

The magazine of the ACCU

www.accu.org

{cvu}

Volume 22 Issue 2 May 2010 £3

Features

Lazy Initialisation of Shared Resources
Iain Charlton

Live to Love to Learn
Pete Goodliffe

Do Repeat Yourself - Code Katas
Frances Buontempo

Developing a PIC based board game timer using TDD
David Sykes

A Game of Nerve
Richard Harris

ACCU Conference 2010
Paul Grenyer

Features Editor

Steve Love
cvu@accu.org

Regulars Editor

Jez Higgins
jez@jezduk.co.uk

Contributors

Frances Buontempo,
Iain Charlton, Pete Goodliffe,
Paul Grenyer, Richard Harris,
Roger Orr, David Sykes

ACCU Chair

Jez Higgins
chair@accu.org

ACCU Secretary

Alan Bellingham
secretary@accu.org

ACCU Membership

Mick Brooks
accumembership@accu.org

ACCU Treasurer

Stewart Brodie
treasurer@accu.org

Advertising

Seb Rose
ads@accu.org

Cover Art

Pete Goodliffe

Repro/Print

Parchment (Oxford) Ltd

Distribution

Able Types (Oxford) Ltd

Design

Pete Goodliffe

Choose Your Language Carefully

A common question I get asked at interviews for programming jobs (as a contractor, this is a fairly frequent experience) is, 'Which language do you prefer: C# or C++?' I've been using C# for some years, having come to it from a C++ background, and it's an interesting question for me, but the interviewers don't always get the response they seem to be expecting!

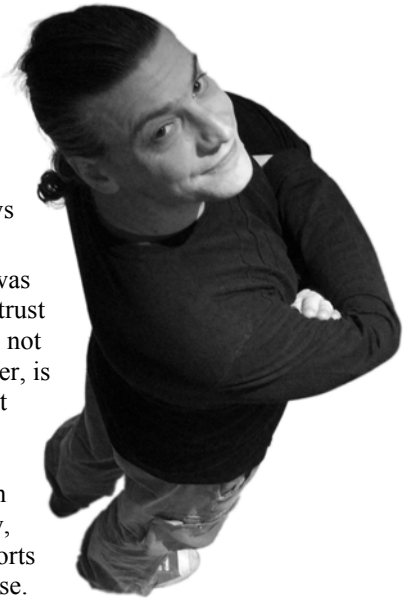
When I first started working in C#, the first thing I missed was deterministic destruction. It was a while before I learned to trust the garbage collector and allow at least some of my types to not implement the IDisposable interface. Top of my list, however, is the whole idea of complexity guarantees for the various .Net collection types and the operations they allow. When I'm using C++'s STL containers and algorithms, I know how to trade off one for another against speed or size constraints. In .Net, there is no indication of whether a `List<T>` is an array, linked-list or (in .Net 3.5) an associative container – it supports some operations which might be appropriate for each of those. There's no way to discover how complex an operation is for a given container type. One expects a hash table to have a faster look-up than a list, in general, but the documentation is unforthcoming on this point.

The things in C# I miss most when writing C++ are mainly about writing succinct code. The sheer richness of the .Net Framework Library means I can do many things 'out of the box', without resorting to 3rd party libraries. But, more fundamentally than that is the interface keyword. This one thing makes writing C# code a little less verbose, and a little easier to get right, than the equivalent C++ code. This leads on to the fact that programming to true interfaces is not as widespread in C++ as it is in C# or Java. Using 3rd party libraries in C++ can be more traumatic due to the lack of that central idiom.

So, which do I prefer? Both. Neither. Something else. It depends. Mostly, on what I'm trying to achieve. As with many things, it all depends on what problem I need to solve.



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

- 19 Desert Island Books**
Paul Grenyer maroons
Pete Goodliffe.
- 21 Code Critique
Competition #63**
Set and collated by
Roger Orr.
- 24 Regional Meetings**
The latest London
meeting.
- 24 Inspirational (P)articles**
Frances Buontempo
keeps us inspired.
- 25 Conference Retrospective**
Paul Grenyer reflects on
this year's conference.

FEATURES

- 3 A Timer for Rummikub**
David Sykes develops a PIC-based timer using test-driven
development.
- 8 Live to Love to Learn (Part 3)**
Pete Goodliffe concludes his journey of self-improvement.
- 11 A Game of Nerve**
Baron Muncharris has another tall tale.
- 12 Lazy Initialisation of Shared Resources**
Iain Charlton looks at managing object lifetime from birth
to death and beyond.
- 18 On a Game of Strategy**
The Baron's student acquaintance performs his analysis.

REGULARS

- 29 Bookcase**
The latest roundup of
ACCU book reviews.
- 32 ACCU Members Zone**
Reports and membership
news.

SUBMISSION DATES

C Vu 22.3: 1st June 2010

C Vu 22.4: 1st August 2010

Overload 98: 1st July 2010

Overload 99: 1st September 2010

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.

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.

COPYRIGHTS AND TRADE MARKS

Some articles and other contributions use terms that are either registered trade marks or claimed as such. The use of such terms is not intended to support nor disparage any trade mark claim. On request we will withdraw all references to a specific trade mark and its owner.

By default, the copyright of all material published by ACCU is the exclusive property of the author. By submitting material to ACCU for publication, an author is, by default, assumed to have granted ACCU

the right to publish and republish that material in any medium as they see fit. An author of an article or column (not a letter or a review of software or a book) may explicitly offer single (first serial) publication rights and thereby retain all other rights.

Except for licences granted to 1) Corporate Members to copy solely for internal distribution 2) members to copy source code for use on their own computers, no material can be copied from C Vu without written permission from the copyright holder.

A Timer for Rummikub

David Sykes develops a PIC-based timer using test-driven development.

A recent game of Rummikub became frustrated by lack of a simple timer. One of the players insisted on stretching his time, but none of our watches had second hands. We tried a sandglass, but this made things worse as finishing a turn early meant you had to wait for it to be reset.

What we wanted, I thought, was a simple box with a button and a light. When you finish your turn you press the button. If the button wasn't pressed before the end of a minute then the light came on, and you took your penalty. There are undoubtedly a million solutions to this, but none would look like the picture I had in my head. One of my hobbies is experimenting with PIC processors, and I knew I could make something that worked the way I wanted.

My paid job involves development and maintenance of a massive legacy system, and much of the time using TDD is not appropriate. Extracting code to test can sometimes take an order of magnitude longer than 'just fixing the bug'. However for new development, and my personal projects, I have found test driven development to be of immense benefit in producing highly reliable code much more quickly, and more enjoyably too. When I began applying it to developing code for PIC processors it encouraged me to isolate myself from the actual hardware, and greatly reduced the time spent with the target circuit, and frustration with the lack of debugger or emulator. This article documents my experience with a very small example project.

What I wanted, and how I thought I would get it

What I imagined happening was an initial short period of fast flashing to show that the button press had been registered, a brief flash every ten seconds to show signs of life, a bit more fast flashing for the last few seconds, then full on. Hold this for a while, say 10 seconds, then put the device into a low power sleep, hopefully with sufficiently low power consumption to remove the need for a power switch. Pressing the button mid sequence will restart the sequence, or wake it up if it has gone to sleep.

Just about any PIC processor would do, but the one that comes with the PIC Programmer I use, the Flash Starter Kit [1], is the 12F675 (£1.77 Maplin). With up to 6 IO pins, an internal oscillator, two timers, and an

```
#include <pic.h>
__CONFIG(INTIO & WDTDIS & MCLRDIS & BORDIS &
UNPROTECT & PWRTEN);

void main()
{
    int iostate, delay;
    GPIO = 0; // clear all I/O pins
    TRISIO = 0b00000000; // make all I/O pins
    outputs

    // Sequence all the output pins.
    while (1)
    {
        for ( delay = 0 ; delay < 1000 ; delay++){
            GPIO = iostate++;
        }
    }
}
```

Listing 1

interruptible sleep mode this chip has everything that is needed for such a simple circuit (Figure 1).

My choice of language is C. It's almost as efficient as assembler, and compilable by the C++ compilers I use for the Test Driven Development. The HI-TECH C compiler [3] for PIC chips is available free, and more than adequate.

The basic hardware framework

The first thing I like to do is start with something to see, and Listing 1 shows the minimum code required to do something visible. The code simply configures the IO pins to be outputs, and then increments the output register with a delay to make the leds flash at a visible rate.

The code is placed in a file, for example `GameTimer.c`, and built on the command line with

```
pic1 GameTimer.c --CHIP=12F675
```

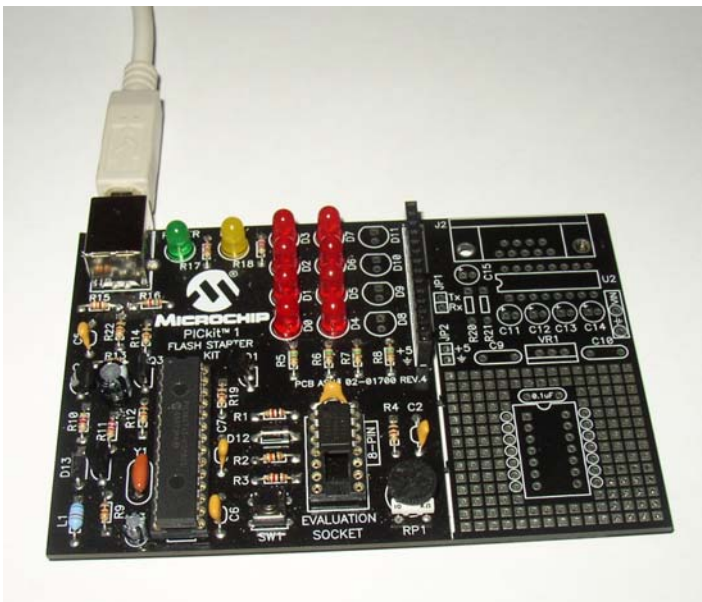
This produces a `GameTimer.hex` file containing the compiled code. Burning the hex file to the chip using the PicKit Classic Programmer results in a bunch of flashing leds.

Introducing a timer interrupt

Many PIC projects react to timed events, as will this one. Timed events can be triggered using one of the two timers. The internal clock drives the timer which, when enabled, causes an interrupt when it overflows to zero. Using the internal 4MHz clock, the timers are incremented every 1uS. Timer 1 is a 16 bit timer, and so it overflows every 65536uS, approximately 15 times a second. This provides ample timing resolution.

Listing 2 shows the framework code using Timer 1 interrupts. Timer 1 interrupts are enabled by setting the GIE, TMR1IE and PEIE flags, the processor then waits. Every 65536uS the timer overflows, causing an interrupt which calls the interrupt routine. The interrupt routine does what

Figure 1



DAVID SYKES

David studied electronic engineering, but was seduced into software by the 6502. He now develops software producing catalogues for products including school supplies and industrial chemicals. He can be contacted at david@sykes.org.uk



```

#include <pic.h>
_CONFIG(INTIO & WDTDIS & MCLRDIS &
  BORDIS & UNPROTECT & PWRTEN);

int iostate = 0;

void main()
{
  GPIO = 0; // clear all I/O pins
  TRISIO = 0b00000000;
  // make all I/O pins outputs

  TMR1IE = 1; // TMR1 Overflow Interrupt
             // Enable bit
  GIE = 1;   // Global Interrupt Enable
  PEIE = 1;  // Enable all unmasked peripheral
             // interrupts
  TMR1ON = 1; // Enable Timer1

  while(1)
  { // Wait for interrupts
  }

}

void interrupt InterruptRoutine(void)
{
  GPIO = iostate++;
  TMR1IF = 0;
}

```

it needs to, clears the TMR1IF interrupt flag, and returns the processor to what it was doing, which in this case is simply waiting.

Development

This completes the basic framework from which the game timer will be constructed. The desired flashing of the led can be considered as a simple sequence of toggling the led state between on and off at specific times. Times between state changes will be measured in interrupt counts, and a single byte could represent up to $256 / 15 = 17$ seconds. If these timings were relative then a byte array would be sufficient to specify the entire flashing sequence. For example one second of around 5Hz flashing might be represented as:

```
{1,2,1,2,1,2,1,2,1,2,0}
```

i.e. on for one interrupt duration, off for two, repeated five times for a total number of 15 interrupts, ~1s. A null terminator will clearly be required to identify the end of the sequence.

```

unsigned char flashingSequence[] = {
  10, 20, 30, 0};
void TestFlashSequence()
{
  Assert(!GetLedState()); // led off
  ProcessInterrupt();     // First action is to
                          // turn the led on

  Assert(GetLedState());
  for (int count = 0 ;
       count < 10 ; count++) ProcessInterrupt();
  Assert(!GetLedState()); // 1st timing event
  for (int count = 0 ; count < 20 ; count++)
    ProcessInterrupt();
  Assert(GetLedState()); // 2nd
  for (int count = 0 ; count < 30 ; count++)
    ProcessInterrupt();
  Assert(!GetLedState()); // 3rd...
}

```

Development can be done with any compiler and test framework that handles C code. My personal projects get done on whatever machine happens to be at hand, which could be a Mac, PC or Linux machine with or without a test framework. For that reason I tend to start simply by using the most trivial of test functionality, and develop it from there if required:

```

void Assert( bool value ) {
  if (!value) std::cout << "TEST FAILURE\n"; }

```

The first code to develop is a routine that is called by the interrupt routine, and that changes the state of the led according to the timing data. For the first test I choose to test a few representative samples, and check that the led changes when I expect it to (Listing 3).

Now is a good time to consider how the test sequence data will be passed to the `ProcessInterrupt` routine. In most environments this might be done through some form of dependency injection.

```
ProcessInterrupt(flashingSequence);
```

However this involves pointers, and on PIC chips the size of the 12F675 pointers are a big deal, to be avoided if possible. Ultimately there will only be one flashing sequence, so the final code should be able to refer to the data as a fixed array. To enable this, and also gain considerable flexibility in mocking access to the hardware, I use a preprocessing seam [4]. The code is written in a file that is included by both the test and the target program. Data items and references to the hardware are mocked simply by providing an alternative definition before the file is included.

Adding mocks for the led state we get:

```

bool ledState(0);
bool GetLedState() {return(ledState);}
void SetLedState(bool state){ledState = state;}
unsigned char flashingSequence[] = {
  10, 20, 30, 0};
#include "ProcessInterrupt.inc"

```

This fails to compile because there is no `ProcessInterrupt.inc` file or `ProcessInterrupt()` routine. The first step is to provide a skeleton file with definition.

```
void ProcessInterrupt() {}
```

The project now compiles, but the tests fail. So far so good. The final step is to provide code to pass the test, shown in listing 4.

I'd like to say this was the first attempt, and I generally find with TDD that by this point so much thinking has gone in that first attempts work more often than not. This time, however, I started countdown with an initial value of 0, and used a post decrement: `countdown--`. This adds one more cycle to each step than I expected, which for this project is irrelevant but for some projects could be a key source of failure. In any event the second version passed the tests.

According to the tests we have the code we need, so lets see how it looks on the PIC. The code is included in a C wrapper, but before the include line the hardware versions of `SetLedState` and `GetLedState` are defined, together with the `flashingSequence` data. A production

```

unsigned char sequencePosition = 0;
unsigned char countdown = 1;
void ProcessInterrupt()
{
  if ( --countdown == 0)
  {
    SetLedState(!GetLedState());
    countdown =
      flashingSequence[sequencePosition++];
  }
}

```

```

#include <pic.h>
#define LED_ON 0x10
#define LED_OFF 0
unsigned char led = 0; // bool not implemented
#define SetLedState(state){ GPIO = led =
state?LED_ON:LED_OFF; }
#define GetLedState() (led == LED_ON)
unsigned char flashingSequence[] = {
    2,1,2,1,2,1,2,1,2,138,3,150,3,150,3,150,
    3,150,3,134,2,1,2,1,2,1,2,1,2,1,2,1,2,1,
    2,1,2,1,2,1,2,1,2,1,150,0};
#include "ProcessInterrupt.h"
#include "ProcessInterrupt.inc"

Prototype the function in its header file,
ProcessInterrupt.h
void ProcessInterrupt(void);

```

version of the circuit might have the led attached to IO pin 0, however the Flash Starter Kit has the leds set up as a multiplexed grid attached to IO pins 1,2,4 and 5. On the flash starter kit setting GPIO to 0x10 will turn LED0 on.

This project is simple enough that writing a subroutine to set and get the LED state would go unnoticed, but I can't bring myself to do it, and use `#define` instead. Note that the GPIO register cannot be trusted to read the same value that was written, so a stored copy is required and Listing 5 goes in to `ProcessInterrupt.c`.

The interrupt routine is also modified to call the `ProcessInterrupt` function, and to clear the interrupt flag when complete.

```

#include "ProcessInterrupt.h"
void interrupt InterruptRoutine(void)
{
    ProcessInterrupt();
    TMR1IF = 0;
}

```

The code is compiled with

```
picl GameTimer.c ProcessInterrupt.c --CHIP=12F675
```

On power up the leds will now flash with the programmed sequence, but once only and it will not respond to the press of the button.

Pressing the button

Pressing the button mid sequence will need to reset the sequence to its initial position, for which we require a routine that resets the sequence state. One way to test this is to start the sequence, resetting it midway and checking to ensure it has been reset as expected. The test cannot rely on the state left by the previous test, and so starts by resetting the state (see Listing 6).

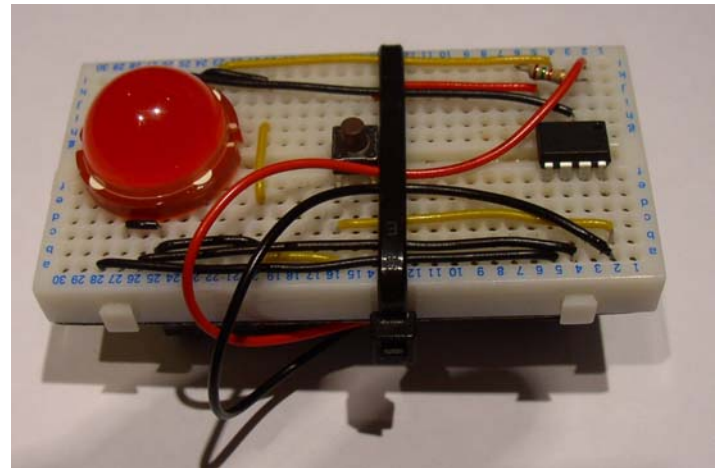
With a simple stub for `ResetState` in `ProcessInterrupt.inc` the code compiles and the test fails.

```
void ResetState(void) {}
```

```

void TestResetState()
{
    ResetState();
    for (int count = 0 ;
        count < 5 ; count++) ProcessInterrupt();
    Assert(GetLedState());
    ResetState();
    Assert(!GetLedState());
    ProcessInterrupt();
    Assert(GetLedState());
}

```

**Figure 2**

Now to write code that passes the test:

```

void ResetState(void)
{
    sequencePosition = 0;
    countdown = 1;
    SetLedState(0);
}

```

This time the tests passed first time.

Implementing the button press code on the PIC

On the 12F675 chip the GPIO3 pin is input only, and therefore an obvious choice for the button input. Unsurprisingly the PIC starter kit provides a button attached to this pin (Figure 2), tied high so the input goes low when the button is pressed. Checking for the button press could be done either in the interrupt routine or the main loop. Placing it in the interrupt routine is simplest, keeping all the functionality there avoids any 'multithreading' issues.

```

void interrupt InterruptRoutine(void)
{
    if (!GPIO3)
        ResetState();
    ...
}

```

After adding a prototype for `ResetState` in `ProcessInterrupt.h` the code builds and runs first time. Pressing the button now resets the sequence back to the beginning.

Going to sleep

When the sequence completes the chip is to go into a low power sleep mode. Putting the chip 'to sleep' requires setting various hardware flags and executing a `SLEEP` instruction. The chip can be woken from sleep by one of several interrupts, the most significant of which for this project is a change of state on the IO pins. This means that the button can be used to wake the chip without any extra hardware.

Because the sleep mode is terminated with an interrupt it must be initiated in the main code. However the end of the sequence is detected in the interrupt routine. Thus the interrupt routine must signal to the main routine that a sleep is required, which can be done by simply setting a flag.

The code to test the initiation of sleep mode is as follows Listing 7.

To get the code to compile the `sleepRequested` flag is defined in `ProcessInterrupt.inc`

```
unsigned char sleepRequested = 0;
```

The code now compiles, but the test fails. To pass the test `ProcessInterrupt` is modified to set `sleepRequested` when the 0 terminator is reached (Listing 8).

Listing 7

```

void TestSleep()
{
    sleepRequested = false;
    ResetState();
    for (int count = 0 ;
        count < 60 ; count++) ProcessInterrupt();
    Assert(!sleepRequested);
    ProcessInterrupt();
    Assert(sleepRequested);
}

```

Listing 8

```

void ProcessInterrupt()
{
    if ( --countdown == 0)
    {
        SetLedState(!GetLedState());
        countdown =
            flashingSequence[sequencePosition++];
        if (countdown == 0)
            sleepRequested = 1;
    }
}

```

Listing 9

```

extern unsigned char sleepRequested;
...
if (sleepRequested)
{
    TMR1ON = 0; // Stop the timer 1 from waking us
    up
    sleepRequested = 0; // Clear the request
    GPIO = 0; // Turn all the leds off
    GPIE = 1; // Enable Port Change Interrupt
    IOCB = 8; // Set Port Change Interrupt for
    // button input

    SLEEP();
    IOCB = 0; // Disable Port Change Interrupt
    GPIE = 0;
    TMR1ON = 1; // Timer 1 back on
}

```

Listing 10

```

void interrupt InterruptRoutine(void)
{
    ...
    TMR1IF = 0;
    GPIF = 0;
}

```

Implementing the sleep on the PIC

Now that the interrupt routine sets the `sleepRequested` flag as required all that remains is to detect the flag change in the main loop and activate the sleep mode.

This involves turning off the led to save power, halting the timer to prevent it triggering an interrupt, and setting GPIO3 to trigger a wake up interrupt on change (Listing 9).

If the port change interrupt is triggered then its appropriate flag must also be cleared at the end of the interrupt routine, just like the timer 1 interrupt. There is no harm in doing this every interrupt (Listing 10).

Turning the prototype into a device

That completes the prototype, and if usb power is available then this could be used as it is.

More usefully, however, this needs to become a stand alone circuit. The 12F675 will cope with a supply of between 2 and 5.5v, so two 1.5v batteries will do as the power supply. The schematic for the device is shown in Figure 3, drawn using McCAD Schematics LITE [5].

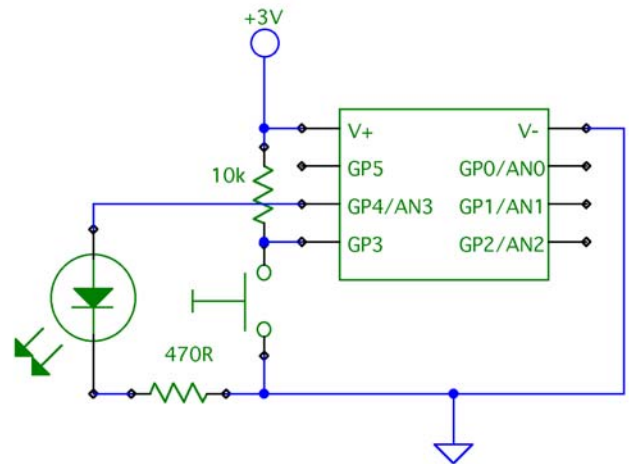


Figure 3

In my mind I picture a coloured half sphere, 2 or 3 inches in diameter, that sits on the table. Pressing the sphere starts or resets the timer. The sphere lights up, and maybe makes a sound, when the time is up. I have yet to find a suitable sphere.

Conclusion

This article should be sufficient to get somebody started with developing basic embedded systems with very little cost. I hope it also shows ways of utilising TDD to speed up development, and avoid dependency on expensive emulators or logic analysers.

The PIC 12F675 is a remarkable little device, but there are hundreds of other chips in the PIC range, and many other types of easy to access microcontrollers such as the Arduino. The possibilities are endless. ■

References

- [1] <http://www.maplin.co.uk/Module.aspx?ModuleNo=37172> (Maplin £45. An alternative is the Velleman PIC Programmer[2])
- [2] <http://www.maplin.co.uk/Module.aspx?ModuleNo=48074>
- [3] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en542849
- [4] *Working Effectively with Legacy Code*, Michael Feathers.
- [5] www.mccad.com

Join the ACCU

visit

www.accu.org
for details

Why does a leading Quant Fund want to recruit leading software engineers?

You're developing software that's highly efficient and never fails. We're using software to generate positive returns in chaotic financial markets. We think we should talk.

Who Are We?

OxFORD ASSET MANAGEMENT is an investment management company situated in the centre of Oxford. Founded in 1996, we're proud of having generated positive returns for our investors each year, including 2008, particularly as many assets are managed for pensions, charities and endowments. We blend the intellectual rigour of a leading research group with advanced technical implementation. We like to maintain a low profile, nobody comes to work in a suit and we are a sociable company.

What Do We Do?

We use quantitative computer-based models to predict price changes in liquid financial instruments. Our models are based on analyzing as much data as we can gather and we actively trade in markets around the world. As these markets become more efficient, partly because of organizations like ours, we need to develop improved models in order to remain competitive. Working to understand and profit from these markets provides many interesting mathematical, computational and technical challenges, especially as markets become increasingly electronic and automated. We enjoy tackling difficult problems, and strive to find better solutions.

Who Do We Want?

Although most of us have advanced degrees in mathematics, computer science, physics or econometrics from the world's leading universities and departments, we are just as interested in raw talent and will consider all outstanding graduate applicants. We expect all prospective candidates to work efficiently both in a team environment and individually. We value mental flexibility, innovative thinking and the ability to work in a collaborative atmosphere. No prior experience in the financial industry is necessary. We want to hear from you if you are ambitious and would relish the challenge, opportunity and excellent compensation offered.

Software Engineers

We are seeking outstanding software engineers to develop and maintain system critical software. You will be responsible for all aspects of software development on a diverse range of projects, such as automating trading strategies, integrating third party data into our system and the development of data analysis tools.

You will have the following:

- A high quality degree in computer science or related discipline
- Several years C++ experience, including the use of the STL and Boost
- The ability to write high performance code without sacrificing correctness, stability or maintainability
- A good understanding of Linux, scripting, working with large numerical data sets and large scale systems
- Mental flexibility, innovative thinking and the ability to work quickly in a collaborative atmosphere

Any experience in the following areas would be advantageous: numerical analysis, optimisation, signal processing, statistics, machine learning or natural language processing.

Benefits

Health Insurance (including family)
Pension Scheme
Life Insurance

Closing Date

Ongoing

Start Date

Spring 2010

How to apply

Please email or post CV with covering letter, stating which position you are applying for to:

Dr Steven Kurlander
Oxford Asset Management
Broad Street
Oxford
OX1 3BP
United Kingdom

accu@applytooxam.com

+44 1865 258 138

www.oxam.com

Live to Love to Learn (Part 3)

Pete Goodliffe concludes his journey of self-improvement.

Tell me, and I will forget. Show me, and I may remember. Involve me, and I will understand.

~ Confucius

This is the third article in my mini-series on how software developers learn. So far we've investigated why this is important, and what we should be learning. We've looked at some learning models and uncovered many valuable facts about the way the brain stores, processes and retains information. But that's all been background. We've not yet actually put any wheels on our boat, and looked at the practical tools and processes that will help us to learn better.

Three issues in, it's about time to rectify this omission. In this final instalment, we'll plumb some much more practical depths. Less hand waving, more brain bending.

I embarked on writing this series on learning because it is vital to invest in your learning. Programmers are knowledge workers – learning (and then applying) new stuff is what we're constantly doing.

The pragmatic programmers describe a vivid and potent metaphor for learning – they talk about your *knowledge portfolio* [1]. Consider your current working set of knowledge like a portfolio of investments. This metaphor beautifully highlights how we should *manage* the information we have gathered, carefully investing to keep our portfolio current, and bringing in new investments to strengthen it. Consider which items you should retire from your portfolio, to make room for other things. Be aware of the risk/reward balance of the items in your portfolio. Some things are common knowledge, but a safe investment to hold – they are low risk, easy to learn about, and guaranteed to be useful in the future. Other investments are riskier – they may not be mainstream technologies and practices, so studying them may not pay off in the future. But if they do become mainstream then you will be one of a smaller set of people experienced in that niche, and so able to exploit that knowledge more. These higher risk knowledge investments may pay much greater dividends in the future. You need a good spread of risk and a healthy range of knowledge investments.

Purposefully manage your knowledge portfolio.

The angle of approach

Whenever I look at a topic in programming (and by now, I've looked at quite a few), the essence of success always seems to be determined by a single issue. And in the field of learning it seems as clear as ever: to be successful at learning you have to adopt the correct *attitude*. Your attitude towards learning will directly determine the quality of your learning.

We saw in the previous column how mental state can affect performance, and the social psychologist Carol Dweck's research into the affect of mental attitude on ability to learn: that those who believed they could learn did so far more effectively than those who thought they couldn't [2]. *Self-belief* is an incredibly important attitude.

We must take *responsibility* for learning – it is something we must do for ourselves, we cannot expect others to make us learn. We must adopt an attitude of *continuous learning*, never believing that we know it all or that we know enough – there must always be a *hunger* for new knowledge, a

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



Thanks for all the fish

It's always great to get feedback about articles published in CVu. Every author appreciates it. But the honest truth is that authors tend to hear very little from the readership, either good or bad. That's just par for the course.

However, this series on learning has garnered a lot more feedback than many of the other articles I've written. I find this genuinely interesting, and rather heartening. It shows how many software developers are considering the importance of better capitalising the power of their brain, and want to improve their learning skills.

Thanks for the feedback, I hope you all have found these articles useful and interesting. And if you don't, then please blame the editor for ruining my fact-packed prose... (Sorry, Steve!)

Perhaps you might also like to take a little time to tell another CVu or Overload author how much you appreciate their article. It's always appreciated.

driving *curiosity*. This couples with *humility*; again recognising that we don't know everything and that we can learn from anyone and anything.

The most effective tool in learning is to *care* about what you're learning, and to be prepared to invest effort in order to learn. Dweck wrote about this in [3]: Effort is one of the things that gives meaning to life. Effort means you care about something, that something is important to you and you are willing to work for it. If you are prepared to invest effort into learning about something, it not only shows that you value the topic, it means that you intend to and desire to learn.

Getting mechanical

We'll conclude our look at learning with some useful and very practical mechanisms to improve our learning. There are some very useful tools, processes, and plans that we can employ to boost our learning effectiveness.

First, what tools are available? As strange as it sounds, the most potent and powerful tool we can employ is the *entire human body*. A human being is a large interconnected system, the brain is our CPU, but it's connected and influenced heavily by the rest of our fleshy substance.

Let's start by focussing on the brain. Sadly, we're not really ever given a user's manual for the brain. (And even if we were, most men wouldn't read it anyway...)

Firstly, we must keep this thing well prepared mechanically. Use the correct fuel: enjoy a healthy diet and make sure you keep well hydrated. It is said that the best brain foods are protein-rich foods such as meat, fish and cheese: these are good sources of amino acids. Also ensure you give yourself a good supply of the appropriate vitamins and minerals (vitamin B, sodium, potassium, and omega-3 are all said to be important for brain function). To keep your brain working in top condition (and let's be clear – to be an effective developer you really should consider how to keep your brain working as well as you can) you should pay attention to this kind of thing.

Other very obvious basic brain mechanics are to get plenty of sleep and avoid as much stress as possible to enable you to be less distracted and better able to think. It sounds glib advice, but how often do you consider how continual stress or burning the candle at both ends affects your effectiveness?

Using all of the brain

In the last column we investigated the left/right brain modes. Most developers lean towards left-brain modes of thinking, so to maximise use

of our potential brain power we need to dial down the left side of our brain to give the right side a fighting chance. Otherwise we'll only exploit half of our potential learning power.

Do remember, though, that both modes of thought are essential. We really need to work out how to get both brain modes to work in concert. In order to think and learn effectively we must be able to bring both 'sides' of our brain into use.

There are some very basic documented ways to stimulate the right side of your brain. *Cross-sensory feedback* will stimulate parts of the brain that we don't normally exercise. Consider trying some of these actions whilst learning:

- listening to music whilst you work,
- doodling whilst you think (yes I am paying attention in your meeting, look at how many doodles I've made...),
- fiddling with something (a pen or a paperclip, perhaps),
- talking whilst you work (vocalising what you're doing or learning, it really does help you retain more knowledge),
- making thought processes concrete rather than purely intellectual – like modelling things with building blocks or CRC cards,
- meditative practices (many help you attain greater focus and cut out distractions).

These actions can help to invoke the right brain whilst performing activities that you might naturally focus more on left-brain operation. Each of these expand sensory input and serve to activate more neural pathways than normal.

Multiple input, maximal output

Different personality types learn in different ways. I can't prescribe the best method of learning for everyone. However, try to learn a topic by absorbing information from many different information sources. In this modern connected world we're presented with many media forms:

- The written: e.g. books, magazines, blogs.
- The spoken: e.g. audio books, presentations, user groups, podcasts, courses
- The visual: e.g. video podcasts, tv shows, performances

Some people will respond better to particular media. What works best for you? For the best results mix several of these sources. Use podcasts on a topic to reinforce what you're reading in a book. Attend a training course and read a book on the topic, too.

Use as many input types as possible to maximise your learning potential.

Take note

Whilst learning, grab a notepad and capture information as you uncover it, rather than let it wash over you.

This serves two purposes. Firstly, it keeps you focussed and helps you to maintain concentration on the topic. It's a basic idea, but remarkably helpful. Secondly, even if you throw those notes away immediately afterwards, the cross-sensory stimulation will aid your recall of facts.

Takes notes as you learn. Even if you throw them away.

The practice of learning

Our key to learning is to pay attention to the *practice of learning*. Or rather, the verb: *practise*. No, learning is an activity; unless you remain consciously and actively involved you will not learn effectively. Learning without practise, without applying the new knowledge, will not lead to deep understanding or long-term recall. We need to perform what is known as *deliberate practise* [3].

It is perfectly possible to perform pointless practise; to not pay full attention and waste your own time and effort. I'm reminded of some high-school recorder lessons that I was subjected to (ooh, many years ago now). This class was dumped into a room, each given a 'learn the recorder' book, and expected to get on with it. That teaching method certainly leaves a lot to be desired. And it's obvious that a group of kids that don't care at all about playing recorder will never learn to play the recorder. There was no motivation, a failing attitude.

But more importantly, a thirty minute lesson where no one is paying attention or deliberately trying to learn will never lead to learning. At the next lesson all the kids started back at page one again; they'd forgotten everything they'd read previously. There was no actual learning taking place in those lessons. A whole term saw very few people progress beyond the first half-hour of the teaching plan. The lessons were literally a load of hot air. And raucous screeching sounds.

Learning without doing is a fruitless task.

Ensure that your learning regimen involves mindful practice. Powerful techniques to consider here are Coding Katas [4] and Coding Dojos. Katas were introduced into the development world by Dave Thomas of pragmatic programmer fame. A kata, like its martial arts synonym, is a small task or process that a student can repeat deliberately in order to learn a skill. Perhaps a simple coding exercise or refactoring task. Coding Dojos are meetings where programmers gather to perform katas together. Specific, deliberate learning meetings.

Dojos are becoming increasingly popular. Jon Jagger and Olve Maudal ran an excellent Dojo at this year's ACCU conference, for example.

Networking

If the brain is the CPU of our learning machine, then we should consider other uses of the machine that will aid learning. In the modern world, few computers are an island. Networking is an essential learning aid. The social context enhances our learning, and adds accountability (aiding attitude) and greater interactivity (aiding deliberate practice and cross-sensory feedback).

These are all invaluable social learning practices:

- pair programming
- study groups
- mentoring and teaching (by teaching others you solidify your knowledge, you are forced to learn more yourself, and as you see a newbie begin to gain a wider picture you will realise flaws in your own knowledge, or see foundational knowledge in a fresh light that will be really beneficial)
- writing articles (for magazines, for the web, for blogs)
- discussion (perhaps in the pub)

I have a cunning plan

In the common work scenario you start from ground zero and need to get up to speed with something in super-fast time. You need to pick up a new codebase, a new set of technologies, and a new problem domain. And be effective and productive almost immediately.

This is rarely possible. But you can be effective fairly rapidly as long as you recognise that you are starting out at the *novice* Dreyfus level (remember our look at the Dreyfus Model of Skills Acquisition in the last column [5]?). In order to be effective at this point you must find a good set of rules to follow, since novices rely on rules to get work done.

But you want to progress beyond novice level, don't you?

In order to learn effectively you need to put in place a considered learning plan. Now, if I wanted to get from my home in Cambridge to visit a friend in Inverness, I wouldn't just jump in the car and set off. I'd need to plan a route first. I wouldn't even set up my satnav and follow its instructions. Should I really trust a device to know best? Does it know how I prefer to

travel, any particularly busy roads at this time of year, which roads the authorities have decided to perform military assaults on today, etc? Similarly, does a teacher know better about how to get me from here to knowledge?

So I'd have to carefully plan a route first. Maybe getting in the car isn't actually the best way to get to Inverness – perhaps I should instead fly there, and hire a car at the other end.

Put in place a deliberate learning plan.

So how might we apply this practically to our learning? Start by recognising exactly what you do know right now. Determine what you *need* to know. Weigh up the possible routes to get there: books, courses, web research, podcasts, etc.

Then determine two or three waypoints along the way you can purposefully aim for. Don't aim too far into the future, as once you have learnt more you may need to re-plan your learning route. Work out how to learn enough to get to those first waypoints. Work out exactly how you'll know when you've got there. (Consider this *Test Driven Learning* – work out some tests (katas, specific tasks, etc) that will prove you know something. You can use these same tests in the future to check for regressions in your learning). Then once you have a specific plan in place, begin learning.

Planning and executing a learning journey will make your work more focussed and directed. It may prevent you from wasting time by browsing for far too general information on the web, or reading all of a book when only two chapters in the middle are relevant.

Mapping your learning journey

The final tool we'll look at in our learning arsenal is an old favourite: the mind map. Mind maps are a powerful outlining technique that appeases the left brain's linear organisational fetish whilst satisfying the right brain's spatial, relational desires.

Mind maps are structured outlines in which you start with a key concept in the middle and add information around it forming a web. You can add extra relations between items, and freely use colour, illustration, size and position to highlight extra information.



I often produce mind maps for talks and articles I'm writing. Being an unashamed sad geek, I tend to produce these electronically (I used FreeMind [6] which works on Windows, Mac, and Linux). Then I can keep them under source control and version them alongside my articles.

However, many people claim that you will miss the full benefit of the mind mapping technique by doing this. The more visceral, physical act of creating a hand-written version helps you explore and retain information. By laying out the relations yourself rather than relying on a machine to typeset the 'document' for you, you consider inter-relations and build a tangible picture in your mind that will aid later recall.

Next time you're learning, why not try to record knowledge in a mind map as you go? When investigating a learning route and working out what you need to know, record your findings in a mind map.

Use mind maps to catalogue information. They are a proven, powerful tool.

Take away

It's time to wrap this journey up. There is so much more that could be said, but there isn't time or space to. If this stuff has piqued an interest for you

(I truly hope that it has – it's vital if you want to stay at the peak of your career!) then start looking into it yourself. Perhaps you could start with Andy Hunt's recent book on the subject, *Pragmatic Thinking and Learning* [7]. It's highly recommended.

Finally, then, here are some practical ideas to help you improve your learning. These are simple techniques to take away...

- Cultivate a healthy set of attitudes: take responsibility for your learning.
- Learn one programming language per year. (This is excellent advice in *The Pragmatic Programmer* that is still very valid today.)
- Scratch an itch (What are you curious about? Perhaps something 'leftfield'. Ensure that at all times you have something you're learning about that isn't directly related to your day job).
- Read at least one book every two months (Read more if you want, but set some kind of benchmark to aim for).
- Look after your learning machine – get good nutrition, and plenty of rest. Avoid stress. Have fun!
- Build mental maps as you learn.
- Try to use both sides of your brain.
- Perform deliberate practice and exercise as you learn.
- Network: actively learn from others, and seek to teach/mentor others.
- Enjoy learning. This stuff is fun.
- Apply any new knowledge cautiously.

Conclusion

You have to take responsibility for your own learning. It's not up to your employer, your state education system, an assigned mentor, or any other person.

You are in charge of your own learning. It's important to continually improve your skills to improve as a developer. And to do that you have to *learn* to learn. To make it rewarding you have to learn to *love* doing it.

Learn to live to love to learn. ■

Questions to ponder

- When were you last in a situation that required learning?
- How did you approach it?
- How successful were you?
- How quickly did you learn?
- How could you have performed better?
- Did you learn, then work, or learn as you worked?
- Which do you think is most effective?

References

- [1] *The Pragmatic Programmer*. Andrew Hunt and David Thomas (1999) ISBN-10: 020161622X
- [2] *Mindset: The New Psychology of Success*. Carol S Dweck. Ballantine Books. ISBN: 0345472322
- [3] *Self-theories: Their role in motivation, personality and development* (1999) Carol S Dweck. ISBN-10: 1841690244
- [4] <http://codekata.pragprog.com/>
- [5] 'Live to Love to Learn, Part Two'. Professionalism in Programming #60. Pete Goodliffe. In: *CVu 22.1*.
- [6] FreeMind. <http://freemind.sourceforge.net/> Other tools are available. Including pen and paper.
- [7] *Pragmatic Thinking and Learning*. Andy Hunt. ISBN-10: 1-934356-05-0

A Game of Nerve

Baron Muncharris has another tall tale.

Hail and well met Sir R----! As ever, it is my very great pleasure to invite you to join me in a glass of restorative spirits and perhaps a little sport.

Why you seem pained my friend; what ails you?

Your back, you say?

It so happens that I have in my possession a quantity of snake oil liniment prepared by the master apothecaries of the South Sea Floating City, of which I can vouchsafe efficacy.

I travelled to that remarkable city after my ill-fated expedition to the South Americas during which I discovered, and subsequently forgot, the location of the fabled City of Gold. Fortuitously, I came away from my adventure with a solid gold pillow with which I was able to secure accommodation in the city's most luxurious hostelry during its passage back to Europe.

The journey was, for the most part, a pleasant one during which I sampled the finest comestibles known to man. Had we not been set upon by the notoriously belligerent South Sea Dragon I should have considered it one of the most pleasant, not to say the most splendidly provisioned, journeys that I ever undertook.

The Dragon set upon the city on a warm Tuesday, during afternoon tea. I was sitting outside a charming bistro in the French quarter enjoying a particularly excellent slice of cake when he descended upon us with a terrible roar. I immediately leapt to the defence of the city with the closest weapon to hand; my dessert fork.

With men falling to his fiery wrath all about me, I charged him. As he opened his jaw, enveloped me in flame and roasted me alive, I leapt onto his tongue and drove my diminutive trident through his palate and into his brain, killing him instantly. If I hadn't had access to that miraculous liniment I suspect that my complexion might have been permanently affected.

I passed the several hours of my convalescence at dice with my consulting apothecary in a game that is most popular amongst their number.

Perhaps you too might appreciate such diversion from your discomfort? Splendid!


Their game consists of up to four turns casting a pair of dice and will cost you nine coins to play. At each turn, both dice must be cast and you may then elect to forfeit your remaining turns and collect their sum as your prize if you haven't the nerve to chance a greater bounty.

That loathsome student acquaintance of mine, upon hearing these rules, started squawking on about how he expected his condition to improve over the future, although, given his record, I should expect it to degrade to dribbling incomprehension in short order.

Now, take another drink and a moment to decide whether you shall play! ■

JOIN ACCU

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



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

Join ACCU to receive our bi-monthly publications *C Vu* and *Overload*. You'll also get massive discounts at the ACCU developers' conference, access to mentored developers projects, discussion forums, and the chance to participate in the organisation.

What are you waiting for?

How to join
Go to www.accu.org and click on Join ACCU

Membership types
Basic personal membership
Full personal membership
Corporate membership
Student membership

professionalism in programming
www.accu.org

cqf.com



Expand Your Mind and Career

Designed by quant expert Dr Paul Wilmott, the CQF is a practical six month-part time course that covers every gamut of quantitative finance, including derivatives, development, quantitative trading and risk management.

Find out more at cqf.com.

ENGINEERED FOR THE FINANCIAL MARKETS

Lazy Initialisation of Shared Resources

Iain Charlton looks at managing object lifetime from birth to death and beyond.

It is interesting that the later part of the lifetime of an object is often well managed, and implementations such as `auto_ptr`, `scoped_ptr`, `shared_ptr` and so on remove the need to explicitly manage deallocation when they are applied correctly.

Why then have we not focussed equally on when to allocate? The answer to this question is probably simple: we know in code when we first need an object (i.e. when a containing class is constructed), so we create it then explicitly. When we want to defer the construction of large objects or the initialisation of resources with a large time overhead, lazy initialisation is often implemented inside an object, such that allocated and initialisation only when a client requests access to that data.

Why lazy initialisation sometimes isn't enough

The code snippet in Listing 1 illustrates how the classic lazy initialisation pattern of a member `shared_ptr` is insufficient for delaying the initialisation of the object beyond its first aggregation into a client class.

In `create_log_users`, as soon as the `log` function is called on the `log_manager` object, the `log_impl` object is allocated, initialised and put into the `shared_ptr`. Ideally, the `shared_ptr` would not be initialised until the `log_user` class genuinely needed to use it, probably inside the `log` function.

Making a `shared_ptr` lazy: `lazy_ptr`

The inspiration for a `lazy_ptr` comes from `shared_ptr`. What's great about `shared_ptr` is its ability to destroy the wrapped object without the client code knowing anything of the destructor implementation. Even better, the destruction is handled by a functor, which can be bound to any appropriate function: a member or static member (public, protected or hidden), any public member of other objects visible to the creator of the `shared_ptr` or any visible free function. With that in mind, `lazy_ptr` takes a construction or resetting functor that returns a pointer to the

```
public:
    virtual void operator() (
        const std::string & ) = 0
    {
        //...
    }
};
// Log manager
class log_manager
{
public:
    log_manager()
    {
    }
    boost::shared_ptr< log > log()
    {
        if( ! m_log )
        {
            m_log.reset( new log_impl );
        }
        return m_log;
    }

private:
    boost::shared_ptr< log_impl > m_log;
};

// Log user
class log_user
{
public:
    log_user( boost::shared_ptr< log > i_log,
              const std::string & i_name )
        : m_log( i_log )
        , m_name( i_name )
    {
    }
    void output ( const std::string & i_string )
    {
        (*m_log) ( m_name );
        (*m_log) ( i_string );
    }

private:
    boost::shared_ptr< log > m_log;
    const std::string m_name;
};

void create_log_users( log_manager & i_log_man )
{
    log_user logger1( i_log_man.log(), "debug" );
    log_user logger2( i_log_man.log(), "normal" );

#ifdef _DEBUG
    logger1.output( "initialised" );
#endif
    logger2.output( "initialised" );
}
```

Listing 1 (cont'd)

Listing 1

```
#include <boost/shared_ptr.hpp>

// Abstract log base class
class log
{
public:
    virtual void operator () (
        const std::string & ) = 0;
};

// Log implementation.
class log_impl : public log
{
```

IAIN CHARLTON

Iain Charlton has a background in mechanical engineering, biomechanics R&D, and software development. He is currently writing C++ and managing projects for motion capture, biomechanical modelling and clinical reporting products. He can be contacted at iain.charlton@vicon.com.



```

#ifndef _LAZY_PTR_H_
#define _LAZY_PTR_H_

#include "lazy_ptr_impl.h"
#include <boost/shared_ptr.hpp>
#include <boost/static_assert.hpp>
#include <boost/type_traits/is_base_of.hpp>

template< typename T >
class lazy_ptr
{
public:
    // Default constructor.
    lazy_ptr()
    {
    }

    // Construct with a construction functor.
    // C's copy constructor must not throw.
    template< typename C >
    explicit lazy_ptr( C i_ctor )
    : m_impl(
        new lazy_ptr_impl_t< T, C, void(*) (T*) >(
            i_ctor,
            &lazy_ptr< T >::destroy ) )
    {
    }

    // As above with a destruction functor.
    // D's copy constructor must not throw.
    template< typename C, typename D >
    lazy_ptr( C i_ctor, D i_dtor )
    : m_impl(
        new lazy_ptr_impl_t< T, C, D >(
            i_ctor,
            i_dtor ) )
    {
    }

    // Copy construction from same type. No-throw.
    lazy_ptr( const lazy_ptr< S > & i_other )
    : m_impl( i_other.m_impl )
    {
    }

    // Copy construction from compatible type.
    // No-throw if it compiles.
    template< typename S >
    lazy_ptr( const lazy_ptr< T > & i_other )
    : m_impl( i_other.m_impl )
    {
        BOOST_STATIC_ASSERT(
            ( boost::is_base_of< T, S >::value ) );
    }

    // Assignment from same type. No-throw.
    lazy_ptr< T > & operator=(
        const lazy_ptr< T > & i_other )
    {
        lazy_ptr< T >( i_other ).swap( *this );
        return *this;
    }
}

```

contained type and takes no input arguments. The basic interface is shown in Listing 2.

Construction of a `lazy_ptr` can be default (such that the wrapped pointer is `null`), with a construction functor of arbitrary type, or paired construction and destruction functors of arbitrary type. The resetting

```

// Assignment from compatible type.
// No-throw if it compiles.
template< typename S >
lazy_ptr< S > & operator=(
    const lazy_ptr< S > & i_other )
{
    lazy_ptr< T >( i_other ).swap( *this );
    return *this;
}

// Reset the pointer.
void reset()
{
    lazy_ptr< T >().swap( *this );
}

template< typename C >
void reset( C i_ctor )
{
    lazy_ptr< T >( i_ctor ).swap( *this );
}

template< typename C, typename D >
void reset( C i_ctor, D i_dtor )
{
    lazy_ptr< T >( i_ctor, i_dtor ).swap(
        *this );
}

// Test for pointer validity.
bool valid() const
{
    return m_impl ? m_impl->ptr_valid() : false;
}

// Access to the wrapped object.
boost::shared_ptr< T > lock()
{
    return m_impl ?
        m_impl->template lock< T >() :
        boost::shared_ptr< T >();
}

// Swap. No-throw.
void swap( lazy_ptr< T > & i_other )
{
    std::swap( m_impl, i_other.m_impl );
}

private:
    // Static destroy.
    static void destroy( T * i_ptr )
    {
        delete i_ptr;
    }

    // Implementation.
    boost::shared_ptr< lazy_ptr_impl > m_impl;
    template< typename S > friend class lazy_ptr;
};

#endif // _LAZY_PTR_H_

```

functions allow pretty much the same behaviour as the constructors such that the wrapped pointer, construction and destruction functors can be replaced. Behaviour is equivalent to creating a new `lazy_ptr` and assigning it to the existing one.

The important part of the `lazy_ptr` interface is the lock function. Rather like `weak_ptr::lock`, this is the only way to access the wrapped pointer

```

#ifndef _LAZY_PTR_IMPL_H
#define _LAZY_PTR_IMPL_H

#include <boost/noncopyable.hpp>
#include <boost/shared_ptr.hpp>
#include <stdexcept>

class lazy_ptr_impl : private boost::noncopyable
{
public:
    // Access to the wrapped object.
    // Throws (see lazy_ptr_impl_t).
    template< typename T >
    boost::shared_ptr< T > lock()
    {
        void * p = private_lock();
        return *static_cast<boost::shared_ptr<T*>>(
            p );
    }

    // Test for pointer validity. No-throw.
    virtual bool pointer_valid() const = 0;

protected:
    // Default construction accessible to
    // implementing classes only.
    lazy_ptr_impl()
    {
    }

    // Destruction accessible to implementing
    // classes only.
    virtual ~lazy_ptr_impl() = 0
    {
    }

private:
    // Private lock function using opaque pointer.
    // Throws (see lazy_ptr_impl_t).
    virtual void * private_lock() = 0;
};

// Template implementation of lazy_ptr_impl.
template< typename T, typename C, typename D >
class lazy_ptr_impl_t : public lazy_ptr_impl
{
public:
    // Construct an empty implementation
    // (embedded shared_ptr null).
    lazy_ptr_impl_t( C i_ctor, D i_dtor )
    : m_ctor( i_ctor )
    , m_dtor( i_dtor )
    , m_constructed( false )
    {
    }
}

```

and will result in lazy allocation and initialisation of the wrapped pointer, returning a `shared_ptr` to it. Boolean conversion provides another `weak_ptr` and `shared_ptr`-like function, returning `false` if the wrapped pointer is `null`.

Privately, the class includes a static `destroy` function (which we will see used in the implementation to provide a default destruction functor when none is provided) and a `shared_ptr` to the implementation class. Finally, `lazy_ptr` objects of different type are friends to allow copying and assignment between compatible types.

Of course, the simplicity of this implementation is dependent on additional complexity in the `pimpl` object: an abstract base class that provides the necessary polymorphic functionality for allocating, initialising and managing the wrapped pointer (see Listing 3).

```

// Public destructor.
virtual ~ lazy_ptr_impl_t()
{
}

// Implementation of lazy_ptr_impl.
public:
    virtual bool pointer_valid() const
    {
        return m_object;
    }

private:
    // Private lock function using opaque pointer.
    // Throws std::logic_error if recursive.
    virtual void * private_lock() // throws
    {
        if( ! m_object )
        {
            if( m_constructed )
            {
                throw std::logic_error(
                    "recursive lazy_ptr construction" );
            }
            m_constructed = true;
            T * object = m_ctor();
            if( object )
            {
                m_object.reset( object, m_dtor );
            }
        }
        return &m_object;
    }

private:
    // Construction functor.
    C m_ctor;
    // Destruction functor.
    D m_dtor;
    // Recursive construction flag.
    bool m_constructed;
    // Object.
    boost::shared_ptr< T > m_object;
};

#endif // _LAZY_PTR_IMPL_H

```

Now, the template lock function on the base class uses a `static_cast` of a `void *` in order to get around templating the implementation. If we did this, implicit up-casting of the `lazy_ptr` wouldn't be as straightforward, as `lazy_ptr`s of differing (but compatible) types wouldn't be able to share the same implementation on a simple `shared_ptr`: there would have to be some kind of proxy class generated. Such an additional class would require nested reference counting (not in itself a bad thing) and an additional allocation on copy or assignment of `lazy_ptr` objects of differing type. This additional allocation is perhaps contrary to expectation and creates new opportunities to throw within the `copy` operation. In the current implementation, `copy` construction and assignment are guaranteed no-throw operations.

One notable omission from `lazy_ptr` and its implementation is the lack of an allocator template argument for construction and reset. This is not by design; rather it is in the interest of brevity. Adding such functionality is trivial, requiring the template allocator functors to be passed through to a new derivative of `lazy_ptr_impl`, which itself passes the allocator on to the encapsulated `shared_ptr`.

Example uses

Now, let's return to the initial example and use `lazy_ptr` instead of `shared_ptr` and see the differences (Listing 4).

```

#include "lazy_ptr.h"

// Abstract log base class
class log
{
public:
    virtual void operator () (
        const std::string & ) = 0;
};

// Log implementation.
class log_impl : public log
{
public:
    virtual void operator() (
        const std::string & ) = 0
    {
        //...
    }
};

// Log manager
class log_manager
{
public:
    log_manager()
    : m_log( &log_manager::create_log )
    {
    }

    lazy_ptr< log > log()
    {
        return m_log;
    }

private:
    static log_impl * create_log()
    {
        return new log_impl;
    }

    lazy_ptr< log_impl > m_log;
};

// Log user
class log_user
{
public:
    log_user( lazy_ptr< log > i_log,
             const std::string & i_name )
    : m_log( i_log )
    , m_name( i_name )
    {
    }

    void output ( const std::string & i_string )
    {
        boost::shared_ptr< log > log = m_log.lock();
        if ( log )
        {
            (*log) ( m_name );
            (*log) ( i_string );
        }
    }

private:
    lazy_ptr< log > m_log;
    const std::string m_name;
};

```

```

void create_log_users( log_manager & i_log_man )
{
    log_user logger1( i_log_man.log(), "debug" );
    log_user logger2( i_log_man.log(), "normal" );

#ifdef _DEBUG
    logger1.output( "initialised" );
#endif
    logger2.output( "initialised" );
}

```

The example code has changed very little, but with a trade in complexity in the log function initially containing the lazy initialisation to the new initialisation function `create_log`. This member doesn't need to be static, but this makes for the simplest code as it is not necessary to write or invoke the use of a binding functor such as `boost::bind` in order to use it.

Now when the code is executed, the calls to `log_manager::log` do not cause the encapsulated `log_impl` to be initialised: instead this is deferred until either of the `log_user` clients need to use their log interfaces and call `lazy_ptr::lock` to do so, again, probably within the log function.

A second class of problem can be neatly avoided using `lazy_ptr`: complex initialisation order of application resources. When we start up an application, there are frequently many resources allocated and initialised which are often inter-dependent. Listing 5 shows how a small number of application resources that depend on each other (although have no circular dependencies) can be declared such that lazy initialisation will result in the independent object being created first and the object with the most dependencies last. (Listing 5)

The order of the manager classes inside the `app_manager` is deliberately different to their preferred construction order to illustrate how the construction order of the `lazy_ptr` members is not tied to the dependency ordering. Instead, the construction order is resolved at run time when one of the `lazy_ptr` objects members are required for use, for instance when `undo_man` is called. This calls lock on the `m_undo_man` member, which internally calls lock on both the `m_log_man` and the `m_event_man` as the construction functions are called. Looking back to the implementation in `lazy_ptr_impl.h`, notice how any circular dependencies between components constructed within `lazy_ptr::lock` will throw when `private_lock` is called recursively.

```

#include "lazy_ptr.h"
#include <boost/bind.hpp>

using namespace boost;

class event_manager
{
public:
    static event_manager * create()
    {
        return new event_manager;
    }
};

class log_manager
{
public:
    static log_manager * create(
        lazy_ptr< event_manager > i_event_man )
    {
        return new log_manager(
            i_event_man.lock() );
    }
}

```



```

private:
    log_manager (
        shared_ptr< event_manager > i_event_man )
    : m_event_man( i_event_man )
    {
    }

    shared_ptr< event_manager > m_event_man;
};

class undo_manager
{
public:
    static undo_manager * create(
        lazy_ptr< log_manager > & i_log_man,
        lazy_ptr< event_manager > & i_event_man )
    {
        return new undo_manager(
            i_log_man.lock(),
            i_event_man.lock() );
    }

private:
    undo_manager(
        shared_ptr< log_manager > i_log_man,
        shared_ptr< event_manager > i_event_man )
    : m_log_man( i_log_man )
      , m_event_man( i_event_man )
    {
    }

    shared_ptr< log_manager > m_log_man;
    shared_ptr< event_manager > m_event_man;
};

class app_manager
{
public:
    app_manager()
    : m_undo_man( bind(
        &undo_manager::create,
        m_log_man,
        m_event_man ) )
      , m_log_man( bind(
        &log_manager::create,
        m_event_man ) )
      , m_event_man(
        &event_manager::create )
    {
    }

    shared_ptr< undo_manager > undo_man()
    {
        return m_undo_man.lock();
    }
    shared_ptr< log_manager > log_man()
    {
        return m_log_man.lock();
    }
    shared_ptr< event_manager > event_man()
    {
        return m_event_man.lock();
    }

private:
    lazy_ptr< undo_manager > m_undo_man;
    lazy_ptr< log_manager > m_log_man;
    lazy_ptr< event_manager > m_event_man;
};

```

An encouraging aspect of this code design is that it is a very easy re-factor from ordinary `shared_ptr` members: the static create functions must be added and the previous instantiation of the `shared_ptr` members in the `app_manager` replaced with `lazy_ptr` instantiations instead. Making the constructors of the aggregated classes private is of course optional.

One caveat to this kind of coding using `lazy_ptr` is that we should still think about initialisation ordering, rather than simply forget about it and defer it until run time. As undesirable as it is, two stage initialisation is occasionally hard to avoid, and this `lazy_ptr` pattern offers little scope for improvement, if anything, making the problem extremely difficult to resolve.

Managing complete lifetime with weak_lazy_ptr

Once we have a working `lazy_ptr` dotted around in our code, we may start to hanker for the other lifetime management tools offered in `boost` and `tr1`: the weak referencing pointer. This would then allow us to hold a reference to a resource that can be in one of three states:

1. Not yet constructed.
2. Constructed.
3. Destroyed.

This is rather an attractive prospect, which is why `weak_ptr` exists. With `lazy_ptr` using `shared_ptr` internally to reference its implementation, this is an easy new class to implement by replacing the `shared_ptr` with a `weak_ptr` as shown in `weak_lazy_ptr.h` and `weak_lazy_ptr.hpp`. (See Listing 6.)

```

#ifndef WEAK_LAZY_PTR_H
#define WEAK_LAZY_PTR_H

#include "lazy_ptr.h"
#include <boost/weak_ptr.hpp>
#include <boost/static_assert.hpp>
#include <boost/type_traits/is_base_of.hpp>

template< typename T >
class weak_lazy_ptr
{
public:
    // Constructors.
    weak_lazy_ptr()
    {
    }

    // Copy construction from same type. No-throw.
    weak_lazy_ptr(
        const weak_lazy_ptr< T > & i_other )
    : m_impl( i_other.m_impl )
    {
    }

    weak_lazy_ptr(
        const lazy_ptr< T > & i_other )
    : m_impl( i_other.m_impl )
    {
    }

    // Copy construction from compatible type.
    // No-throw if it compiles.
    template< typename S >
    weak_lazy_ptr(
        const weak_lazy_ptr< S > & i_other )
    {
        BOOST_STATIC_ASSERT(
            boost::is_base_of< T, S >::value );
    }
};

```

```

template< typename S >
weak_lazy_ptr(
    const lazy_ptr< S > & i_other )
: m_impl( i_other.m_impl )
{
    BOOST_STATIC_ASSERT(
        ( boost::is_base_of< T, S >::value ) );
}

// Assignment from same type. No-throw.
weak_lazy_ptr< T > & operator=(
    const weak_lazy_ptr < T > & i_other )
{
    weak_lazy_ptr< T >( i_other ).swap( *this );
    return *this;
}

weak_lazy_ptr< T > & operator=(
    const lazy_ptr < T > & i_other )
{
    weak_lazy_ptr< T >( i_other ).swap( *this );
    return *this;
}

// Assignment from compatible type.
// No-throw if it compiles.
template< typename S >
weak_lazy_ptr< S > & operator=(
    const weak_lazy_ptr < S > & i_other )
{
    BOOST_STATIC_ASSERT(
        ( boost::is_base_of< T, S >::value ) );
    weak_lazy_ptr< T >( i_other ).swap( *this );
    return *this;
}

template< typename S >
weak_lazy_ptr< S > & operator=(
    const lazy_ptr < S > & i_other )
{
    BOOST_STATIC_ASSERT(
        ( boost::is_base_of< T, S >::value ) );
    weak_lazy_ptr< T >( i_other ).swap( *this );
    return *this;
}

```

In order for `weak_lazy_ptr` to work, it must be declared a friend of `lazy_ptr`, as it requires access to the private, shared implementation.

Armed with `weak_lazy_ptr`, we can take weak references to an object that has not yet been initialised such that initialisation can safely fail if all strong `lazy_ptr` references go out of scope. There is a limitation of this design, in that strong `shared_ptr` references to the initialised object will not keep a `weak_lazy_ptr` 'alive', as they reference the encapsulated object, rather than the shared implementation. In order to design this into the weak referencing system, `shared_ptr` and `lazy_ptr` must share implementations, meaning some modifications to boost headers. These changes are by no means difficult, but for many may be undesirable. For those using `shared_ptr` and `weak_ptr` from an STL implementing TR1, this is even less likely to be a favourable option.

Conclusions

Although lazy initialisation is a relatively infrequently used design pattern (compared to the factory pattern or RAII), providing a mechanism for lazy allocation of shared resources is a useful tool. What's more, the implementation presented here makes porting code with heavy use of `shared_ptr` and `weak_ptr` very easy.

The overhead of using `lazy_ptr` is marginal, compared to `shared_ptr`. Creating the object initially requires a single heap

```

// Test for pointer validity.
operator bool() const
{
    boost::shared_ptr< lazy_ptr_impl > impl =
        m_impl.lock();
    return impl ? impl->pointer_valid() : false;
}

// Access to the wrapped object.
boost::shared_ptr< T > lock()
{
    boost::shared_ptr< lazy_ptr_impl > impl =
        m_impl.lock();
    return impl ?
        impl->template lock< T >() :
        boost::shared_ptr< T >();
}

// Swap. No-throw.
void swap( weak_lazy_ptr< T > & i_other )
{
    std::swap( m_impl, i_other.m_impl );
}

private:
    // Implementation.
    boost::weak_ptr< lazy_ptr_impl > m_impl;

    template< typename S >
    friend class weak_lazy_ptr;
};

#endif // _WEAK_LAZY_PTR_H_

```

allocation in addition to the construction of a `shared_ptr` (which itself incurs two heap allocations) and subsequent copies incur a `shared_ptr` copy (two straight pointer copies). Using `weak_lazy_ptr` is only slightly more costly than using `weak_ptr`, with no additional allocations or copies taking place other than of the encapsulated `weak_ptr` itself.

The marginal downside to the simple implementation of `lazy_ptr` presented here is the weakness (no pun intended) of the behaviour when only shared and weak references exist: any `weak_lazy_ptr` references will cease to be valid when the last `lazy_ptr` referencing the same memory goes out of scope, rather than any `shared_ptr` references taken from `lazy_ptr::lock` keeping them alive. This deficiency is however one of behaviour, rather than instability and can be overcome by careful design of client code.

The example uses of `lazy_ptr` presented here are by no means exhaustive and it is likely that there are usage patterns beyond the scope for which the class was initially designed. ■

Acknowledgements

This article and the code presented were checked and critiqued by Nick Bullock, who contributed some valuable ideas to the design of the `lazy_ptr` and the `weak_lazy_ptr`.

On a Game of Strategy

The Baron's student acquaintance performs his analysis.

The Baron's game consists of taking turns to place coins on empty squares on a chess board until each player has placed three coins.

During play, each square on the board is considered to be controlled by the player who has the closest coin, with the distance being measured by the least number of moves required by a chess king piece to move from the one to the other. After the three turns, the first player is deemed the winner if he controls more squares than the second and the second player is deemed the winner otherwise.

I was immediately struck by the observation that assignment of control of squares is equivalent to constructing a Voronoi diagram, being a tessellation of the plane into regions each containing all those points closest to one of a given set of points, albeit in this case a discrete rather than a more familiar continuous one.

A noteworthy property of Voronoi diagrams is that each is the dual of a Delaunay triangulation of the same set of points. These are the uniquely defined sets of triangles whose vertices coincide with the given set of points such that every triangle can be circumscribed with a circle which contains no points other than its vertices or, in rare circumstances, those lying along its edges.

Indeed, one of the simplest schemes for drawing a Voronoi diagram is to first construct a Delaunay triangulation and to then figure its dual, or in other words to construct those irregular polygons that result from connecting the centres of the circumscribed circles of neighbouring triangles.

Unfortunately neither I nor my fellow students could fathom how this observation might help in forming an effective strategy for the playing of this game. Furthermore, our initial attempts to form a strategy met with little success. We therefore resolved to play as many games as possible in order to develop some sense of how the game might unfold.

We were able to somewhat reduce the immense scope of this endeavour with the realisation that the first move need not range over the entire board. By considering symmetries of rotation and reflection, we concluded that we need only play on the first four ranks and furthermore on the first rank, just the first file, on the second the first two, on the third the first three and on the fourth the first four.

So, armed with an amply supply of that splendid new fortifying beverage, coffee, we spent a sleepless weekend furiously playing the game and I can report that at no point did we find a sequence of moves by which the first player could ensure victory.

I should therefore have advised Sir R----- to have accepted the Baron's challenge, although only if he were willing to think very carefully about the consequences of his moves. ■

The figures

Figure 1 shows a Delaunay triangulation with its circumscribed circles.

Figure 2 shows that connecting the centres of the circles yields the Voronoi diagram.

Figure 3 is the set of first moves.

The listings

Listing 1: Constructing a sequence of trial moves.

Listing 2: Deciding whether to ignore a first move.

Listing 3: A recursive maximin search for the best score.

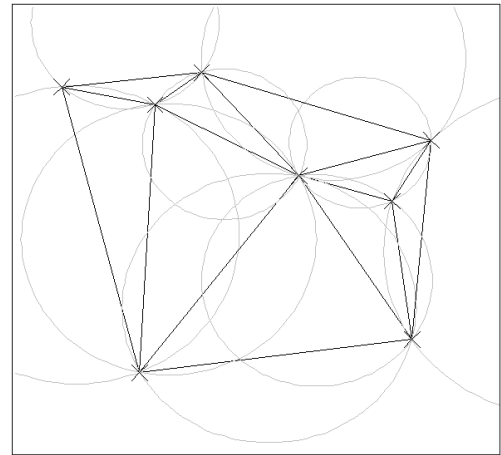


Figure 1

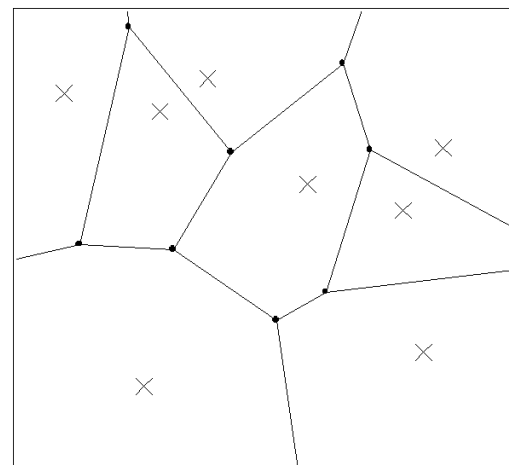


Figure 2

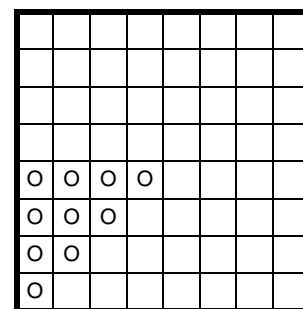


Figure 3

```
std::vector<position>
make_positions()
{
    std::vector<position> positions(64);
    for(size_t rank=0;rank!=8;++rank)
    {
        for(size_t file=0;file!=8;++file)
        {
            positions[rank*8+file] = position(
                rank, file);
        }
    }
    return positions;
}
```

Listing 1

Desert Island Books

Paul Grenyer drops Pete Goodliffe on the island.

While I was speaking aloud about ways to describe Pete Goodliffe, Jez Higgins suggested I use a recent comment from accu-general, 'blond, balding, barefoot'. While accurate these aren't a particularly nice way to describe Pete, although barefoot is at least accurate and not disparaging, like the other two.

When I think of Pete four things instantly spring to mind, his friendly personality, his bare feet, Cambridge, and curry. I have had many a curry and more than a few beers with Pete Goodliffe, both in Oxford and on occasion in Cambridge. When I think about Pete technically, his sound and solid technical knowledge and advice stand out. He has a lot of experience in software development and not only does he make it work for him to generate great software, he is always happy to help others and pass on the knowledge where he can (sorry Pete, I won't mention the book).

I very narrowly missed out on working with Pete a few years ago, which was a shame because I would have learnt a lot.

Pete Goodliffe

So here I am, stuck on a desert island with nothing but the birds for company. Still, it could be worse; at least the conversation won't get too geeky. I can't help but wonder what kind of calamitous event catapulted

me from my secure (if dull) office chair to this remote tropical location. Probably one of those stories that no one will ever believe back home, but it almost certainly involved a few beers, an ill-placed wager, and a temperamental particle physicist. Curse you CERN and Speyside!

At least I've got the decent weather. Let's just hope global warming doesn't shrink my new island home even further. Perhaps this stack of improbably placed books would provide me a small tower to sit on if the tide does indeed rise.

For my company on this forsaken isle, the gods of Fate and Calamity [PJG: I've never been described like that before!] have seen fit to bestow upon me four programming books, one novel, and two albums.

The computer books seem sadistic, cruel and somewhat ironic, as they haven't left me a computer [PJG: You didn't ask!]. If they had, I'd ram it full of e-books, anyway. At least I'm good for toilet paper for a while. The CDs are just plain torture, as the swines didn't leave me a CD player [PJG: See previous comment]. I'll have to fashion one from a coconut tree and an albatross. Or just whistle.

So that means it's only the novel that's useful, then. Better pick a big one...

On a Game of Strategy (continued)

Listing 2

```
bool
ignore_first_move(const position &pos)
{
    return pos.rank<4 && pos.file<=pos.rank;
}
```

Listing 3

```
std::pair<size_t, size_t>
best_score(size_t move,
           std::vector<position> &positions,
           std::vector<position> &first,
           std::vector<position> &second)
{
    if(move==3) return score(first, second);
    std::pair<size_t, size_t> first_score(0, 64);
    for(size_t i=0; i!=positions.size(); ++i)
    {
        if(move!=0 ||
           !ignore_first_move(positions[i]))
        {
            std::swap(positions[i], positions.back());
            first.push_back(positions.back());
            positions.pop_back();
            std::pair<size_t, size_t>
                second_score(64, 0);
            for(size_t j=0; j!=positions.size(); ++j)
            {
                std::swap(positions[j],
                           positions.back());
                second.push_back(positions.back());
                positions.pop_back();
                std::pair<size_t, size_t> score =
                    best_score(move+1, positions,
                               first, second);
```

Listing 3 (cont'd)

```
        if(long(score.second)-long(score.first) >
           long(second_score.second)-
           long(second_score.first))
        {
            second_score = score;
        }
        positions.push_back(second.back());
        second.pop_back();
        std::swap(positions.back(),
                  positions[j]);
    }
    if(long(second_score.first)-
       long(second_score.second) >
       long(first_score.first)-
       long(first_score.second))
    {
        first_score = second_score;
    }
    positions.push_back(first.back());
    first.pop_back();
    std::swap(positions.back(), positions[i]);
}
return first_score;
}
std::pair<size_t, size_t>
best_score()
{
    return best_score(0, make_positions(),
                     std::vector<position>(),
                     std::vector<position>());
}
```

So what computer books should I chose? I could select four books I have always wanted to read, but never got round to buying. I could select four random books on a whim that might be interesting. But given that I won't be able to fiddle with a computer any more (unless I find myself a convenient passing rescue party), I may as well select four classic books that have enthused me in days gone by. Books that will help me reminisce about the good old days when I had hair, when keys were rubbery, and when computers had rainbows on them.

And that is indeed my first choice. I'll re-read *First Steps with Your ZX Spectrum* by Carolyn Hughes. I haven't picked this book up in about 30 years, but it was my first ever programming book, and the thing that first gave me a passion for programming. I digested the entire book before I ever got near a computer. It has immense personal significance to me. And it also had pretty cartoon pictures of computers telling you what to do. I've not read a book like it since.



In a fit of nostalgia, I recently bought that same book on Amazon. For three pence. Seriously. Three pence. The postage was two orders of magnitude more than the book itself. That really does show how the value of knowledge decreases over time.

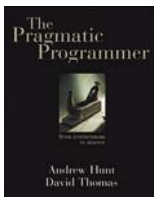
The next book providing this stranded programmer with a stroll down memory lane is Booch's classic *Object Oriented Analysis and Design With Applications*. I may be developing a theme here, as I recall this book also had a number of interesting cartoons in it. But that's not why I'm choosing it. I first encountered OOAD at university and it provided an incredibly clear, well reasoned and enjoyable overview of quality design techniques and the application of OO principles to software design. It was a genuinely great introduction to the philosophy of software design. It is also unusual, being one of the minority of hard-cover programming books I've ever owned. So it might be useful for hitting wild animals with.

This book pre-dates the quadrilateral joys of UML, and I always loved Booch's OO diagramming style. He represented classes by clouds; simply because they were easier to draw on the back of a napkin. The man's a genius.

The Pragmatic Programmer by Andy Hunt and Dave Thomas will be my third book. It's a wonderful clear, entertaining, and motivating discourse of the practice of programming from a personal and social standpoint. It's the kind of book that I love to read, and one of the few books that you can read over and over again, because you actually want to. It's full of sage, refreshing advice.

However, it could have been vastly improved with a few carefully placed cartoons.

My last techie book is a hard call. Of the numerous books that have challenged and/or aided me in my career I'd struggle to chose just one. I could spend some serious



time digesting a hardcore C++ tome, like Bjarne's *TCPPL*, or any of Meyer's or Sutter's excellent C++ references. I could pick up a development process book, like Beck's *eXtreme Programming Explained* (I remember that one being a very fresh read when I first opened it). Or I could drill into any of a number of design books, like the classic (and painfully obvious) Gang of Four tome or a number of later pattern languages. Perhaps something more recent would be a good companion like Michael Feathers' legacy code book.

I'm stuck.

So I think I'll select Sam's *Teach Yourself C++ in 24 Hours* and be done with it. It's bound to be wonderful nonsense, and is certain to produce an emotional response. Can't be any worse than a Schildt book, can it? I think I'll need the laughs.

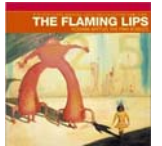


Now to ship those books securely to my remote prison I'll need to pack them carefully. I wonder if anyone will notice if I wrap them in a stack of ACCU magazines?

Presuming that I'm supplied with standard desert island accoutrements such as a copy of the Bible and the complete works of Shakespeare (perhaps with an accompanying infinite number of monkeys on an infinite number of typewriters – hopefully on their own island) then my choice of novel will be a collection of C.S. Lewis' *Chronicles of Narnia*. I've always had a soft spot for these books (as with much of Lewis' other writing). I love the Christian symbolism in them, and have started getting into them with my children, so these books would bring back a selection of old and new memories as I'm stranded here.



It just remains for me to make the choice of two albums. I'm an avid music lover with a wide and somewhat eclectic taste. However, should my soundtrack for a tropical adventure be something relaxing like a Sigur Ross album, something by Lamb, or Air's chilled *Moon Safari*, or perhaps something more upbeat like a Kings of



Leon or Killers album? My choice will instead be the awesome Flaming Lip's masterwork *Yoshimi Battles the Pink Robots*, and Delirious' *Mezamorphis*. Both of these are epic, dense, layered, meaningful and rewarding listens.

So that's it. Spare a thought for me marooned here, and pray I don't suffer from a coconut allergy or sun stroke.



Next issue: Phil Bass

What's it all about?

Desert Island Disks is one of Radio 4's most popular and enduring programmes. The format is simple: each week a guest is invited to choose the eight records they would take with them to a desert island (<http://www.bbc.co.uk/radio4/factual/desertislanddiscs.shtml>).

The format of 'Desert Island Books' is slightly different from the Radio 4 show. You choose about five books, one of which must be a novel, and up to two albums. Some people even throw in the odd film. Quite a few ACCUers have chosen their Desert Island Books to date and there are plenty more to go.

The rules aren't too strict but the programming books must have made a big impact on your programming life or be ones that you would take to a desert island. The inclusion of a novel and a couple of albums helps us to learn a little more about you. The ACCU has some amazing personalities and Desert Island Books has proved we only scratch the surface most of the time.

Each issue of CVU will have someone different. If you would like to share your Desert Island Books please email me: paul.grenyer@gmail.com.

because good code matters

ACCU

PROFESSIONALISM IN PROGRAMMING

www.accu.org

Code Critique Competition 63

Set and collated by Roger Orr.



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

Last issue's code

A new version of the C++ standard is nearing finalisation and is partly implemented in both Microsoft's Visual Studio 2010 compiler (out in beta) and the latest versions of g++ (also available on cygwin for Windows users). So I thought it was a good time to present a code sample trying to use one of the new standard libraries: `<regex>`. I found some example code implementing a minimal subset of the well-known `grep` command that was originally written to use the boost `regex.hpp` library and converted it to use the new standard library. I was expecting this to be a simple matter of changing the include file and some namespace names, but to my surprise the code didn't work correctly on Visual Studio 2010; it compiled cleanly and at first appeared to run successfully but when I used with the `-i` option (for a case insensitive search) the program failed with a runtime error:

```
C:>grep -i test
```

```
Error: regular expression error
```

So this issue's code is slightly unusual in that I'm also providing part of the `<regex>` header to help you identify the problem. As usual, please also use the opportunity to comment on other parts of the code.

Listing 1

```
// define USE_BOOST for the old way
#ifdef USE_BOOST
#include <boost/regex.hpp>
using namespace boost;
#else
#include <regex>
#if defined(_MSC_VER) && (_MSC_VER <= 1500)
// Visual Studio 2008 has it in 'tr1'
using namespace std::tr1;
#else
using namespace std;
#endif
#endif
#include <map>
#include <stdio.h>
#include <stdlib.h>
static const int MAX_LINE_LEN = 65536;
std::map<std::string, bool> option;
void usage()
{
    printf( "Usage: grep [-i -l] <pattern>\n");
    printf( "Options:\n");
    printf( "-i      case insensitive\n");
    printf( "-l      line numbers\n");
}
int process(regex expbuf)
{
    char buff[ MAX_LINE_LEN + 1 ];
    long lineno = 0;
    /* Start reading the file */
    while (fgets( buff, MAX_LINE_LEN, stdin))
    {
        lineno++;
        bool rc = regex_search(buff, expbuf);
        if (rc)
        {
```

Critiques

Balog Pal <pasa@lib.hu>

As usual, I start with a fast scan of the code to collect suspicious things, and analyze them later to be good or bad.

```
        if (option["l"])
            printf("%li:", lineno);
        printf(buff);
    }
}
return 0;
}
int main(int argc, char **argv)
{
    regex expbuf;
    int cflags = regex_constants::ECMAScript;
    if ( argc <= 1)
        usage(), exit(1);
    argv++;
    argc--;
    if (**argv == '-')
    {
        switch (**+argv)
        {
            case 'i':
            case 'l':
                option[*argv]++;
                break;
            default:
                printf( "-i or -l expected\n");
                exit(1);
        }
        argv++;
        argc--;
    }
    try
    {
        if (option["i"])
        {
            cflags |= regex_constants::icase;
        }
        expbuf.assign(argv[0], cflags);
    }
    catch (std::exception & ex)
    {
        fprintf(stderr, "Error: %s\n",
            ex.what());
        exit(1);
    }
    return process(expbuf);
}
```

Listing 1 (cont'd)

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



```

enum option_type
{
    ECMAScript = 0x01,
    // ...
    icode = 0x0100,
    // ...
};
template<class charT,
        class traits = regex_traits<charT> >
        class basic_regex
    {
    //...
    typedef option_type flag_type;
    // ...
    basic_regex& assign(const basic_regex& that);
    basic_regex& assign(const charT *ptr,
                       flag_type f = regex_constants::ECMAScript);
    basic_regex& assign(const charT *ptr,
                       size_t len,
                       flag_type f = regex_constants::ECMAScript);
    template<class s_traits, class A>
    basic_regex& assign(
        const basic_string<charT, s_traits, A>& s,
        flag_type f = regex_constants::ECMAScript);
    template<class InputIterator>
    basic_regex& assign(InputIterator first,
                      InputIterator last,
                      flag_type f = regex_constants::ECMAScript);
    // ...
    };
    typedef basic_regex<char> regex;
    template<class charT, class traits>
    bool regex_search(const charT *s,
                     const basic_regex<charT, traits>& re,
                     regex_constants::match_flag_type f =
                         regex_constants::match_default);

```

- **MAX_LINE_LEN** constant with all-caps and excess ‘static’ predicts a likely keyhole problem [ED: see <http://www.aristeia.com/TKP/>]
- use of `printf()`
- `process` gets `regex` by value
- C-style array/fixed size buffer
- use of `fgets`
- `cflags` typed `int` initialized with something from another namespace
- juggling with `++` and `*s` processing `argv`
- postincrements/decrements
- `cflags` (type `int!`) used with binary `or`
- `argv` not checked after option elimination
- `option` uses a `string/bool` map and access like `["i"]` smells overkill
- multiple calls to `exit(1)` from different situations

The text says problem encountered using the “i” option. The only difference this creates is to set a different value in `cflags`, so let’s follow that first. `cflags` is fortunately used only once, in call to `assign`. Assign has a bunch of overloads, which should be ours? Let’s call them #1 to #5 in the order they are in the listing.

We pass two arguments, with types (`char *`, `int`).

- #1 has one arg, out.
- #2 wants `enum` as second arg, out.
- #3 can convert ours to `const char *`, `size_t`, and add the default as third, selectable.

- #4 wants `enum` as second arg, out.
- #5 wants the same type for first two args, will not work for either `char *` or `int`, out.

So #3 will be called, passing our string as `s`, our or-ed special value as `len`, and adding the default `cflags`. Hardly what we wanted, when #2 looks like taking a C-style string and a `flag_type` that seems to use the same constants our `cflags` did. Though implementation of the functions is not provided it is a safe bet that `len` in #3 is used to tell the length of the string passed (instead of the 0-termination), sending over 0x100 with some short string in `argv` could easily lead to a crash or other UB, the reported error is just luck, the library probably reads the input from its start and throws at the early ‘\0’ encountered, never looking past that. The ‘good’ behavior in the optionless case is more a mystery, I’d predict it should only use the first letter of the pattern ignoring the rest. [ED: this is correct]

The solution shall be obvious: `cflags` shall have to be `flag_type`, then the correct overload is called with the correct values and we have some chance to work. Certainly `!=` will not work unless the library provides an overloaded operator – we need a cast, or better a function that adds a flag to the passed `enum`, doing the necessary cast internally.

Now let’s look for the other items in the list. (I reordered the bullets by increasing severity).

- **map**: The option map could be `map<char, bool>` with the same effect and similar use just changing double quotes to single. Hardly a big deal, but why waste resources. I don’t recall `grep` having multiletter options. [ED: some more recent versions of `grep` do]
- **exit(1)**: I personally don’t like calling `exit()` at all, and try to return from `main`, to have full cleanup of everything. (Including locals in `main()`). In this code I see little reason to not use `return` instead. If numbers are returned, why not return different values for different problems to provide extra info? Or if we stick with binary, why not use `EXIT_SUCCESS` and `EXIT_FAILURE`? This again is no big deal, but could make little better.
- **regex** by value: hopefully if compiles, the class handles the copy in a sensible way. I’d pass by reference.
- **argv** processing: the code ‘eliminates’ the filename and options to leave just the pattern. The `argc` check is only at front, so if we use a command line with options only, `argv` will be at trailer, and pass a null pointer to assign. That is likely illegal. Even if legal not nice, and doesn’t fit the logic that should print a command line error indication. The development probably got abruptly stopped due to the posted problem, and loops were planned to collect multiple options.

I don’t like using `*` with `argv` in general, and a thing like `+++` would not pass my review for sure. Especially as it is not even needed. A cleaner way is to refer `argv[0]` to the item, and look for ‘-’ in `argv[0][0]` and option character in `argv[0][1]` (or later with an index variable).

Incrementing `argv` could be questioned too, but it is quite idiomatic. Though preincrement is the thing we expect.

The option in the map is set by `++` too. It is defined and works for `bool`, but I’d use `= true` as that is more obvious.

As summary this block is better rephrased, and must check `argc` an extra time.

- fixed `buff[]` and `fgets`: overrun-wise the code looks okay, though passes discrepant values, there is no point to allocate an extra char if it is not used ever. `fgets` is okay 0-termination-wise too. But if the input line is too long, the code will not detect it, and go ahead incrementing `lineno`. The output will be like if the program wrapped the lines at the `MAX` boundary, and counted accordingly. The provided numbers will not match the ‘real’ numbers. The problem is not trivial to deal with, but one sensible solution is to emit a warning on long lines, that just part was checked, then read on until newline discarding input, so keeping the count correct. Reading unlimited amount into memory could be wrong.

- **printf**: in usage text is output using **printf**, without params, that is an overkill. We use **printf** for formatted output, and with great care to match the format string and params. For other situations it is better avoided.

And we reach the big killer with it in process.

```
printf(buff);
```

That is what we never ever do. Or at least shouldn't, in order to avoid those security patches some companies send us monthly – or the vulnerabilities they are patching. The first param of **printf** is the format string. So that is what shall be there. Stuff that comes from outside, that is not in our control, is not healthy food in that field. The text read from the stream can have % params, and those will be processed, using trash found on stack. With luck only causing a core dump immediately, but could do anything really. Many 'run arbitrary code' attacks use exactly this form of code.

What is the saddest part of the story, the text claims the code 'compiled cleanly' on two compilers. I know for sure, that g++, even old versions have checks on **printf** family functions, and can warn about passing non-literal as format. So clean compilation was obtained by not using possible warnings. [ED: In my experience only experts know how to turn on g++ warnings! I used **-Wall** which didn't catch it.]

Commentary

I was puzzled by the main problem with this code – the **-i** option not working – as the same code *does* work with the regex version of boost. And, I thought, the new standard regular expressions were based on the boost implementation. I had even reviewed that chapter in the first draft of the new standard.

I discovered the problem was a combination of two issues: 'wobble room' in the draft standard and a piece of 'helpfulness' in the implementation.

In the draft C++0x standard **syntax_option_type** was allowed to be either an **enum** or an integer type. The original boost regex library used an integer type and so **cflags**, as an **int**, matched the second parameter (**flag_type**) of the second **assign** method.

In Microsoft's implementation **syntax_option_type** was an **enum**, and so, as Pal stated, the **cflags** argument does not match the second parameter for this **assign** method. So I would have expected, from the draft standard, to get a compilation error at this point.

However Microsoft's implementation added a **default value** for the third parameter of the third overload of **assign** – presumably trying to be helpful for the case when the value supplied would be usual regex default. However this means that the two arguments supplied (**const char *** and **int**) will match the parameters **const char *** and **size_t**.

This is an example of the problems that can be caused by conversions between enumeration and integer values, coupled with the dangers of having both default arguments and overloads.

Note: Since the critique was published the C++0x standard has been revised again (it is now in 'final committee draft' (FCD) state) and the 'wobble room' of allowing the **syntax_option_type** to be an integer type has been removed. This will help to ensure consistent behaviour across implementations, but of course won't help this particular problem; I hope Microsoft remove the default argument in a subsequent release of their compiler.

The Winner of CC 62

As Pal was the only entrant (again!) I am awarding him the prize. Come on the rest of you – I'm sure you all have things to say about this code!

Code Critique 63

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

We have a little piece of straight C code this time.

```
/**
 * Program to read first_name last_name
 * and print sorted. See also cc50
 */
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
typedef
int (*compar)(const void*, const void*);
typedef struct name
{
    char *ln; // last name
    char *fn; // first name
} name;
// order last names before first names
int compare(name *n1, name *n2)
{
    int t = strcmp(n1->ln, n2->ln);
    if (!t)
        t = strcmp(n1->fn, n2->fn);
    return t;
}
int main()
{
    char line[80];
    struct name *names = NULL;
    int n = 0;
    int i;
    while (gets(line))
    {
        struct name *nm;
        char *p;
        n = n + 1;
        names = (name*)
            realloc(names, n * sizeof(name));
        nm = names + n - 1;
        p = strstr(line, " ");
        nm->fn = (char*)malloc(p - line);
        strncpy(nm->fn, line, p - line);
        nm->ln = strdup(p+1);
    }
    qsort(names, sizeof(name),
        n, (compar)compare);
    for (i = 0; i < n; i++)
    {
        printf("%i: %s %s\n",
            i, names[i].fn, names[i].ln);
    }
}
```

The following program is designed to sort a list of names, but does something very strange. Here is what I get when I try to run it:

```
C:> cc63
Anthony Hopkins
John Guilgud
Michael Caine
Charlie Chaplin
^Z
0: Caine Anthony
1: Charlie Chaplin
2: Hopkins Guilgud
3: Michael John?O3
```

Help!

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 helps overseas members who typically get the magazine much later than members in the UK and Europe.

Regional Meetings

Chris Oldwood enjoys a night out data mapping and generating code.

ACCU London March 2010

Paul Grenyer: Enterprise Web Application Development in Java with AJAX and ORMs

March 2010 saw the London branch of the ACCU meet to listen to Paul Grenyer talk about Enterprise Web Application Development. After a brief biography and definition of what an Enterprise Application is he leapt into the heart of the topic – writing code to communicate with a database. Unlike traditional talks where code is illustrated with slides, he decided to go ‘live’ and use Eclipse, a shell and later a web browser as his presentation tools. This is always a brave move but aside from one minor glitch it went very smoothly and I think everyone appreciated seeing real code running against a real database.

The early part of the presentation looked at techniques for mapping objects to relational databases. His first example was a classic manual SQL approach where the developer writes all the code to invoke the query, extract the result data and clean up. Anyone who attended his previous talk about resource management in database access code will know the lengths required for this approach. He then switched to using Hibernate, one of the most popular O/RM tools, to take much of the legwork out of this affair. This was refined over subsequent iterations to slowly reduce the amount of client code at the expense of a more complex class hierarchy using the Data Mapper and Registry patterns. The payoff though was the ability to Unit Test and Integration Test the Database Access layer.

The Integration Testing aspect was of particular interest to me as I had been working on this issue myself, albeit in C#, and was glad to see I was following a similar approach. Paul had MySQL running locally and the technique involved creating a database from scratch that contained just the tables and objects – no data. The tests would then be responsible for adding and removing any data; he used the natural rollback feature of transactions to perform the clean up. His test code essentially performed a round-trip by writing a new object and then reading it back again, being careful to ensure he was really hitting the database and not being served up his previous object from some internal cache.

The latter part of his presentation looked at the GUI side of things and the Google Web Toolkit (GWT) in particular. This toolkit allows you to write your presentation code in Java as well - turning it into JavaScript to run in the client's browser – so that you can use the same code on both client and server. The server-side logic was exposed as a web service and invoked from the client via AJAX. Having only done classic ASP in the past I found it quite impressive to see what could be achieved with such little code and the murmurs from the audience seemed to concur. His parting gift was to skin the UI with a light dash of CSS to show how easily it could be tarted up without touching the code.

As always we moved from the conference room to the pub across the road to continue any discussions, although I probably monopolised the speaker as I had a number of questions about his Integration Testing methods. His talk may have had Java in the title, but as is often the case many of the concepts and techniques he illustrated apply equally well to other similar platforms such as C#/.Net.

Inspirational (P)articles

Frances Buontempo inspires us once more.

A kata is a martial arts exercise used to practise a sequence of moves over and over to perfect it, and allow it to become second nature. Recently I saw a blog [1] concerning code katas. Though I hadn't really paid attention the first time I read about Uncle Bob's bowling kata [2], I got it this time. Find a small problem to solve and write the code for it every day until it becomes second nature. For example, in interviews there are certain questions that crop up again and again that I used to be able to solve immediately, but it's been a while and I've forgotten all the details, so I tend to come out with a hand waving response which is close but doesn't get all the fine details. Perhaps if I practise them again and again, they will become second nature, and that will stop the brief panic of thinking I used to be able to do this, but it's all gone vague. In addition to being able to code a specific algorithm, this means I will actually write some code on a regular basis, something which doesn't always happen in my day job, even though I'm a programmer. You can use the same solution every time, or find different solutions. You can concentrate on the tests. You will discover things that are holding you up and find ways to improve. You will learn your development tools better. For example, I keep catching myself using the mouse, which slows me down. This line of thought provoked me into finding a list of keyboard shortcuts for the IDE I use at work, which cheered me up.

Initially, the idea of writing code to do the same thing over and over again seemed odd. However a chess player, musician, racing driver and any other hobbyist or professional you can think of will practise, possibly every day. They continue to practise, even when they are the world leader, or quite good. I personally feel as though I am attempting to cruise on what I've learnt so far, which is no fun at all. I want to get better. I've seen several puzzles, problems at work, and interview questions in my time: rather than saying, 'Seen it', I will practise some of these as katas and get better. Hopefully, this will inspire you to practise too. Reference [3] gives a series of thoughts and links concerning code katas, and there are many other sites out there with lists of suitable kata problems.

References

- [1] <http://blog.objectmentor.com/articles/2009/11/21/whats-all-this-nonsense-about-katas>
- [2] <http://butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata>
- [3] <http://codekata.pragprog.com/>

If you have an inspiration you want to share, to encourage the readers of CVu, please send it to Frances.Buontempo@gmail.com.

A Conference Retrospective

Paul Grenyer reflects on his time at this year's conference.

This year's ACCU conference took place in Oxford, UK in April. It was another excellent conference, fully packed with technical sessions, seminars, lightning talks, birds-of-a-feather sessions, social events and more. In this article, Paul Grenyer provides a summary of his conference experiences. If you weren't able to attend, this will hopefully give you a flavour of what you missed. Photos courtesy of Anne-Jayne Metcalfe.

Wednesday

The Roots of Scrum: How the Japanese Lean Experience Changed Global Software Development

Jeff Sutherland

Synopsis: Dr. Jeff Sutherland covers the history of Scrum from its inception through his participation with Ken Schwaber in rolling out Scrum to industry, to its impact on Google, Microsoft, Yahoo, Oracle, Siemens, Philips, GE, and thousands of other companies. He describes the relationship of Scrum to experience at Bell Labs, MIT, iRobot, and the Grameen Bank, his communications with Kent Beck who used Scrum experience to help create XP, and how the Agile Manifesto accelerated Scrum adoption. Most important, he concludes by describing how team spirit is at the root of product innovation and hyperproductive teams and how that spirit can transform organizations.

Just as last year with Robert Martin's key note, the 2010 ACCU conference got off to a storming start with this key note from Jeff Sutherland. I wish I could have got my entire team, the managers and to some extent the board to see. It contained good explanations about so many things about Scrum that I have been trying to explain for some weeks now.

Sutherland, one of the fathers of Scrum, described how Scrum is based on leadership, honesty and transparency and how, in comparison, fraud, greed and corruption fuelled the current global financial crisis. He went on to explain how Toyota's lean Scrum methodologies helped it to put its main competition, General Motors, out of business and how, when it let leadership, honesty and transparency slip, it got into trouble that resulted in car recalls. He also described how lean allowed Toyota to develop the Prius in half the normal time by removing waste and impediments from the project. This case study also helped to demonstrate how lean agile methodologies helped define Scrum.

In talking about Scrum itself, Dr Sutherland went on to describe how Scrum:

- Removes management practices in favour of leadership
- Empowers employees
- Uses rules and procedures as enabling tools
- Uses hierarchy to support organisational learning.

Gantt charts are always 100% wrong

are always 100% wrong, as things change quickly and they become out of date. Teams must, therefore, be able to self-organise in order to be truly agile.

With the use of a diagram, Jeff Sutherland, showed how simple a framework Scrum really is. He went on to explain how Gantt charts

**I had previously
believed that Sprints
had to be 30 days to be
Scrum**



Introduction to Scrum: Shock Therapy – How new teams in California and Sweden systematically achieve hyperproductivity in a few sprints

Jeff Sutherland

Synopsis: New teams need to learn how to do Scrum well starting the first day. This talk will describe how expert coaches at MySpace in California and Jayway in Sweden bootstrap new teams in a few short sprints into a hyperproductive state. This requires new teams to do eight things well in a systematic way. Good ScrumMasters will make sure their teams understand these basics for high performance and great ScrumMasters will make sure the teams execute all of them well. This session will review the critical success factors for new Scrum team formation.

Hoping to learn some more about Scrum I went into Dr Sutherland's follow-up presentation and I wasn't disappointed as there was more of the same! It was good to have the idea that Sprints could be one, two, three or four weeks as I had previously believed that Sprints had to be 30 days to be Scrum.

Sutherland described how he encourages Scrum to be implemented and understood from board level down and two of the main things that get enforced, by the board, are Test Driven Development (TDD) and Continuous Integration (CI). I see these as two of the bare minimum requirements for software development.

I especially like the definition of 'done' given:

- Feature complete
- Code complete
- No known defects
- Approved by product owner
- Production ready

Dr Sutherland reiterated his previous point that Scrum should be lean and reminded me that although the right way to estimate stories is with relative story points, in a Sprint planning meeting those stories should be broken into tasks and real time estimates given to each one.

PAUL GRENYER

An active ACCU member since 2000, Paul is the founder of the Mentored Developers. Having worked in industries as diverse as direct mail, mobile phones and finance, Paul now works for a small company in Norwich writing Java. He can be contacted at paul.grenyer@gmail.com



Real QA – the argument for smarter quality assurance than mere testing

Tom Gilb

Synopsis: Traditional testing, as a means towards quality assurance, is far too costly and far too ineffective. There are much smarter ways to approach software quality. This session will argue with facts for a half dozen more-cost-effective ways to get reliability and all other qualities, like usability, security, adaptability, into your systems, than conventional testing.

It seems that Scrum themed sessions were the order of the day for me as Tom Gilb works closely with Jeff Sutherland and both attend each other's presentations. I went into this presentation unsure what to expect and hoping to learn something. Although I enjoyed it and may just possibly have missed something subtle, it appeared the whole point was that quality should be specified quantitatively in requirements, and those requirements should also be high quality and bug free, rather than trying to ensure it with only testing at the end.

Jon Jagger's Coding Dojo

I was invited, well it was more like press ganged, to a late collaborative programming session. I was a member of one of five pairs who were developing a simple unsplice function and accompanying tests. We ran several iterations and were informed we should be working as a group, rather than competing. At the end of each iteration the people in each pair were changed, and each pair would work on another pair's code. It was an interesting look at the way teams collaborate together in an agile environment. In retrospect I wish we'd planned and shared a bit better, but with the information we were given I think we did quite well.

Thursday

Hello, I'll Be Your Tester Today

James Bach

Synopsis: This talk is how I introduce myself to programmers. It is especially aimed at programmers who may wonder why any intelligent person would willingly be a tester, and why projects need testers, and how to work with testers. I will talk about what makes skilled testers different and special and about the commitments I make to the programmers I work with. I will help you set a high, but reasonable standard for the testers you work with.

If I am successful, then by the end of my talk, at least a few programmers in the audience will have become testers.

The ACCU conference has always been about quality, but this year is very special, especially with the keynote speakers. James back is a highly charged and very amusing individual. In fact he's the sort of tester I'd like to be working with.



He explained that he started his testing career as a test manager for Apple about 20 years ago. Throughout his key note, James highlighted the many differences between testers and developers. Including the fact that, apparently, testers have more social interaction. Another subtlety he pointed out was that testers don't break code, developers do, and testers do not steer projects, they make suggestions and hold the light so developers can create great code.

One of the best parts of the keynote were the testing related subtitles James added to a clip from *The Towering Inferno* where Steve McQueen is gathering the requirements he needs to fight the fire. I can't describe its brilliance in a way that will do it just here.

The point that resonated the most for me is that testers need to be able to think their way around a system with or without a specification or requirements. They need to think outside the box as, after all, requirements are often rumours, right? This is the opposite end of the scale from what Tom Gilb was saying yesterday. From an understanding of what the system should do and bags full of common sense testers should be able start testing and test effectively.

Genemodulabstraxibility

Steve Love

Synopsis: The title says it all! If it's hard to write unit tests, or they take too long to run, if your plugins need the whole application for their distribution, if you can't (re)use a bit of your colleague's code without importing the entire team's work, if your multi-threaded code performs better when you run it sequentially, or you've got 5 versions of the same 3rd party library littering your source repository, then you're suffering from this. Or perhaps just part of it...

Modern languages provide us with many tools for creating beautiful, modular, general, flexible and simple abstractions, yet it seems they give us even *more* tools for writing ugly, monolithic, specific, rigid and complicated, er, *concretions*.

With examples of both kinds in C#, Python, C++ and maybe even C++0x (if it stays still long enough to get the syntax right :-)), this is a talk about our (code's) propensity for wearing too many hats.

Steve is one of my oldest friends from the ACCU, going back to the first conference I attended in the early naughties. The lessons and observations he discussed in the session are what the ACCU is all about for me. He described lots of the common pitfalls of the code that a lot of people write and how to avoid them sensibly.

There was of course just the right amount of singleton bashing and plenty of humour, not to mention of course, the subliminal elephant that no one was meant to be thinking about. Other important points made were that too many comments is a sign of poorly written code and public inheritance is the strongest form of coupling.

What surprised me the most was Steve's description of a painful code review where he was pulled up for using single entry, single exit. The company Steve was working for actually explicitly stated that they tried to enforce multiple exit points. Just when I thought I'd heard it all....

Data warehouse, OLAP, Data Cubes, Business Intelligence – Buzzwords explained

Sven Rosvall

Synopsis: In the database world there are many buzzwords that most software engineers only hear about but never get experience with. This talk aims to clarify what lies behind some of these buzzwords and describe key differences from the more common transactional database. The talk will also provide enough insight for engineers to decide if any of these technologies are useful in their current or future projects.

The presenter has worked for many years on the periphery of the databases, wondering about these buzzwords. He eventually got involved with data warehouses and now wants to share his experiences with fellow engineers.

Sven is another ACCU member I've known for many years. His presentation did exactly what it says on the tin. I came away understanding the difference between OLTP and OLAP and that we really need to try and break our own databases up onto separate servers to improve performance.

Although I'm still a little hazy on data cubes, I now understand dimensions and how data is reorganised when it is transferred from OLTP to OLAP to make it easier to query. This was certainly one of the most useful sessions of the conference so far.

Database Testing Demystified

Sven Rosvall

Synopsis: Testing databases is not as easy as unit testing of classes and functions. Databases are full of state and internal logics which must be set up before testing can start. There are also lots of dependencies that are difficult to isolate or stub out.

This presentation will look at some techniques to create automated tests for databases and how to debug SQL code including single stepping stored procedures. We will automate database testing with the use of popular unit testing frameworks. We will test simple CRUD statements, calls to stored procedures and verify triggers. Test suites in Java and .Net will be demonstrated.

We will also see how a database can be developed with agile methods. Databases are traditionally developed up front as the database schema is difficult to change later.

Another great session from Sven (immediately following the previous one) even if the material from the second half does overlap what I'm speaking about a fair bit. The highlight here for me was seeing the SQL Unit Testing Framework SPUit in action and the ability to step through stored procedures in SQL Server Management Console.

TDD at the System Scale

Steve Freeman, Nat Pryce

Synopsis: We present our experience applying 'system-test first' test-driven development (TDD) in the development of large systems and systems-of-systems. We try to address integration and system testing as early as possible. The sooner the system is in a deployable state, the easier it is to react to changing business needs because we can deliver new features to users as soon as is deemed necessary. We therefore start by writing tests that build and deploy the system and interact with it from the outside and, to make those tests pass, add code to the system in the classic 'unit-test first' TDD style.

Many teams applying TDD start writing unit-tests and leave integration and system testing until late in the project because they find it difficult to write tests that cope with the distributed architecture and concurrent behaviour of their system. We will describe how we address common pitfalls, including unreliable tests, tests that give false positives, slow-running tests and test code that becomes difficult to maintain as the system grows.

We will also describe how writing system tests guides our architectural decisions. Writing unit tests first guides design of the code to be flexible and maintainable. In a similar way, we have found that writing system tests first guides the architecture to be easier to monitor, manage and support.

A couple of colleagues and I saw Steve and Nat speak at JP Morgan in February and it was a little bit like disorganised chaos with Steve doing most of the talking and Nat chipping in. There was a marked improvement this time, with Nat doing the majority of the speaking.

I was a little shocked when the layout of the room indicated it would clearly be interactive

I was a little bit concerned to see that they were advocating starting with system level tests before unit tests, but I can see how it might work in some circumstances. The same old system level problems, such as driving the



GUI, faking external systems, etc, are still there, but Nat and Steve came up with some interesting ideas to overcome these.

The main points that stood out for me were:

- Making the system scheduler external so that it can be faked during test.
- Use an audit interface instead of direct logging so the system can decide what it wants to report as a whole.
- Add hooks so that you can easily ask the system if it is happy.

Another point I realised is that I really need to read their book!

Friday

The Multicore Revolution: We've Only Just Begun

Russel Winder

I will and have travelled a long way to hear Russel speak, even for a short period of time, on many occasions. He is very entertaining and today was no different. Before talking about the architecture of multi-core processors, Russel spoke about the different forces occurring in atoms. Unfortunately it's so long for me since A-level physics, I was rather lost!

The two important points that Russel made were that we need to stop thinking in a single processor, single memory model and consider a multi-core, multiple memory model and that although the hardware is mostly there, there is a lot of catching up to do on the software side.

Fortunately no kittens were harmed during the production of Russel's slides as he is using Open Office.org instead of Powerpoint.

Understanding User Stories

Rachel Davies

Synopsis: On traditional projects, software developers are often presented with volumes of requirements that seem to be definitive and yet are full of holes. On agile projects, the whole team gets involved in establishing user stories on index cards. Come along to this session to understand the life-cycle of a user story from inception to implementation. You'll get a chance to practise working through some real user stories. We'll also investigate ways that agile teams handle requirements that don't seem to be user stories.

I didn't read the description of the session sufficiently! So I was a little shocked when the layout of the room indicated it would clearly be interactive. I don't really enjoy that sort of thing, but this was superb and the 90 mins just flew by!

This session was, for me, really about confirming what I have read recently about writing user stories. It was nice to know I was generally right and to find ways to smooth some of the rough edges.

We played a game where one person wrote down a story, e.g. 'A man opens a door.' Then the next person draws it and the following person then writes down what they think has been drawn and this goes right around the group. Each person only sees what the previous person has written or drawn. The

they were advocating starting with system level tests before unit tests

idea of the exercise is to show how if one person describes something and another person documents it, things get lost and other things get added. The final description was 'A man with a hat, closes a door.'

We also did a brief exercise where one member of a three or four person team is the product owner and the other members of the team ask them questions to determine what is needed for a system and write stories. This also worked very well.

Based on what I saw I'd be very happy to have Rachel come and coach my team.

FindBugs: Testing (Java-) Code Against Bugs And Defects

Bernhard Merkle

Synopsis: This session should be really fun. We will make a deep dive into findbugs, a very useful tool which every programmer should use to test Java code against potential bugs, defects and antipatterns.

After a first overview of the problem domain (static analysis), we will take some of Joshua Bloch's 'Effective Java' Rules from his book and see how/ if they can be implemented in a tool like findbugs. I will show how findbugs works internally, and we will implement a new bug detector (findbugs rule) to find specific defects we are interested in.

Additionally we cover different levels of static analysis like code-, design- and architecture-level. Interestingly findbugs checks code and design level, but fails itself (!) on the architecture level; which means the tool itself suffered major architecture erosion from version 0.7 to the current 1.3.8.

I am participating now in the findbugs project and contributed several architectural refactorings on the findbugs 2.0 branch which is currently under development and should hopefully be finished during the time of the ACCU 2010 conference. We will see how far an automated and tool supported approach can lead to better results and how tools like findbugs can discover areas of code- and design-erosion and suggest improvements. Additionally we will look at the next generation of analysis which is architecture analysis.

Bernhard took us through a very interesting look at what findbugs provides and showed us many examples of bugs in established code bases, such as Eclipse and the Java SDK itself, that could have been found simply by running findbugs. He included some great photos of non-programming bugs and lots of excellent humour.

Findbugs can also highlight architectural problems and Bernhard showed us how the architecture of findbugs itself has got steadily worse over the years.

Bernhard convinced me that we should add findbugs to our static analysis

coupled code is very difficult to change and even harder to understand

arsenal, which current consists of just PMD.

Saturday

Lightening Keynotes

Due to the volcano eruption in Iceland, Dan North didn't make it to the conference for the Friday keynote, so Russel Winder had been moved forward to Friday. The Saturday keynote was replaced with four fifteen minute 'Lightning talks'. Lightning talks are just very brief talks, usually only five minutes and usually for people who aren't already speakers to give them a chance to try it out. On this occasion the talks were given by four conference speakers:

- Robert 'Uncle Bob' Martin – The birthday problem.
- Walter Bright – C's Worst Mistake.
- James Bach – Testing models.

- Jim Hague – Are you getting enough?

All were humorous in one way or another and all very good.

Coupling as resistance to change, understanding and fixing it

Tony Barrett-Powell

Synopsis: Coupling, the number of things a unit of code depends on, is an important consideration when designing and maintaining software. Without due care a slow increase in coupling between units over a few releases of a software product, or even over a number of iterations within a release, can lead to software that is difficult to change, or worse results in a ripple effect throughout seemingly unrelated units. We cannot write software without some coupling, how else would the code achieve anything? Thus, coupling is not a bad thing in itself, it is the degree to which a unit is coupled to other units that can be undesirable, yet is not an absolute measure. Understanding of what coupling is, its many forms and how to recognise them is useful if we want to avoid unnecessary coupling in the software we create. It is important to control coupling in software if it is to be improved over a number of releases, as high coupling slows development, increases time to market and inevitably lost revenue.

This session attempts to explore coupling in all its manifestations, examine the difference between good and bad coupling and to consider its bedfellow cohesion. The session will also explore the techniques used before, during and after the act of design which can be used to reduce unnecessary coupling and as an aside looks at those which can lead to increased coupling. The intent of the session is to arm those designing and writing software with the understanding and techniques to create loosely coupled and maintainable software.

I have known Tony a good few years, but this is the first time I have seen him speak. He is very good and this subject is close to my heart.

Tony explained how lots of coupled code is very difficult to change and even harder to understand. He took us through inheritance coupling and temporal coupling and explained cohesion as a measure of the strength of functional relatedness of elements within a module. If it's all together it's cohesive, if it's split up it's coupled.

Tony continued by explaining Afferent and Efferent coupling. Afferent coupling occurs at the package level and efferent coupling is the number of classes inside a package that reference classes outside of a package.

A Simple Matter of Configuration – how can we tame the complex world of configuration?

Roger Orr

Synopsis: Configuration is a vital element of many programs. However it is often hard to get configuration right, leading to people wasting time and programs that do not work correctly.

In my experience explicit discussion of design options for the configuration of a program is rare and, all too often, the choice is made arbitrarily. I believe that looking at the usage patterns of the program early on helps to pick the best method(s) for configuring it and hence reduce the cost of problems caused by configuration issues.

Configuration is a complex subject and there doesn't seem to be an obvious single solution that works in every case, but we can try to fight against the common 'anti-pattern' of using multiple, unrelated, configuration techniques at the same time.



If it's all together it's cohesive, if it's split up it's coupled

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.

I must thank Blackwells and Computer Bookshop for their continued support in providing us with books.

Jez Higgins (jez@jezuk.co.uk)

Agile Estimating and Planning

By Mike Cohn, published by
Prentice Hall,
ISBN: 978-0131479418

Reviewed by Paul Grenyer

Verdict: Highly recommended

I bought this book because I'm generally



rubbish at estimating (I usually under estimate). Also, although we have the technical elements of agile (source control, unit tests, continuous integration, etc) sorted, our agile project management is not all it could be. *Agile Estimating and Planning* may be as close as I ever get to a silver bullet.

To be honest I expected to be let down and that the scenarios described in the book would not

match the situations I find myself in. I was not let down at all. The book covers both planning when features are important and planning when a deadline is important.

It taught me that it was wrong to break stories into tasks when release planning and to leave that for iteration planning. The book discusses the use of both story points and ideal days in estimating, what they both are, the differences between them and then suggests you should use story points.

It described what release and iteration planning are and when to use them. It also discusses how to predict, where necessary, and how to measure velocity in order to calculate the duration of projects. One of the most important things covered from my point of view was how, when

A Conference Retrospective (continued)

I will look at some of the issues surrounding configuration; firstly by trying to answer the 'six key questions' (who, why, what, where, when and how) for this subject to help understand the size of the problem and the forces at work.

I'll then sketch out some possible design patterns and look at their trade-offs and interaction with the intention that I (and you) can reduce the pain of getting programs working in different environments by making more informed and deliberate decisions.

I am looking at the problem in the general case, using examples from various problem domains, and expert knowledge of a particular language or API is not assumed.

This was the last session I attended as I was speaking in one of the final slots of the day. Again, I have known Roger, through the ACCU a long time and shared many a drink and lunch with him. He took us through a humorous look at how to configure applications. This wasn't a look at build configuration, but the way we actually pass settings to a program.

I have come across many of the same problems in my career, and although there was no specific solution to the problem, one of the proposed solutions, storing settings in a database is one I use regularly. Roger showed me lots of methods and their pitfalls, that I had never even considered!

Enterprise Web Application Development in Java with AJAX and ORMs

Paul Grenyer

Synopsis: Recently Java enterprise web application programming has been leaning towards a more classical J2EE approach. Traditional Java Server Page (JSP) programming, and even libraries such as Struts, are being replaced by new AJAX libraries that make GUI programming more straight forward, robust and easier to unit test.

In this session I will look at what an enterprise web application is.

I will demonstrate how to develop a more robust GUI with an AJAX library and how to create a more object orientated Data Access Layer (DAL) with an Object Relational Mapping (ORM) library.

After defining what an enterprise web application is I will move on to demonstrate how to create a DAL, with real code examples, and explain how to use a registry to abstract away Data Access Objects (DAOs) so that the real DAOs can be used in production and integration testing while seamlessly substituting mock objects for unit testing.

Then I will look at an AJAX library and demonstrate how to create the presentation layer in Java, again with real code examples, and make Remote Procedure Calls to access the DAL. I will then look at how to integrate the AJAX library into traditional Spring MVC in order to tap into the vast library of functionality that the Spring Framework can provide for web based enterprise application. I will explain how the tools provided by Spring make integration testing of DAO objects very simple.

Finally I will look at how to use Spring Security to authenticate users of the application and secure individual Remote Procedure (RPC) calls made from the client application, running in a browser, to the server.

I don't know how good I was, but this presentation flowed for me easier than any other presentation I have done. I have never managed to speak for the full ninety minutes before and I came in on time.

It was one of the final sessions of the conference, I was up against some stiff competition and most people had actually already gone home. Even so I had an audience of about half a dozen.

Finally

This was the best ACCU conference I have ever been to, simply because of the amount directly useful information I was able to absorb. I'm sure next year will be even better.

and with what to report to the product owner and stake holders.

The book finishes with a 60 page case study. I was tempted not to bother reading this as it goes over the main points covered in the rest of the book again. I was glad I read it and if you buy this book you should read the case study if you read nothing else. It helps put in context how estimating should be done and describes the processes surrounding it.

All I have to do now is write a distilled version for my team, including the project managers, product owners and stakeholder and put it into practice.

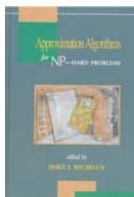
Computers and Intractability: A Guide to the Theory of NP-Completeness

by Michael R. Garey and David S. Johnson, published by W. H. Freeman and Co., 1979 (with an 'Update for the Current Printing' spanning two pages dated December, 1983)



Approximation Algorithms for NP-hard Problems

by Dorit S. Hochbaum et al., published by PWS Publishing Company, 1997



Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties

by G. Ausiello; P. Crescenzi; G. Gambosi; V. Kann; A. Marchetti-Spaccamela; and M. Protasi, published by Springer, 1999

Reviewed by Paul Colin Gloster



None of these books is suitable as a first book on algorithms.

They are all worth reading. They share a lot of overlap, but each also has its own merits making it worthwhile to read all three instead of merely any one of the three books.

Practical examples of how some of the abstract, general algorithms can be used were given in *Approximation Algorithms for NP-hard Problems*. As with many other books, varied application domains were mentioned without

managing to convey more than a tiny fraction of how many of the application domains these are useful for. It seems to me that more examples related to physics or electronic engineering were mentioned than in average books, but many other targets for deployment were also mentioned. The other books under review do not provide any advice as to some applications in the real world of the abstract algorithms. Of course, the onus is on you determine the relevance or otherwise of someone else's work to your own problem, but help does not go astray as it often is not obvious. For example, I was surprised to notice a variant of the knapsack problems promoted for refactoring in Bouktif; Antonioli; Merlo; and Neteler, 'A novel approach to optimize clone refactoring activity', proceedings of the 2006 annual conference on genetic and evolutionary computation.

Computers and Intractability: A Guide to the Theory of NP-Completeness mainly deals with NP-completeness, but also briefly deals with some complexity classes omitted in many textbooks. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties* deals with many more such complexity classes and does so in depth, but does not cover all of the complexity classes in *Computers and Intractability: A Guide to the Theory of NP-Completeness*.

However, some classes were briefly mentioned in *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties* without being explained, viz. ZPP; QP; NPOPB-complete; APX-intermediate; and various DTIME complexities. (Though it may not be too difficult to deduce what is meant by DTIME.)

Approximation Algorithms for NP-hard Problems also has many complexity classes which are not in introductory textbooks, and also classes which are only very briefly mentioned (such as RP; IP; and MIP). Few of the classes dealt with in depth in one of these books appears at all in any of the other two books. NC and RNC (complexity classes for concurrency) are in neither *Computers and Intractability: A Guide to the Theory of NP-Completeness* nor *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties* and are very briefly mentioned in *Approximation Algorithms for NP-hard Problems*.

Approximation preserving reducibility is an important topic which fortunately has a chapter devoted to it in *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. It is unfortunately not always possible, and this gloomy outlook is the closest thing to a mention of it in *Computers and Intractability: A Guide to the Theory of NP-Completeness*. This topic was almost briefly mentioned in *Approximation Algorithms for NP-hard Problems*. Indeed, a conspectus of the few hundreds of pages of *Computers and Intractability: A Guide to the Theory of NP-Completeness* is: you're screwed, in contrast to the other two books which provide bounds and advice on published alternatives to intractable algorithms. However, the vast majority of the hundreds of NP-complete problems (and some NP-hard problems; problems which are 'NP-complete in the strong sense'; problems not known to be in NP nor co-NP; #P-complete variants; pseudo-polynomial variants; and fortunately polynomial variants) listed in *Computers and Intractability: A Guide to the Theory of NP-Completeness* are ignored in the other two books. *Computers and Intractability: A Guide to the Theory of NP-Completeness* also had a smaller list of open problems, by which was meant a list of problems whose complexities were unknown at the time. That list has been out of date for over a decade.

The chapter on heuristic methods in *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties* is too short. This book's appendix on mathematical preliminaries is not rigorous. Problems were described as being equivalent to other problems (for example on page 424) with no way to find these other problems in the book.

The bibliography of *Computers and Intractability: A Guide to the Theory of NP-Completeness* helpfully shows where papers were cited in the book.

Obtaining an integer programming solution based on a real linear programming solution was mentioned on page 65 of *Approximation Algorithms for NP-hard Problems* with no indication that this does not generally result in an almost optimal integer programming solution until page 460. In this book it was mentioned that evaluating a matrix's determinant is bounded by $O(n^3)$ with no hint that less complex algorithms exist for this even in the general case. This book has a chapter on Monte Carlo algorithms based on Markov chain simulation for statistical physics. Physicists would not easily understand this chapter, but I could say that about anything in computer science.

In one section of *Approximation Algorithms for NP-hard Problems*, care was taken to avoid confusion between an approximation algorithm's absolute performance ratio and its asymptotic performance ratio. This is the only book under review which has a chapter on online

Bookshops

The following bookshops actively support ACCU (offering a post free service to UK members – if you ever have a problem with this, please let me know – I can only act on problems that you tell me about). We hope that you will give preference to them. If a bookshop in your area is willing to display ACCU publicity material or otherwise support ACCU, please let us know so they can be added to the list

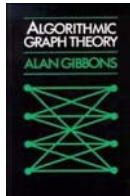
- Holborn Books Ltd (020 7831 0022)
www.holbornbooks.co.uk
- Blackwell's Bookshop, Oxford (01865 792792)
blackwells.extra@blackwell.co.uk

algorithms. In this sense, an online algorithm is an algorithm which is provided with only some of the input before it must output some of the output, as opposed to being online in the sense of being powered on or networked.

Algorithmic Graph Theory

By Alan Gibbons, published by Cambridge University Press, copyright 1985, 'digital printing 1999', ISBN: 978-0521288811

Reviewed by Colin Paul Gloster



This is a small book which is intended for students to learn about graph theory. It does have some algorithms in it, but it is not a rival to a textbook on algorithms, not even for merely graphs. It does not always give the most efficient algorithms: instead an attempt is made to explain about the problems, so research papers presenting more advanced algorithms than those detailed in the book are cited for further study. Instead of an unhelpfully unannotated bibliography, each chapter has a section with good educational background to relevant papers.

I am not convinced that the book's emphasis on understanding instead of as a cookbook of algorithms succeeds in imparting better understanding, but it does not make it harder to understand. A clear advantage of this book is that unlike other books (and even research papers) is that it does not present special cases as if they are generalities.

Do not allow the amount of criticism in this review to provide an unbalanced impression of this book... too many terms and concepts are introduced in exercises instead of in the chapters' bodies. The book has too many goto statements. It is sometimes hard to see the complement diacritic in subscripts. Even titles in American English were translated into U.K. English.

EJB3 In Action

By Debu Panda, Reza Rahman, Derek Lane, published by Manning, ISBN: 978-1933988344

Reviewed by Paul Grenyer



Verdict: Recommended

I bought and read this book as I wanted to learn about Enterprise Java Beans having only used Spring for Enterprise Java development up to this point. This is an excellent book for just that.

It explains in a reasonable amount of detail what stableful and stateless session beans, message driven beans and entity beans are. After a number of chapters describing how to use session beans and a further chapter on message driven beans a large amount of the book is turned over to entity beans and the Java Persistence API. As a user of Hibernate I found going over a very similar API somewhat tedious in places, but I am sure this would not be the case for a

novice ORM user. I also think Hibernate does it better.

I found the general style of the book, although chatty, quite easy to read. Although the authors mention many of the areas where Spring has similar or even better functionality it is clear, as you would expect, that the authors favour EJB in all cases even when EJB is still quite heavy weight in comparison.

My biggest criticism of the book is that it's more of a text book than a practical guide. Although the source code for the example application is available to download, it is not possible to put the application together just from reading the book. Deploying an application to a container is also handled very late in the book. If I was actually wanting to do some EJB development at this stage I would want to try things out and therefore having deployment examples at the start of the book would be paramount.

iPhone Advanced Projects

Various authors, published by APress, ISBN: 978-1-4302-2403-7

Reviewed by Pete Goodliffe



Verdict: OK

This is another book in the APress iPhone 'projects' series. It follows the form of its predecessors; it is a series of 11 essays by different authors on a particular topic which that they have some familiarity (I'm not sure I can always say 'expertise') in.

Production

First, the book production bears discussion. Unlike previous books, this one is printed in plain black and white rather than the accustomed luxurious full-colour. The book still costs the same amount, so this appears to be a ploy by publisher to increase profit rather than a way to position book at lower price point to gain more readers.

Unfortunately, the book suffers because of this. Not only is it less visually appealing, it is less practical and easy to follow. Some visual modification should have been applied to the format to make it work in black and white. Without colour, the sans-serif body text is harder to read and is less distinct from headings, subtitles, and illustrations.

In particular, many screen shots suffer. Often an author refers to details that are completely lost in a sea of black ink. These images should have been edited to improve contrast, or the text reworked to explain the point without an illustration.

Editing

On the whole this book has been well edited and proofed, probably better than some of the other books in the series. There are only a few typos. It hangs together as well as the other books, which is fine – it's a series of unconnected essays on different topics.

As another 'grab bag' book, you'll either want to make a purchase if you care about a specific topic included, or you just want a general overview of a number of topics.

Topics

The book covers the expected iPhone topics: Graphics/UI (creating particle systems, OpenGL ES reflections, and making responsive UIs), networking (writing an app backend server to integrate with iPhone app, using push notifications, yet another socket-based UDP example, and sending an email from your app), audio, debugging, and data handling (both in SQLite, and Core Data).

Most chapters are sufficiently detailed. The audio chapter, I felt, was a chatty overview, but didn't provide as good a grounding as just reading the iPhone audio docs.

As ever, the detail in each chapter cannot replace the iPhone SDK docs which are complete and detailed. However, there are often little information gems that will help your development and save you a little time working with that technology.

Conclusion

If you're new to iPhone programming, this might be an interesting book. Also consider the other APress titles in the series, and chose the book with the topics you are most interested in.

JavaScript for Programmers

By Paul J. Deitel and Harvey M. Deitel, published by Prentice Hall, ISBN: 978-0137001316

Reviewed by:

Summary: Not recommended

I didn't get to the last page of this book, in fact only page 100 or so. It is simply awful. From the title one would expect that programmers could read this and gain JavaScript knowledge. In fact it turns out to be a long-winded painful introduction to all sorts of things including the history of the web, CSS and even object-oriented programming, which is so painfully and badly explained that 'programmers' would only tear their hair out. I need write no more.



Ruby in Practice

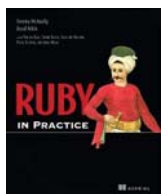
By Jeremy McAnally & Assaf Arkin et al, published by Manning, ISBN: 978-1933988474

Reviewed by Simon Sebright

Summary: Recommended with reservations

I enjoyed reading this book. It was just the thing for me, being an experienced programmer and having just got into Ruby, in my case through the Rails framework.

The book is a series of small, fairly independent articles/chapters on using Ruby in the real world. These are collated into a number of Parts:



View From The Chair

Jez Higgins
chair@accu.org



2009 has been more of the same for ACCU.

In many ways, this is a good thing. The organisation continues to do the things it does, which are the things the membership enjoy and benefit from, and so they continue to support them. Things sustain themselves. Traffic on accu-general is fairly high, the signal-to-noise ratio is good (even if the signal-to-pun ratio needs work), and the atmosphere is friendly.

CVu and Overload are consistently interesting, across a broad range of subjects. The appointment of Steve Love as regular CVu editor has provided some welcome stability there. While our series of guest editors through 2008 and into 2009 was a success, it put a lot of

pressure on, in particular, the Publications Officer and our Production Editor. Steve's just celebrated his first year at the CVu helm, with Ric Parkin shortly to mark two years steering Overload.

While I'm pleased the organisation seems to be ticking over, I confess, as I stand down as Chair, to a certain disappointment. When I took over as Chair in 2006, I was hopeful that we would be able to raise the membership. At that time, the roles of Conference Chair and the ACCU Chair were expressly split. Chairing the conference is a time consuming activity, and dividing the roles would allow the ACCU Chair to focus on the organisation. While that's been the case, the time I've spent hasn't been used quite as I anticipated.

Voluntary organisations typically do roll steadily on and change is, generally, only provoked by some kind of crisis. By crisis, I don't necessarily mean anything disastrous, but

simply some kind of unsettling event – losing funding or, indeed, gaining funding, a resignation from a post, and so on. The crisis provides an opportunity to try new things. The ACCU Chair standing down is such a time, and presented such an opportunity, but I missed the chance it gave me. On the plus side, I didn't completely balls it up. I've enjoyed it too.

My successor in the Chair, Hubert Matthews, will be here to introduce himself next time.

I'd like to thank the many people who've helped and advised me over the last four years, including the various journal editors, the conference Chair and committee, and the past and current members of the committee.

In particular, I'd like to single out the work done by secretary Alan Bellingham, membership secretary Mick Brooks, and treasurer Stewart Brodie. If not for them, we wouldn't be here now.

Bookcase (continued)

Ruby Techniques, Integration and Communication, Data and Document Techniques.

Each article is well-written, and has (apparently) sample code to go with it where appropriate, although it's not quite clear who actually wrote each article, there being rather a lot of "with" authors on the front page. The authors are well aligned with the Pragmatic Programmer mindset, which gives the astute reader a boost in confidence.

I learned quite a bit about Ruby in general, in particular how many areas are covered by freely-available libraries, and how the scripting approach to solving some problems can be so easy.

I have reservations for a couple of reasons. Firstly, there could have been more discussion and demonstration of the actual Ruby techniques involved in each of the areas. Secondly, it's aimed at people who are experienced programmers plus they are 'getting

into' Ruby. But, if you are one of those people, then happy reading.



Learn to write better code

Take steps to improve your skills

Release your talents