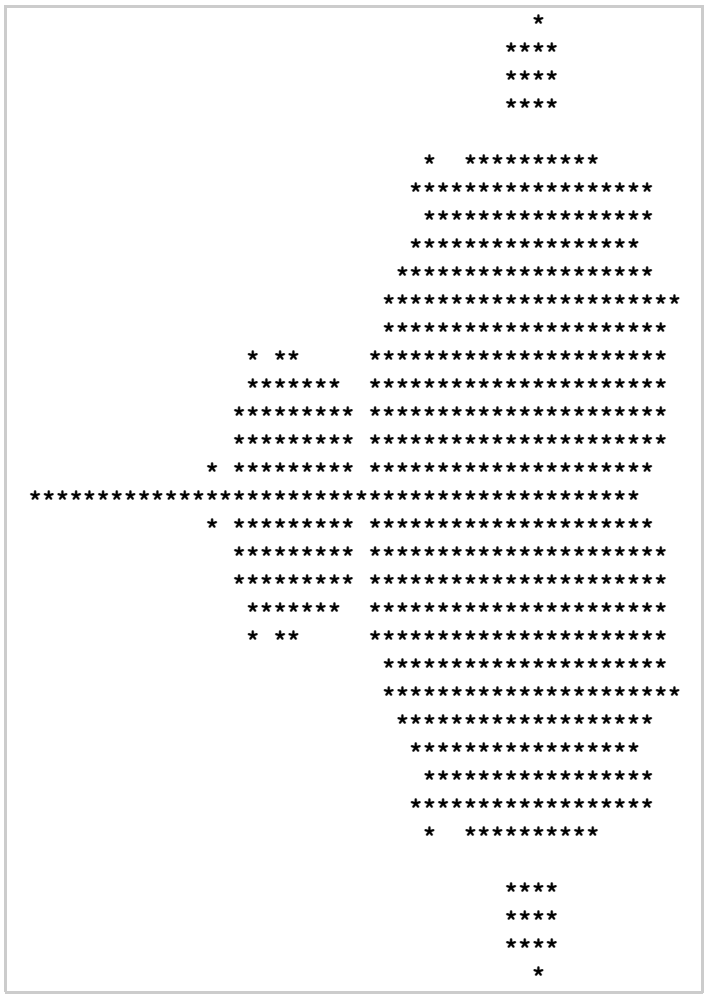
So I started adding some debug traces. I noticed fairly quickly that the values of complex `c` and `z` always had zero imaginary parts. I looked at initializer expression for `c`

```
std::complex<double> c = (xcoord(x), ycoord(y));
```

**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

Figure 1



and saw the first light. That doesn't initialize a `std::complex` with two fields. It's an operator, expression in parentheses. The result of the expression is a double, and there's a `std::complex` constructor overload that takes just a double, so it can be converted to `std::complex`.

In C++11 uniform initialization style, this should be

```
std::complex<double> c = {xcoord(x), ycoord(y)};
```

The next thing that I noticed were some excessively high values coming from `xcoord(x)`. These looked suspiciously like `INT_MAX`. I added traces for all of the values used in `xcoord()`, and they all looked reasonable.

Then I looked a bit more closely at the expression:

```
return origin.real() + (xpos -
  data[0].size()/2) / scale;
```

In terms of types, that is

```
double + (int - size_t/int) / double
```

If it were the case that the `size_t` in the middle were converted to `int`, then all would be well. But it isn't. If I look at the typeid of `data[0].size()/2` (and even demangle it, then it is `unsigned int` on a 32bit build and `unsigned long` on a 64bit build. The same is true for `xpos - data[0].size()/2`. What this leads to is that it isn't the `size_t` that is converted to `int`, it is the `int` that is converted to `size_t`. `xpos` ranges from 0 to 59, and `data[0].size()/2` is 30. So when `xpos` is less than 30, the result is negative. Or it would be if the expression type were not `size_t`, which has no negative values. Instead it wraps around to a very high positive number.

The fix for this is quite simple. Instead of using literal `2` in the expression, use `2.0`, thus

```
return origin.real() + (xpos -
  data[0].size()/2.0) / scale;
```

the types this time are

```
double + (int - size_t/double) / double
```

This time, in the subexpression `size_t/double`, the `size_t` gets converted to `double`, which again happens for `(int - double)`. Since `double` has no problem representing negative numbers, this works as intended. The same change needs to be made in `ycoord`.

This is one of my pet peeves, people using literals of the wrong type in expressions. I've even seen stuff like `(double)2)` rather than the more obvious `2.0`.

Lastly, how could these have been avoided? Well, both mistakes were well formed C++, so the compiler was of no help. The huge numbers produced by `xcoord` could have been detected fairly easily with a bit of analysis of what the expected upper and lower bounds are, and a suitable `assert`.

Finally, the initialization expression for `c`. As `0.0` is perfectly valid for `c.imag()`, an `assert` won't help.

A unit test could be used, but that would require considerable changes to break up `Mandelbrot::operator()`, which I think would be excessive, so that leaves good old fashioned functional testing and debugging.

### Jim Segrave <jes@j-e-s.net>

Problems with this code:

The member functions `xcoord` and `ycoord` don't work as intended. The sub-expressions involved will be evaluated as integers and conversion to `double` comes too late, their fractional part will have been truncated. A simple fix is to change the divisor from `2` (an integral value) to `2.0L`, a `double` value. Then the parenthesised sub-expressions will evaluate as doubles and the results are what is expected.

The second major problem is in the statement:

```
std::complex<double> c =
  (xcoord(x), ycoord(y)), z = c;
```

This does not call the constructor for a complex `double` with a `real` and an imaginary part. Instead, the parenthesised expression is treated as a pair of expressions separated by a comma – the `ycoord()` result is discarded and the complex `double` is constructed using the single argument constructor which sets the imaginary component to `0` and only initialises the real component. Changing the declaration to use a braced initialiser fixes this problem.

The compiler, with sufficient warnings enabled, should complain about mixing signed and unsigned values in the comparisons in the two for statements which iterate over `x` and `y`. Changing the types for `x` and `y` to `size_t`s (and similarly changing the `xcoord()` and `ycoord()` functions to expect `size_t` arguments, clears this warning.

Finally, the compiler may complain about the initialisation of the variable `data`. This variable is an C++ array of C-arrays of rows, where a row is a C++ array of a C `array` of bools. To initialise this, data needs to initialise the C++ array of rows, so it needs a pair of curly braces. Within that pair, you need an initialiser for a C array of rows, hence another pair of curly braces within the first one, which should contain an initialiser for a row. That calls for a third set of curly braces, which should contain an initialiser for a C array of bools. That needs a fourth set of curly braces around a false value:

```
matrix data{ { { { 0 }} }}
             1 2 3 4
```

1 is the initialiser list for data

2 is the initialiser list for a C array of rows

3 is the initialiser list for a row

4 is the initialiser list for a C array of bools

It's ugly, but that's what g++ 4.8 and clang 3.5.1 want to see.

With those corrections, the program compiles without warnings or errors and produces the expected output. However, it can be improved:

A Mandelbrot set is symmetric around the X axis, so it seems the program should produce a symmetric pattern. It also seems reasonable to expect that

the origin point (-0.5, 0) should be plotted. But with an even number of output lines (40 in this case), there are 20 lines printed above the X axis and 19 below the X axis. For the same reasons, there are 30 points plotted to the left of the Y axis and 29 to the right. I changed the number of rows and columns to 41 and 61, which makes the output symmetrical around the origin point, but the origin point itself is no longer plotted. I altered **xcoord** and **ycoord** so that they will plot an equal number of rows above and below the origin and an equal number of columns left and right of the origin, while still plotting the X row and Y column passing through the origin.

A final change is one of my pet obsessions – there are **if** statements in this code which control a single statement and omit any braces to make the controlled statement a compound one. Omitting the braces is a recipe for introducing bugs into code in the future. It is my firm belief that no **if**, **for**, **do** or **while** statement should ever omit the braces to make any controlled code (even an empty statement) be a compound statement.

Then anyone adding statements later will be forced to notice and respect the statement blocks.

The updated code is:

```
#include <array>
#include <complex>
#include <iostream>
using row = std::array<bool, 61>;
using matrix = std::array<row, 41>;
std::ostream& operator<<(std::ostream &os,
  row const &rhs)
{
  for (auto const &v : rhs) {
    os << "* "[v];
  }
  return os;
}
std::ostream& operator<<(std::ostream &os,
  matrix const &rhs)
{
  for (auto const &v : rhs) {
    os << v << '\n';
  }
  return os;
}
class Mandelbrot
{
  matrix data = {{ {{ 0 }} }};
  std::complex<double> origin = (-0.5);
  double scale = 20.0;
public:
  matrix const & operator()()
  {
    for (size_t y(0); y != data.size(); ++y)
    {
      for (size_t x(0); x != data[y].size();
           ++x)
      {
        // use braced initialiser list for c
        std::complex<double>
          c{xcoord(x), ycoord(y)}, z = c;
        int k = 0;
        for (; k < 200; ++k)
        {
          z = z*z + c;
          if (abs(z) >= 2)
          {
            data[y][x] = true;
            break;
          }
        }
      }
    }
    return data;
  }
private:
  double xcoord(size_t xpos)
  {
    return origin.real() +
      (xpos - (data[0].size() - 1)/2.0L)
        / scale;
  }
  double ycoord(size_t ypos)
  {
    return origin.imag() +
      ((data.size() - 1)/2.0L - ypos)
        / scale;
  }
};
int main()
{
  Mandelbrot set;
  std::cout << set() << std::endl;
}
```

### James Holland <james.holland@babcockinternational.com>

The first thing to do is to clear the decks by getting rid of any compiler warnings. My compiler issues warnings about the comparisons between signed and unsigned integer expressions that occur in the two outer **for** statements of **operator()();** The problem is that **data[y].size()** and **data.size()** return unsigned types while the variables used in the comparison, **x** and **y**, are signed types. It would be convenient to ask the **std::array** what type it uses for representing sizes and to use that. This is achieved by referring to the **std::array**'s public definition of **size_type** as shown below.

```
for (row::size_type y(0);
   y != data.size(); ++y)
for (matrix::size_type x(0);
   x != data[y].size(); ++x)
```

Unfortunately, the software still behaves in the same unexpected way. After quite a bit of effort it was discovered that **xcoord()** and **ycoord()** are not returning correct values. Investigations revealed that using both signed and unsigned types in the return expression are to blame. As noted above, **data[0].size()** and **data.size()** return unsigned types while **xpos** and **ypos** are of type **unsigned int**. C++ has set of rules it uses to determine how to handle situations like this. It turns out that, in this case, signed values are converted to unsigned types. The result of the expression is also unsigned. This has the consequence that what was thought to be small negative numbers are manipulated as if they were large positive numbers. This accounts for **xcoord()** and **ycoord()** returning unexpected values.

In our situation, there are two ways in which the problem of signed values can be resolved. Probably the most direct way is to cast the unsigned value to signed values as shown below.

```
double xcoord(int xpos)
{
  return origin.real() +
    (xpos - static_cast<signed int>
      (data[0].size())/2) / scale;
}
double ycoord(int ypos)
{
  return origin.imag() +
    (static_cast<signed int>(data.size())/2 -
     ypos) / scale;
}
```

This works well where the sizes of the dimensions of data are even numbers. Dividing such numbers by 2 (as is done in **xcoord()** and **ycoord()**) results in no loss of information. Had the sizes been represented by odd numbers, dividing by 2 would lose the fractional part

of the result. In such cases it would, perhaps, be simpler to make the divisor a **double** instead of a **signed int**. This can be achieved by simply replacing the **2** in **xcoord()** and **ycoord()** with **2.0**. Dividing an unsigned type by a **double** will result in a **double** which is inherently signed and will preserve any remainder. For the problem at hand, I have chosen the casting method as it makes clear the intentions of the programmer and, as the **size()** value is even, dividing by a **double** is not necessary.

Having corrected **xcoord()** and **ycoord()** it is disappointing to discover that the program still does not produce the expected result. Further investigation is required. It turns out that the initialisation of **c** is at fault. C++ allows variables to be correctly initialised in a variety of ways but there are a few cases where what seems obvious are inappropriate. This is one of them. In this case, the values within the parenthesise of the initialisation of **c** are being interpreted as arguments of a comma operator. The result is that the first value (**xcoord(x)**) is ignored and the second value (**ycoord(y)**) is copied to the real part of **c**. The imaginary part is automatically set to zero. This is not what is required. Correct initialisation of **c** is achieved by any of the following statements.

```
std::complex<double> c{xcoord(x), ycoord(y)};
std::complex<double> c
  = {xcoord(x), ycoord(y)};
std::complex<double> c(xcoord(x), ycoord(y));
```

The use of parentheses (**(** and **)**) and braces (**{** and **}**) both have their advantages and disadvantages so it is largely a matter of choice as to which is used. It is probably best to be consistent throughout the program, however.

Having resolved the initialisation problem, the program should work as expected with the display of the Mandelbrot set. There are, however, a couple of amendments that could be made to improve the appearance of the source code. As mentioned above, a consistent initialisation approach would be beneficial. The initialisation of data, origin and scale uses three different syntaxes. The initialisation of **x** and **y** uses yet another. I suggest, for no particularly strong reason, that brace initialisation is used throughout. Another slightly curious feature is the declaration of **k** just before the for loop. **k** would be better declared as part of the **for** loop as it is not required outside the scope of the loop.

For completeness, I provide the corrected program.

```
#include <array>
#include <complex>
#include <iostream>
using row = std::array<bool, 60>;
using matrix = std::array<row, 40>;
std::ostream & operator<<(std::ostream &os,
  row const &rhs)
{
  for (auto const &v : rhs)
    os << "* "[v];
  return os;
}
std::ostream & operator<<(std::ostream &os,
  matrix const &rhs)
{
  for (auto const &v : rhs)
    os << v << '\n';
  return os;
}
class Mandelbrot
{
  matrix data{};
  std::complex<double> origin{-0.5};
  double scale{20.0};
public:
  matrix const & operator()()
  {
    for (row::size_type y{0};
         y != data.size(); ++y)
    {
      for (matrix::size_type x{0};
           x != data[y].size(); ++x)
      {
        std::complex<double>
          c{xcoord(x), ycoord(y)}, z = c;
        for (int k{0}; k < 200; ++k)
        {
          z = z * z + c;
          if (abs(z) >= 2)
          {
            data[y][x] = true;
            break;
          }
        }
      }
    }
    return data;
  }
private:
  double xcoord(int xpos)
  {
    return origin.real() + (xpos -
      static_cast<signed int>(data[0].size())
        /2) / scale;
  }
  double ycoord(int ypos)
  {
    return origin.imag() +
   (static_cast<signed int>(data.size())/2 -
      ypos) / scale;
  }
};
int main()
{
  Mandelbrot set;
  std::cout << set() << std::endl;
}
```

## Commentary

This problem was, it seems, a little frustrating to analyse. This is partly because there were two unrelated bugs and finding and fixing just one of them was not enough to produce sensible output.

Fortunately many of our debugging tasks involve a single root cause of the problem, but it is worth remembering that this is not always the case, even in short code fragments like this one.

I think between them the critiques covered most of the issues. One issue no-one covered is that the **operator<<** defined for row and matrix are in the default namespace and use types from the **std** namespace so there is potential for a violation of the One Definition Rule if another source file in the final executable were to define the same operator on one of the types.

The odd placement of **k** outside the **for** loop was historical – the original program used the final value of **k** to produce a more flexible output than just a simple boolean value.

## The winner of CC93

There were good points made by all entrants. I also note that all three entries referred to removing compiler warnings: I take this as a good sign that warnings are getting better and/or programmers are learning to use the inbuilt static detection provided by the compiler. Paul did not actually remove the warnings though – it might have been useful for a critique to have explained why they weren't directly relevant in this case.

There are many problems that can be caused by mixing types, particularly signed and unsigned integers. In this case dividing by 2.0 rather than 2 fixed the problem, but primarily because double is signed rather than because of truncation as Jim suggested – but doing this does avoid problems if the program changes in the future.

# Letter to the Editor

Dear Editor,

I liked the idea of Vassili Kaplan's easy-to-code parser ('Split and Merge – Another Algorithm for Parsing Mathematical Expressions', *C Vu* 27.2, May 2015).

One small thing I think we should point out, though, is that the algorithm as described in that article treats the associativity of operators in a way we probably don't want.

With the commutative, transitive operators + and *, associativity doesn't matter: A*(B*C) = (A*B)*C and same goes with +. But things can be more subtle with - and /.

To take the example of 3+2*6-1 becoming 3+12-1 and then 3+11, what if that first + were changed into a -, so we have 3-2*6-1 becoming 3-12-1 and 3-11? That would give a result of -8, instead of the -10 you'd get if you did it as (3-2*6)-1 as we'd normally understand it.

Moreover, the Split and Merge algorithm, as described in the article, does not consistently treat operators as right-associative either. If the operator is preceded by another of the same or lower precedence level, then it will be left-associative: 3-12-1 would get -10. But if the operator is preceded by one of a higher precedence level, it will become right-associative with the operators before that: 3-2*6-1 would get -8. This is a bit inconsistent and I can't think of a language where we'd actually want this behaviour, although there might perhaps be one out there somewhere.

To make the algorithm consistently left-associative, we'd have to change the parser by having it break out of the merge loop and going back to the beginning after any successful merge, although it doesn't have to do this if it's already dealing with the first thing in the expression, or if the next operator is + or * and therefore can be evaluated with either associativity. Having to break out of the loop will of course increase the time complexity a little, and let's hope we're not coding a language in which some operators are left-associative whereas others are right-associative.

When inventing a new way of parsing expressions, it might be a good idea to test the code with as many expressions as possible. Perhaps write a random expression generator and feed the resulting expressions both to your code and to an established parser (such as Python's **`eval`** function) and have it alert you about any differences in the result, or at least any difference in the first 3 significant digits (as you can't count on both floating point systems to be using the same precision).

Of course, a lot would depend on how thorough your random expression generator is, so perhaps test its thoroughness first by making sure it flags up the problem with the existing version of the code. That's what Richard Feynmann would call an 'A-1' experiment: first check our observation methods by verifying we can observe what we have already, before changing things.

While such 'fuzz testing' does not provide a 100% solid guarantee of catching all cases (you'd have to use formal code-proving methods for that, preferably checked by automatic proof-checking tools and it's still possible to slip up by using them incorrectly), at least 'fuzz testing' should catch most things: I hope a decent fuzz-tester would have drawn our attention to the inconsistent associativity.

*Silas*

Silas S Brown http://people.ds.cam.ac.uk/ssb22

*Have a reputation for being reasonable*
~ Philippians 4:5, Phillip's

# Code Critique Competition (continued)

There are now many different ways to initialise variables in C++ and James' preference for use of a consistent initialisation style in a single source file is a good idea and one likely to reduce errors. (Scott Meyers' recommendation in Item 7 of *Effective Modern C++* is to use brace initialisation where possible.)

I liked Paul's final reflective question "Lastly, how could these have been avoided" – even though in this case there are few quick answers – and overall his critique wins the prize by a short head.

## Code Critique 94

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

> I had some code that used an anonymous structure within an anonymous union to easily refer to the high and low 32-bit parts of a 64-bit pointer. However, I get a warning that this is non-portable (I'm not quite sure why - MSVC and g++ both accept it) but after googling around for a solution I found one that uses #define. It all compiles without warnings now so I think it's fixed.

Can you give some advice to help this programmer? The code is in Listing 2.

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

Listing 2

```
--- address.h ---
#pragma once
typedef struct {
  union {
    struct {
      int32_t offsetLo;
      int32_t offsetHi;
    } s;
    void *pointer;
  };
} address;
// simulate anonymous structs with #define
#define offsetLo s.offsetLo
#define offsetHi s.offsetHi

--- test program ---
#include <cstdint>
#include <iostream>
#include "address.h"
int main()
{
  int var = 12;
  address a;
  a.pointer = &var;
  std::cout << "Address = " << std::hex
    << a.offsetHi << "/" << a.offsetLo
    << std::endl;
}
```

# Bookcase

## The latest roundup of book reviews.

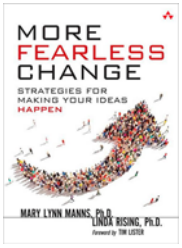
If you want to review a book, your first port of call should be the members section of the ACCU website, which contains a list of all of the books currently available. If there is something that you want to review, but can't find on there, just ask. It is possible that we can get hold of it.

After you've made your choice, email me and if the book checks out on my database, you can have it. I will instruct you from there. Remember though, if the book review is such a stinker as to be awarded the most un-glamorous 'not recommended' rating, you are entitled to another book completely free.

Thanks to Pearson and Computer Bookshop for their continued support in providing us with books.

Astrid Byro (astrid.byro@gmail.com)

### More Fearless Change




**By Mary Lynn Manns & Linda Rising, published by Addison Wesley, ISBN 978-0133966442**

**Reviewed by Ewan Milne**

*More Fearless Change* is an update to *Fearless Change*, ten years on from the publication of this catalogue of patterns for introducing organisational change (reviewed in *C Vu* 17.1). The original is a title that I have often seen recommended or referenced, but never actually read. It consists of 48 patterns covering strategies to kick-start and sustain change in a wide range of situations. The new book is neither a new edition nor a straight follow-up, instead it is more akin to the deluxe edition of a classic album, repackaged with bonus material.

The patterns from the first book are presented in their original form, and in most cases accompanied by new sections with additional insights – these range from brief comments to more in-depth sections covering a full page or two. The patterns themselves are typically three or four pages long, following a well-structured format which allows the narrative to be presented clearly and concisely.

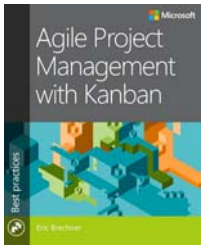
In addition to the original set, there are 15 new patterns. These patterns have been refined in PLoP conferences – I'm not sure if this was the case for the originals – and as a first time reader the old and new fit together well.

The book has three parts, with the main part containing the patterns preceded by two introductory sections. A overview of five short chapters summarises the new patterns in a logical grouping: strategize, share information and seek help, inspire others, and target resistance. This is a useful way of introducing the patterns and how they relate to each other, which would have been nice to see extended to cover the full catalogue. This is followed by two brief stories intended to give the reader some ideas for using the patterns. This section doesn't work so well: the stories are too short to be of much value, and while the first is a real case study, the second is fictitious.

No matter, the bulk of the text is devoted to the patterns themselves, which are a truly valuable resource for anyone attempting to pursue change or translate their ideas into real action.

On occasion there is an issue of cultural translation, with the writing sounding quite American – there are patterns named BROWN BAG and TOWN HALL MEETING, and a general optimism comes through which is easy to be cynical about. But such quibbles aside, this is a great book which I can imagine dipping into regularly for advice in getting things done effectively. Recommended.

### Agile Project Management with Kanban




**By Eric Brechner, published by Microsoft Press, ISBN 0-7356-9895-3**

**Reviewed by Ewan Milne**

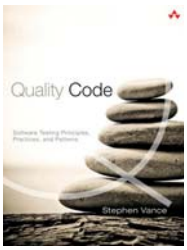
With a title that echoes Ken Schwaber's *Agile Project Management with Scrum* – also from the same publisher – you might expect this to be an authoritative guide to Kanban. Rather than a thorough analysis of theory and practice, however, instead what you get is a lightweight introduction to the topic which is focused on the practical application of Kanban as an approach to managing development teams.

Once I got over my expectation for a rather more in-depth treatment of the subject, perhaps leading with some coverage of reasons why you would want to apply Kanban, the book did make a little more sense with its launch straight into how to get the approach up and running. Indeed, this sleeves-rolled-up approach starts right from the first chapter, devoted to getting management consent. I was unconvinced by the open letter to your manager included not only in this chapter, but also as a downloadable file on the book's website, which seemed rather contrived. It provides a superficial overview of problems that Kanban can solve, plus risks and mitigations to be considered, and is clearly far too trite to be effective in the real world.

This shallow approach continues in the following chapters, which cover a 5 step quick-start guide, guidance for adapting from waterfall or evolving from Scrum, and coverage of upstream and downstream processes. Much of the advice given is sound but lacking in depth, and there is no feel for Kanban as an evolutionary change method.

The book is pitched as 'Kanban in a box': read the quick-start guide and you're up and running fast. It will help you get started, but I doubt how far along your journey it will take you. For further support, the pointers provided in the final chapter (which finally has a section on why Kanban works) will be of more use in the long term.

### 'Quality Code' Software Testing Principles, Practices, and Patterns




**By Stephen Vance, published by Addison-Wesley, 2013, ISBN 978-0321832986**

**Reviewed by Matthew Jones**

This book is quite high level and manages to cover most aspects of a very broad subject in a bit over 200 pages. The focus is on traditional unit testing, i.e. the tests we write as we code, or to test existing code at low level. There is nothing specific about system, security, acceptance testing etc, although many of the techniques and ideas can be applied to all flavours of testing. Example code is mainly Java. As a C++ programmer with only a passing knowledge of Java I found this language bias largely irrelevant because the examples are short and the code self documenting.

The book starts with a section on 'Principles and Practices', introducing the current ideas, best practice and terminology. As you would hope, there's lots of good stuff here, and it ensures readers of all levels are prepared for the rest of the book. For a complete novice it might be a little brief but all the concepts and terms can be found on the web if more detail is required.

The majority of the book is taken up by part II: 'Testing and Testability Patterns'. This is a

catalogue of patterns and techniques grouped thematically into chapters. The final chapter in the section stood out to me as it tackles something usually completely avoided when discussing testing, namely 'Parallelism'. This relatively lengthy chapter introduces some clever tricks and techniques for finding seams in concurrent code, giving you way to control your code to make testing reliable, or at least more predictable. It gave me hope that I might just consider tackling this subject systematically next time I have a problem, rather than just giving up or trying black magic.

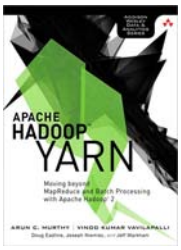
The final part of the book is a couple of short worked examples, one in Java and one in Javascript. The languages are secondary, since the chapters are mainly prose, with plenty of discussion around a limited number of listings, ending with a useful retrospective discussion.

My only real complaint was that there was very little mention of TDD, and the 'Design and Testability' chapter was one of the shortest. It might have been a conscious choice to skim over this since its a large subject, and not for the faint hearted (or, realistically, a beginner).

I started out not liking this book because it didn't contain any bold assertions or novel ideas. But that's because I was hoping for something ground breaking like the GOOS book, and I think this is really a 'software testing primer'. As a text book, it is comprehensive, detailed without waffling, and backed up with good references to take you further should you wish. The discussion often places the subject in a wider software engineering context, considering other factors such as compromises and pitfalls. This makes it clear that testing and quality can not be considered in isolation.

Overall I would recommend this book to junior programmers, or anyone coming to the subject for the first time. It would also serve well as a companion text to a training course. If your bookshelf already contains a few up to date coding and testing bibles then I don't think you will find much novelty in this book, although it might fill a few knowledge gaps.

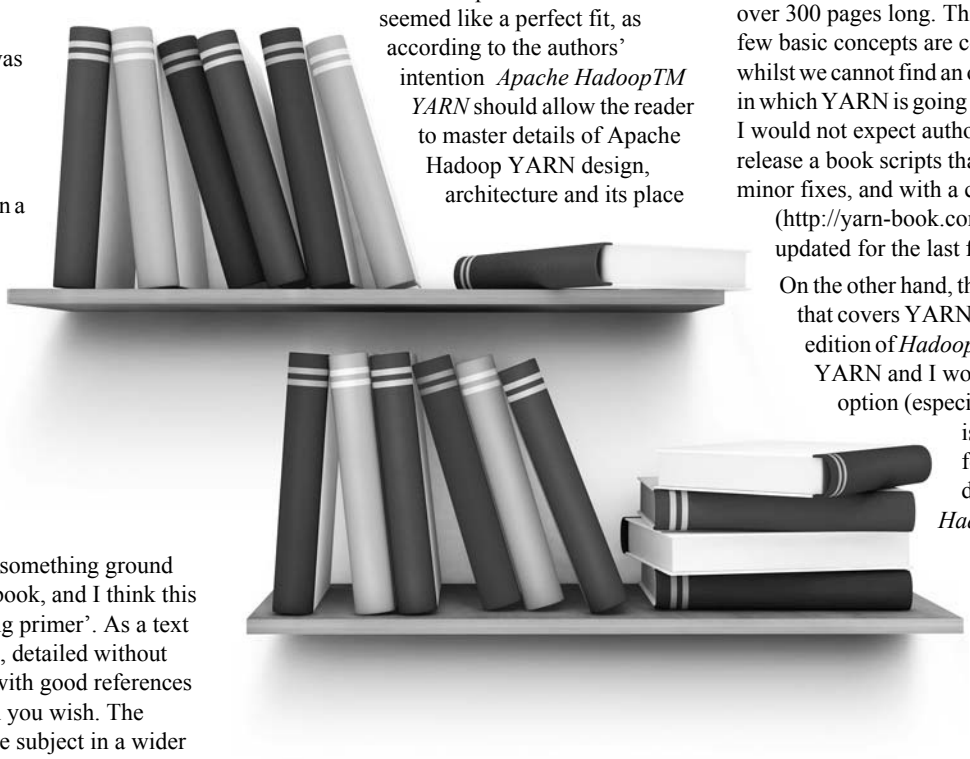
## Apache Hadoop™ YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop™ 2




**By Arun C. Murthy Vinod Kumar Vavilapalli, Doug Eadline, Joseph Niemiec, Jeff Markham, ISBN-10: 0321934504, ISBN-13: 978-0-321-93450-5, published by Addison-Wesley**

**Reviewed by Stefan Turalski**

With the advent of Hadoop 2.x we were promised YARN, a cluster resource manager that will reassure Hadoop position as a go-to big-data solution, taking it beyond MapReduce into a role of a multi-purpose platform. Therefore, I was really eager to pick up this title and learn how to utilise capabilities of YARN. It seemed like a perfect fit, as according to the authors' intention *Apache HadoopTM YARN* should allow the reader to master details of Apache Hadoop YARN design, architecture and its place in the Hadoop ecosystem. Beyond that, the reader should learn how to install, configure and administer a cluster and write both YARN applications and frameworks that would run on top of YARN. Have they succeeded?



Sadly, I think that the answer is not quite, and I would not recommend this particular book to anyone other than to a hardcore YARN fan. In fact, if you have read the introductory YARN material on either Apache's or Hortonworks' web pages, you are probably after details, administration tips, insight on YARN architecture. You would find these, and first-hand too; however, you might be in for a bit of disappointment.

First of all, by the time I received the book for review (October 2014), its content had become partially obsolete, which seems to be a common fate of publications closely covering features of fast moving open source products. Putting inevitable aside, I have few other things to point out. I was a bit surprised that the authors: Murthy, Vavilapalli, Markham, and Niemiec, who work for the Hadoop's mothership – Hortonworks Inc – decided to cover Hadoop version 2.2.0 at a time when a subsequent version was almost ready. (The 2.2.0 version was a generally available version of Hadoop 2.x that introduced YARN in October 2013, version 2.3.0 shipped in February, to be followed by 2.4.0 in April). Such a decision might have led to my second issue with the book – to put it bluntly, it feels rushed. It might be the side effect of the co-operation of 5 authors on a text just over 300 pages long. This could explain why a few basic concepts are covered multiple times, whilst we cannot find an outline for the direction in which YARN is going in the future. However, I would not expect authors of this calibre to release a book scripts that do not work without minor fixes, and with a companion webpage (http://yarn-book.com/) that has not been updated for the last few months!

On the other hand, this is still the only book that covers YARN in detail. Holmes' 2nd edition of *Hadoop in Practice* focuses on YARN and I would recommend that option (especially as the 2nd chapter is available free online) for now. However, in its defence *Apache Hadoop™ YARN* definitely gives us better understanding why things evolved into what we got in 2.x (if a reader is interested in such historical digressions).

Actually, I think it might be best to wait for Lam & Davis 2nd edition of *Hadoop in Action* or one of the best Hadoop books ever – Tom White's 4th edition of *Hadoop: The Definitive Guide*, in which the YARN chapter will hopefully get updated before it is shipped (the latest Hadoop releases (especially 2.6.0) pushed YARN capabilities quite a bit further).

Of course, we could wish for the 2nd edition of *Apache Hadoop™ YARN* as well; however, I would rather see the authors focusing on coding, as Hadoop hardcore users are definitely interested in ironing out these few remaining YARN issues at the cost of a concise, well-rounded position offering a bigger picture. After all we have the source code, right?

## View from the Chair

**Alan Lenton**
**chair @accu.org**

As April's 2015 ACCU Conference recedes into memory I'd like to thank all those who helped make it yet another successful one – organisers, speakers and attendees. Well done!

Thanks also to those who attended the AGM on the Saturday lunchtime. A couple of interesting items came up. The first was that we took an indicative vote on whether people preferred to keep the magazines printed rather than solely digital. The result was overwhelmingly in favour of dead trees rather than recycled electrons, something which reflects my own personal view. It also confirmed my view that no change to digital should be made without the passing of a specific vote to do so by the membership.

The other interesting thing that emerged was that we need to codify the status of decisions made via electronic voting. This is particularly important in cases where the following meeting is not quorate. There is also the matter of what constitutes a quorum for electronic votes. I suspect we are not the first organisation to have to tackle this issue, and we certainly won't be the last. The committee will need to bring forward amendments on this issue to the Annual General Meeting next year to clarify the matter.

And here's an interesting conundrum – do we allow electronic voting on the rules for electronic voting? Answers on a postcard...

At our last committee meeting I discovered one of the disadvantages of electronic meetings. My street had a power cut just at the wrong time! Normally I could have just grabbed my laptop, gone down to a cafe and at least joined in the discussion with text only – Chiswick High Road has more cafés per foot of frontage that anywhere else in London. Unfortunately, I just happened to be in the middle of installing a new version of Linux onto the laptop when the cut happened, so it was unusable.

Power cuts like these make one very aware of just how dependent we are on power in our society.