The magazine of the ACCU

www.accu.org

{cvu}

Volume 22 Issue 5 November 2010 £3

Features

A Comparison of Boolean Flags
Paul Floyd

This Time I've Got It...
Pete Goodliffe

What Is Code-Dojo?
Jon Jagger

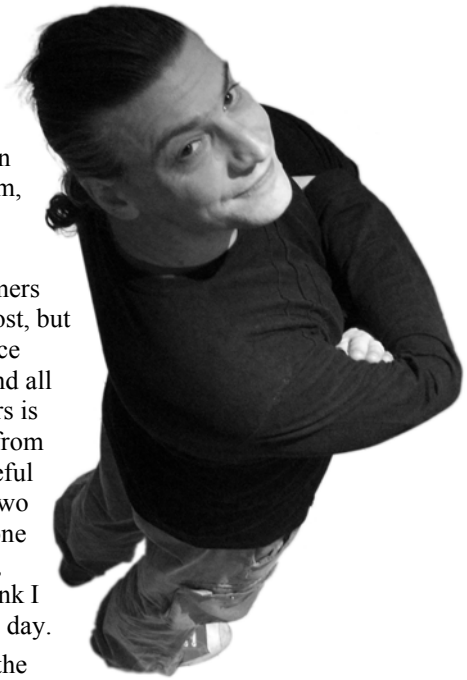
A Game of Roulette
Baron Muncharris

Regulars

Inspirational (P)articles
Desert Island Books
Code Critique
Book Reviews

{cvu}
Volume 22 Issue5
November 2010
ISSN 1354-3164
www.accu.org

What's It Like?



I've never had the opportunity to do pair programming 'for real' in a commercial environment. It's always considered either a waste of expensive resources (people) or, bluntly, a pointless exercise – which is just a variation of the first reason. Some teams I have spoken to about it have been more positive about the idea, but still often view it as best done on an ad-hoc basis, for solving a particularly difficult problem, or dealing with new code, rather than as a generally applicable practice to be done all the time.

For myself, I've only ever paired with other programmers for short periods of time – maybe an hour or two at most, but often just a few minutes – but I've found the experience enlightening and productive. A second pair of eyes, and all that. The argument that the sum of paired programmers is more than the two individuals is the common riposte from proponents of the practice, to those who see it as wasteful of resources, but I also see the difficulty with having two people doing one job. When I have paired with someone to solve a particular problem, it's been a quite intense, fast-paced and yes, exciting experience, but I don't think I could sustain that level of concentration all day, every day.

But then, maybe I have it all wrong, and I'm missing the point. All along I have been looking for that middle ground, the compromise which compromises the benefits of it, in the end. Perhaps it's one of those things that I must immerse myself in completely to enjoy the benefits the advocates of pair-programming claim. Now we're back at the beginning – I still need an environment to do that in!

If you've worked in such an environment, I'm sure I am not the only reader of C Vu who would be most interested to hear of your experiences – good, or bad.

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

12 Desert Island Books
Alan Stokes chooses to take old friends with him.

13 Inspirational (P)articles
Frances Buontempo continues her quest for positive experiences.

14 Code Critique Competition #66
Set and collated by Roger Orr.

18 Regional Meetings
A roundup of local meetings and events.

REGULARS

20 Bookcase
The latest roundup of ACCU book reviews.

24 ACCU Members Zone
Reports and membership news.

FEATURES

3 A Game of Roulette
The Baron has two games to play!

4 On a Game of Chase
The Baron's student acquaintance analyses his previous game.

6 A Comparison of Boolean Flags
Paul Floyd compares different ways of representing flags in C++.

9 What is Code-Dojo?
Jon Jagger explains the game.

10 This Time I've Got It...
Pete Goodliffe tells us a story of stress, short-sightedness, and solutions.

SUBMISSION DATES

C Vu 22.6:	1st December 2010
C Vu 23.1:	1st February 2011
Overload 101:	1st January 2011
Overload 102:	1st March 2011

WRITE FOR C VU

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

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

ADVERTISE WITH US

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

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

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

COPYRIGHTS AND TRADE MARKS

A Game of Roulette

The Baron has two games to play!

Greetings Sir R-----! I must say that it is a pleasure to see you this fine evening. I take it that I might tempt you with a glass of fortifying spirit and perhaps a little sport alongside?

Splendid fellow!

I propose a game popular at the gaming tables of the court of Y--- gakha.[1] I availed myself of such many times during my long stay as most honoured guest within its vast maze of gleaming spires. This great and, truth be told, extremely rewarding honour was granted me upon my discovery that one of the courtesans was, in fact, a kumiho; a were-fox of murderous intent, over-fond of the taste of human liver and distressingly common in those parts at the time.

After a spate of killings amongst the minor nobility, my suspicions were aroused that just such a beast was responsible by a sudden and unexpected shortage of fava beans and Chianti in the court's provisions.

I therefore donned my colours and rode, horn in hand, into the royal audience chamber. All eyes turned to me as I sounded the call to hunt, but one pair in particular, belonging to a courtesan with a somewhat vulpine countenance, betrayed fear rather than shock. I knew instantly that I had found my quarry and spurred my steed toward her as she took her true form and fled.

That pursuing so nimble a creature over the roofs of those spires on horseback was an impossible task hardly needs mention; nor, for that matter, does my having successfully done so.

But I am keeping you from your sport.

Come, join me at the roulette wheel and I shall explain the rules.

Your stake shall be one coin for which you shall spin the wheel three times. After each spin you shall mark the point on the wheel that is closest to you. If the triangle formed by connecting these points with a silken thread encloses the centre of the wheel you shall win four coins from my purse.

If this does not take your fancy, I have another.

In this game, on paying your stake of one coin a volunteer shall, having been blindfolded and spun about until a little confused, pick a point upon the face of the wheel with the aid of a pin. You shall then spin and mark the wheel three times as before but shall win only if the triangle of thread encloses the chosen point, albeit six and one half coins rather than four.

When I described these games to that odious student acquaintance of mine, he started blabbering on in his familiar incomprehensible fashion but, in an unusual display of lucidity, eventually noticed that I cared not one whit for his stilted nonsense and demanded that I show more sympathy to those of his vocation. Vocation! The miserable cur has the gall to suggest that he and his ilk have a divine calling!

Presumably buoyed by this self appointed elevation in status he compounded his insubordination by suggesting that I should take better care of my uniform! Were I not, as you well know, the most modest and merciful of noblemen, I should have run him through on the spot!

Still, let us not tarry on this distasteful episode; come replenish your glass whilst you consider this wager!

Listings

Listing 1 is a A C++ implementation of the 1st game.

Listing 2 is a C++ implementation of the 2nd game.

Notes

[1] With thanks to Seung Yang for this puzzle.

Listing 1

```
struct point
{
  double x;
  double y;
  point(const double x,
     const double y) : x(x), y(y)
  {
  }
};

point
spin()
{
  static const double pi = 2.0*acos(0.0);
  const double theta = 2.0*pi*double(rand())/
     (1.0+double(RAND_MAX));
  return point(sin(theta), cos(theta));
}

bool
enclose_0(const point a, const point b,
   const point c)
{
  return (a.x*b.y<a.y*b.x && b.x*c.y<b.y*c.x
         && c.x*a.y<c.y*a.x)
     || (a.x*b.y>a.y*b.x && b.x*c.y>b.y*c.x
         && c.x*a.y>c.y*a.x);
}

bool
play()
{
  const point a = spin();
  const point b = spin();
  const point c = spin();
  return enclose_0(a, b, c);
}
```

Listing 2

```
bool
enclose_o(point a, point b, point c,
   const point o)
{
  a.x -= o.x;  a.y -= o.y;
  b.x -= o.x;  b.y -= o.y;
  c.x -= o.x;  c.y -= o.y;

  return enclose_0(a, b, c);
}

bool
play(const point random_point)
{
  const point a = spin();
  const point b = spin();
  const point c = spin();
  return enclose_o(a, b, c, random_point);
}
```

On a Game of Chase

The Baron's student acquaintance analyses his previous game.

The Baron's latest game involves racing a pair of coins around the outer squares of a chessboard, with Sir R-----'s coin beginning at the lower left and the Baron's at the lower right.

Each turn consists of the Baron moving his coin four squares away from Sir R-----'s and Sir R----- moving his towards the Baron's by a number of squares equal to the roll of a die.

If the Baron is able to return his coin to the lower row without being caught up or overtook he receives one coin from Sir R-----. If not, Sir R----- receives forty coins for each square between the Baron coin and the lower left square.

We can reckon Sir R-----'s expected winnings at this game as the winnings if the Baron is caught on the first move multiplied by the probability of doing so, or zero, plus the expected winnings from the remaining turns multiplied by the probability that he is not, or one.

Whilst this may not seem especially insightful, we can likewise reckon the expected winnings from the remaining turns and thusly arrive at the desired result.

Noting that the Baron's n^{th} move leaves his coin on the $4n+7^{th}$ square from the bottom left, we have

$$e_n = p(r_n \geq 4n+7) \times 40(21-4n) + p(r_n < 4n+7) \times e_{n+1}$$

where r_n is the position of Sir R-----'s coin on the n^{th} turn.

Since the Baron will reach the lower row of the board on his 6^{th} turn, we also have

$$\forall n > 5 \qquad e_n = 0$$

where the upside down 'A' means *for all*.

Now r_n will be equal to the sum of n rolls of the die so we must figure the probabilities of this sum being equal to any particular value.

The usual method for calculating such probabilities is by convolution. If we have two integer random quantities, x and y, this means

$$p(x+y=k) = \sum_i p(x=i) \times p(y=k-i)$$

where the capital sigma stands for the sum of the expression that follows it for all integers.

If we denote the sum of n rolls of a die as x_n, this becomes

$$p(x_n = k) = \sum_{i=1}^{6} \frac{1}{6} \times p(x_{n-1} = k-i)$$

$$p(x_1 = k) = \begin{cases} \frac{1}{6} & 1 \leq k \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

If we were to use these probabilities in our reckoning we would be committing a grievous error since the probabilities we require are those conditional on Sir R----- not yet having won the game. These would give weight to turns after the game had in fact finished and would consequently lead to the overestimation of the expected winnings of the game.

I explained this problem to the Baron when he described the rules of his game to me, but I fear he didn't entirely grasp its significance.

Listing 1

```
double
N(const double n, const double k)
{
  if(n==0.0)          return (k==0.0) ? 1.0 : 0.0;
  if(k<n || k>n*6.0) return 0.0;
  if(n==1.0)          return 1.0;
  double sum = 0;
  for(unsigned i=1;i<=6;++i)
     sum += N(n-1.0, k-double(i));
  return sum;
}

double
p(const unsigned long n, const unsigned long k)
{
  return N(double(n),
     double(k)) / pow(6.0, double(n));
}
```

We can figure the correct probabilities by counting the number of ways we can arrive at a square from one that was behind the Baron's coin in the previous turn. We do this by noting that the lowest roll of the die, x, that could possibly deliver Sir R-----'s coin to the k^{th} square must satisfy

$$k - x < 4(n-1) + 7$$

and hence

$$x > k - 4n - 3$$

The number of ways Sir R----- can arrive at the k^{th} square on the n^{th} turn is consequently

$$N(r_n = k) = \sum_{i=\min(1,\,k-4n-2)}^{6} N \quad (r_{n-1} = k-i)$$

with the special case of

$$N(r_1 = k) = \begin{cases} 1 & 1 \leq k \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

To calculate the probability we must divide this number by the number of possible games that take n turns, given by

$$N(r_1) = 6$$

$$N(r_n) = 6 \times N(r_{n-1} < b_{n-1})$$

Using this probability in our formula for the expected winnings yields 475 of 486^{th} parts of a coin and in consequence I should not have advised Sir R----- to take up this wager. ■

Figures

Overleaf are histograms of dice sums:

- Figure 1 for 1 dice
- Figure 2 for 2 dice
- Figure 3 for 3 dice
- Figure 4 for 4 dice

Figure 1

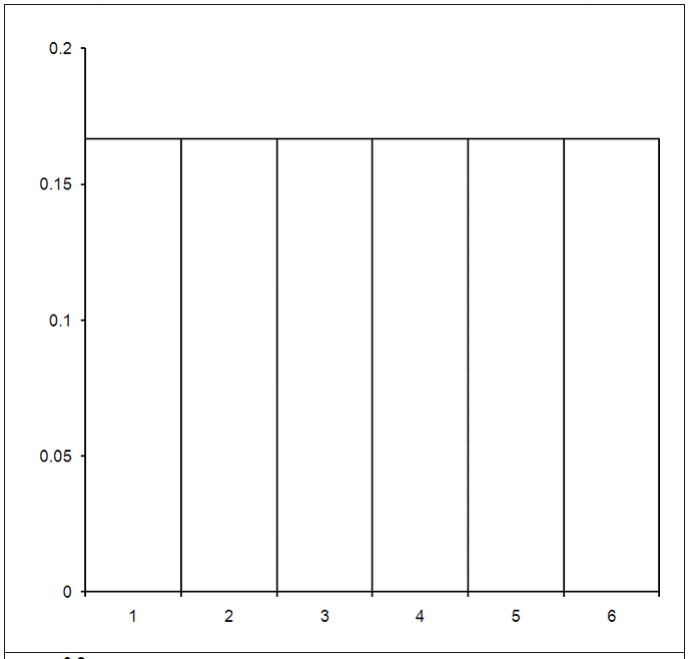


Figure 2

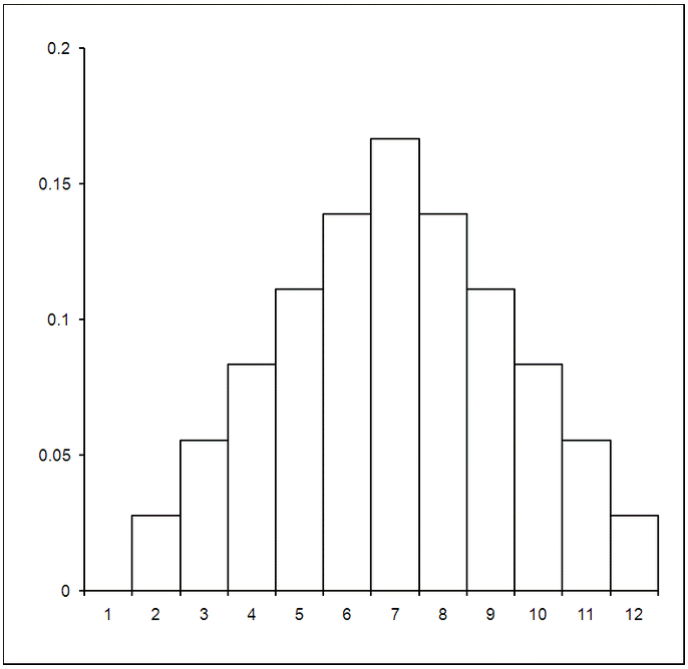


Figure 3

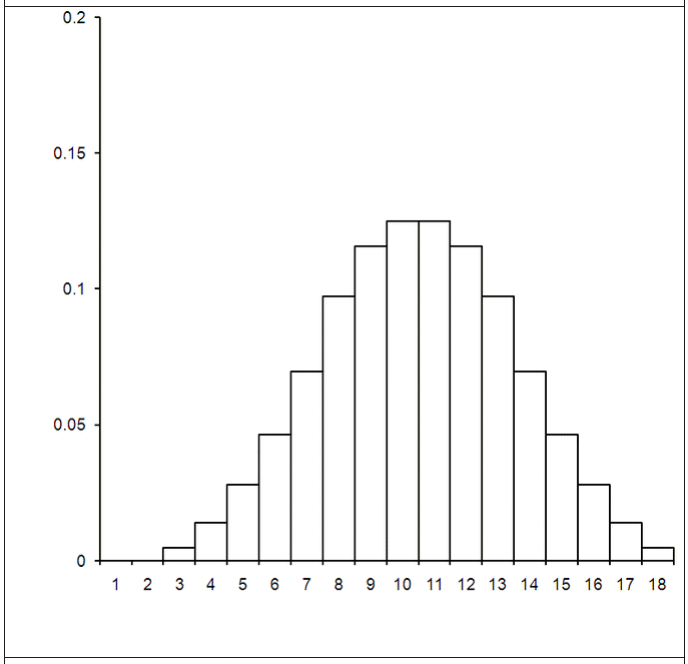
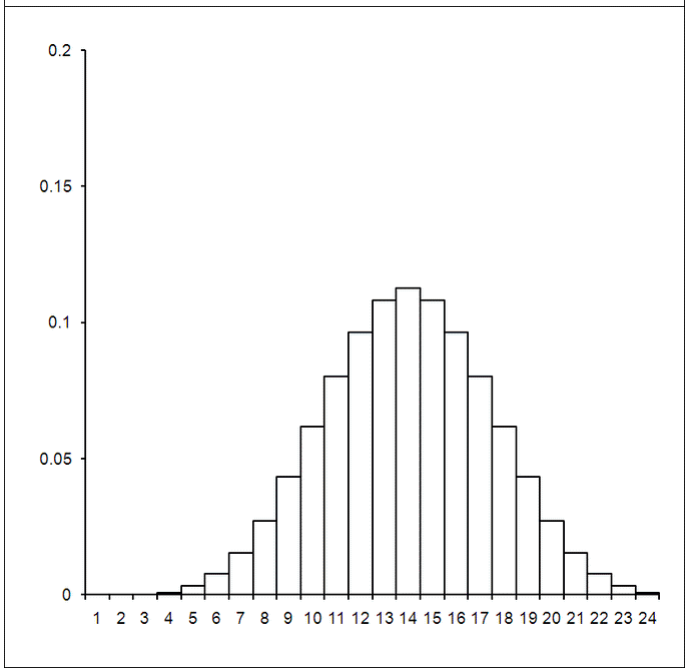


Figure 4



A Comparison of Boolean Flags

Paul Floyd compares different ways of representing flags in C++.

In this article, I will explore some of the trade-offs between the various ways that you can represent flags in C and C++. This is aimed at the context of class/struct members and I will concentrate on memory use and speed.

Let's start with a quick tour of the representations that I'll be looking at

- `bool`
- `bitfields`
- `std::bitset`
- `std::vector<bool>`

A `bool` flag needs no explanation. `Bitfields` have a limitation that they can only be members of structs/classes. Further, in this article I'll only consider bitfields that are one bit wide – the minimum to represent a binary flag. Both of these types of flags use named variables. `std::bitset` and `std::vector<bool>` are a bit different. For these, the collection of flags is named as a variable and the elements are accessed by index (or iterator for `std::vector<bool>`. It is fairly straightforward to achieve the same effect as `bool`/`bitfield` named flags with these two, using an `enum` to name the flags. For instance, consider:

```
class A
{
  public:
    enum { flag0, flag1, flag2, flagSize };
    std::bitset<flagSize> flags;
};
```

Then it is possible to write code like:

```
if (flags[flag0])
{
  ...
```

compared to using a bitfield:

```
class A
{
  public:
    unsigned int flag0:1;
    unsigned int flag1:1;
};
```

And the client code:

```
if (flag0)
{
  ....
```

Whilst you can do with a `bitset` or `vector<bool>` what can be done with `bitfield`s or `bool`s, the opposite is not true. `std::bitset` offers many functions that are oriented towards manipulating flags. To list a few, conversion to/from strings, logical operations between bitsets, checking whether any bits in the bitset are set, set/reset/flip the whole bitset. `Std::bitset` is statically sized. For instance, you might declare a collection of flags as:

```
std::bitset<10> flags;
```

which makes flags an array of bits with a static size of 10. As you might expect with such a template, each instantiation with a different size represents a different type. This means that if you have a `std::bitset<10>` and a `std::bitset<12>`, then you won't be able to use `std::bitset` operators like `==` or `=` with them.

`std::vector<bool>` has the usual standard library container features, plus the `flip()` member function to invert all of the flags. It is dynamically sized. Obviously, if you want to have named flags, then the number of names cannot be extended at runtime – as an example, in class `A` above, you can't add a `flag3` to the enum at runtime. `Std::vector<bool>` offers all of the associated standard container features, plus the availability of the associated standard algorithms.

In order to measure the code size for accessing a flag of the various types, I wrote a series of small `set`/`get` functions such as those in Listing 1.

Listing 1

```
class bf
{
  public:
    bool getBool() const;
    ...
  private:
    bool mBool;
    ...
};

bool bf::getBool() const
{
  return mBool;
}
```

I then compiled this with a number of compilers/platforms, using basic high optimization. Where possible, I used `nm` to obtain the sizes of the functions. `nm` is a UNIX tool for examining the contents of compiled object, library and executable files. From the man page 'nm - print name list of an object file'. It can tell you things like the names, sizes, offsets, types (function, data), bindings (static, global), index (which may be `undef` for an undefined symbol). The output is usually quite verbose, as most compilers add lots of extra little bits and bobs. Listing 2 contains an example. (I've trimmed over 100 lines of the output.)

`nm` does have a different format on BSD and SysV systems. Personally, I prefer the SysV output. Linux `nm` supports both, defaulting to BSD format, but you can get the SysV format by using the `-f sysv` option.

On platforms where `nm` is not available (or is not capable of producing size information), I resorted to dumping assembly, and calculating the sizes from the offsets. This technique does mean that the results may be a little distorted between platforms – sometimes the figure includes padding.

Table 1 presents the average bytes per function, and the average normalized sizes of the functions, relative to the smallest. For instance, this means that on average, the `get bitset` function was 2.24 times the size of the minimum. Here, 15 combinations of platforms and compilers were used.

PAUL FLOYD

Paul Floyd has been writing software, mostly in C++ and C, for over 20 years. He lives near Grenoble, on the edge of the French Alps and works for Mentor Graphics developing a mixed signal circuit simulator. He can be contacted at pjfloyd@wanadoo.fr.

Table 1

	set						get					
	nothing	bitfield	bitfield3	bool	bitset	vector	nothing	bitfield	bitfield3	bool	bitset	vector
Avg Bytes	6.07	25.43	28.14	11.64	30.29	126.931	9.07	15.43	19.29	9.43	18.43	109
Normalized	1	5.24	5.9	2.2	6.14	26.42	1.07	1.87	2.42	1.06	2.24	13.83

Listing 2

```
nm -C bitfield

bitfield:

[Index]   Value      Size     Type  Bind  Other Shndx   Name

[95]    | 134547680|      100|FUNC |GLOB |0    |9      |main
[93]    | 134547280|        0|FUNC |GLOB |0    |UNDEF  |rand
[73]    | 134547264|        0|FUNC |GLOB |0    |UNDEF  |srand
```

Whilst I did expect **bitset** to require more code than **bool** (due to logical masking and shifting), I was somewhat taken aback at the much larger code generated by all compilers/platforms for **vector<bool>**. For the two template classes, **bitset** and **vector<bool>**, the object code may also include template instantiations, depending on how the C++ standard library has been built for that compiler/platform. This object code was not counted in the above figures.

There are two versions of the **bitfield** function. The first one, plain **bitfield**, reads and writes a **bitfield** at position zero. The second is at position 3. The table shows that accessing **bitfield**s other than at position zero adds roughly 3 bytes to the code size for each read and write. This is because no shift is required at bit position zero.

This shows the relative size of the code to access the different types of flags. How does this balance with the size of instances of the flags? I'll restrict the comparison to just **bitfield** and **bool**. On average, a **bitfield** costs 9.86 bytes more per read, and 16.5 bytes per write in the code text. On the other side of the balance, there's the heap memory containing **struct** instances. Assuming that the **bitfield**s are all contiguous and part of the same 32bit unsigned **int**, the memory occupied per struct is 4*ceil(*n*/32) compared to a straight n bytes for bools. That makes a difference of n – 4*ceil(*n*/32). Let's take an example of a struct with 28 flags, 1000 reads and 100 writes. That works out at an extra cost of 9.86*1000 + 16.5*100 = 11510 bytes when using **bitfield**s over **bool**s. Each instance of the struct saves 28 – 4*ceil(28/32) = 24. So to break even, 11510/24 = 480 instances are required before the use of **bitfield**s will starts paying off with a total reduced memory footprint. This example uses a fairly high packing factor of 28/32. If there are only 10 **bitfield**s, then 1919 instances would be required. If 4 or fewer **bitfield**s are used, then there is no break even, **bitfield**s will never use less memory.

To measure the speed, I used a similar series of standalone functions (see Listing 3). These were called using a test harness based on the code from Jon Bentley's *Programming Pearls*. For this test, the code was compiled without optimization (with optimization turned on, it is often difficult to prevent the compiler from optimizing away the whole timing loops).

The results were obtained from 14 combinations of platforms and compilers.

Here we see that the **bitfield** version is a fair bit slower than the **bool** one. **bitfield3** is only a tiny bit slower than **bitfield**. Both the **bitset** and **vector<bool>** versions are very much slower. It also seems that there is a wide range of performance between different compilers/platforms.

Conclusions

If performance is of no issue, then use **std::bitset**, enjoy the functionality that is offered, and save yourself re-inventing the wheel.

If speed is the key, then use **bool** flags.

If memory is your bottleneck, then things become more complicated. You will have to choose between **bool** and **bitfield**s. If you have much code and few data instances, then **bool** wins. When the number of data instances much outnumber the code accesses, then **bitfield**s will win. If you do use **bitfield**s, try to ensure that your most used flags is put in bit position zero, and don't use them unless you have 5 or more flags.

Afterword

Compilers/platforms used:

- Solaris 10 x64, Sun Studio
- Solaris Express 32bit, Sun Studio
- Solaris 10 SPARC, Sun Studio
- Mac OS X 10.5.1, GCC
- FreeBSD 6.2, GCC
- Fedora Core 6 32bit, GCC
- Windows 2000, Visual Studio 2003, GCC, OpenWatcom, Digital Mars
- RHEL 4 64bit, GCC, Sun Studio
- RHEL 4 32bit, GCC, Sun Studio

The only big omissions that I know of are the Intel and Borland Compilers. I could have added HP-UX, but I think that I have quite a wide spread of tests already.

I struggled a bit with the compilers that I'm not familiar with (OpenWatcom and Digital Mars). In particular, I couldn't extract the function sizes with Digital Mars, so this is omitted from the test. OpenWatcom gave very poor speed results, so these were excluded from the results. The **bitset** results for GCC/cygwin are excessive, and I removed that platform/compiler combination.

The time results are printed directly. The size results had to be further processed after compiling the object files. Where possible I used **nm**. On Mac OS X I used **otool -t -v**. For Windows GCC and VisualStudio I used **objdump -d**. Lastly, for OpenWatcom I used **wdis**.

The normalized figures don't always have a minimum of 1. For the size measurements, this was due to there not being one function that always had the lowest byte count.

I haven't included figures for the tests that I did with **char**/**int**/**long**. On average, **int** was a little faster than **bool**, but even less space efficient. ■

Table 2

	get						set				
	empty	bitfield	bitfield3	bool	vector	bitset	bitfield	bitfield3	bool	vector	bitset
Avg time	10.48	15.24	15.29	11.34	157.88	89.13	16.88	17.12	11.57	163.14	95.3
Norm diff	-	5	5.06	1	154.92	82.66	5.88	6.1	1	140.15	77.86

Listing 3

```
#include <bitset>
#include <vector>

class bf
{
public:
   bool getBool() const;
   void setBool(bool v);
   bool getBitset() const;
   void setBitset(bool v);
   bool getBitfield() const;
   bool getBitfield3() const;
   void setBitfield(bool v);
   void setBitfield3(bool v);
   bool getVector() const;
   void setVector(bool v);
   bool getNothing() const;
   void setNothing(bool v);
   void unused() const;

private:
   bool mBool;
   std::bitset<8> mBitset;
   unsigned char mBitfield:1;
   unsigned char mBitfield2:1;
   unsigned char mBitfield3:1;

   std::vector<bool> mVector;
};

bool bf::getBool() const
{
   return mBool;
}

void bf::setBool(bool v)
{
   mBool = v;
}

bool bf::getBitset() const
{
   return mBitset[0];
}

void bf::setBitset(bool v)
{
   mBitset[0] = v;
}
```

Listing 3 (cont'd)

```
bool bf::getBitfield() const
{
   return mBitfield;
}

bool bf::getBitfield3() const
{
   return mBitfield3;
}

void bf::setBitfield(bool v)
{
   mBitfield = v;
}

void bf::setBitfield3(bool v)
{
   mBitfield3 = v;
}

bool bf::getVector() const
{
  return mVector[0];
}

void bf::setVector(bool v)
{
  mVector[0] = v;
}

bool bf::getNothing() const
{
   return true;
}

void bf::setNothing(bool /*v*/)
{
}

void bf::unused() const
{
}
```

Write for us!

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

What do you have to contribute?

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

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

What is Code-Dojo?

Jon Jagger explains a way to practise our coding skills.

In martial arts (such as karate) a dojo is a place where martial artists meet to practise their martial art. Similarly a code-dojo is a place where software developers meet to practise writing software.

A code-dojo is all about doing the practice – it's a way to try and improve your skill rather than finish a piece of software.

In a typical code-dojo you split into small groups, each group works on the same small problem, often in a test-driven manner, periodically rotating the keyboard driver.

Code dojo issues

After attending numerous code-dojos I began to notice some things which didn't seem ideal:

- It frequently took a long time for some of the groups to install their chosen compiler and test framework onto their laptop.
- Different people use different editors and this again creates a bias in favour of the laptop-owner in each group.
- Often, at the end of the practice each group presents their solution. Again this seemed to waste a lot of time as each group struggled in turn to attach their laptop to a projector.
- Your options are very limited if someone has to leave early and take an in-use laptop with them.
- Sometimes one of the groups is pronounced 'the winner'! What is that all about?
- After everyone had finished and walked home the collective efforts were typically lost, never to be thought about again.

It occurred to me that all of these problems could be alleviated if you could code and test inside a browser! I figured it wouldn't be too hard to write a server offering a simple browser-based code-editor environment together with a button to submit the code to the server. The server would then save the submission, run the tests, and return the test outcome; green if all the tests passed, red if one or more tests failed, yellow if there was a syntax error. With a setup like this:

- The start up time would drop to almost nothing.
- Everyone would be equally disadvantaged by having to use the same browser-based editor (I like to think I abuse all people equally regardless of their race, colour, creed, sex, religion, ...).
- Any laptop could instantly substitute for any other laptop.
- At the end everyone could have instant access to every run-tests-increment of every group's code (without needing a projector).
- After the end everyone would still have access to everyone's code.

often each group implicitly assumes they are competing against the other groups

What are we practising?

As I thought about this browser-based code-dojo idea I started thinking hard about exactly what skills a code-dojo should help you practise.

I thought about something many classic books and ageing gurus try and tell us, but which we seem prone to forget each new generation:

- Jerry Weinberg: 'It's always a people problem' (*The Secrets of Consulting*)
- Tom DeMarco and Tim Lister: 'The major problems of our work are not so much technological as sociological in nature' (*Peopleware*)
- Fred Brooks: 'The Mythical Man Month is only incidentally about software but primarily about how people in teams make things.'
- Richard Gabriel: 'My overall bias is that technology science, engineering and company organization are all secondary to the people and human concerns in the endeavor.' (*Patterns of Software*)

it's not uncommon for people to spend two hours in a group of five people and not even ask the names of the other four people in the group!

I felt an ideal code-dojo should encourage social practice as well as technical practice. For example, in a regular code-dojo it's not uncommon for people to spend two hours in a group of five people and not even ask the names of the other four people in the group! And if you think that's bad think about the group to group communication. Or rather the lack of it. It is quite stunning to watch a code-dojo unfold and see how often each group implicitly assumes they are competing against the other groups. Or, that they're not allowed to even talk to the other groups.

CyberDojo

The more I thought about trying to break these implicit assumptions the more attractive the idea of a code-dojo performed completely inside browsers became. So I've written a code-dojo server! It's called CyberDojo.

- CyberDojo currently supports C, C++, C#, Java, PHP, Perl, Python, and Ruby.
- There's a CyberDojo server running at http://www.cyber-dojo.com
- It's open-sourced at http://github.com/JonJagger/cyberdojo so you can build your own server if you want to.
- There's more information about CyberDojo at http://jonjagger.blogspot.com/p/cyberdojo.html

Figure 1 is a screenshot showing the Pandas (each group chooses an animal avatar for their laptop) who are currently at red.

The traffic-light display on the bottom-left indicates their previous two submissions were green and red.

The weakest-link-tv-show-style points-ladder on the right indicates there are at least five groups playing in this CyberDojo; the Pandas are at red, the Bats and the Badgers are at yellow, the Bears and the Alligators are at green. Two greens means 200 points are currently on offer, so if any group clicks 'Bank' the balance will increase from 300 to 500. ■

JON JAGGER

Jon Jagger is a self-employed software coach-consultant-trainer-mentor-programmer who works on a no-win no-fee basis. He likes the technical aspects of software development but mostly enjoys working with people. He can be contacted at jon@jaggersoft.com

This Time I've Got It...

Pete Goodliffe tells us a story of stress, short-sightedness, and solutions.

'Just one more minute,' Jim said. 'I think I really do know what the problem is now. This time I'll fix it.' Julie had been watching him trying to solve the problem for almost a whole day now, with increasing amusement.

Jim had been hunched over the keyboard for hours straight. He'd hardly glanced up. He'd certainly not eaten. And he'd only had the one cup of coffee that Julie had brought mid-morning, mostly out of pity.

It wasn't like him at all. He was a man on a mission.

A sense of urgency, if not mild panic, had been brought about by a 'level 1' bug discovered in the live system. How it had got through the QA process was anyone's guess.

It was thought to be a problem in some of Jim's code, and so Jim sprang into action. It was partly pride that stopped him from asking for help, but there was also a hint of naiveté – he thought he'd have it tracked down in ten minutes, and he would then look like a hero for fixing the running system.

So far that plan had failed.

With every minute that passed, the pressure increased. Reports from customers were trickling in about the problem. One or two reports early in the morning had become a steady stream. Before long that stream would become a flood, and then the whole team would be dumped in it. Indeed, if the problem wasn't fixed soon the company could suffer as a result.

No one wants that on their conscience.

Or their CV.

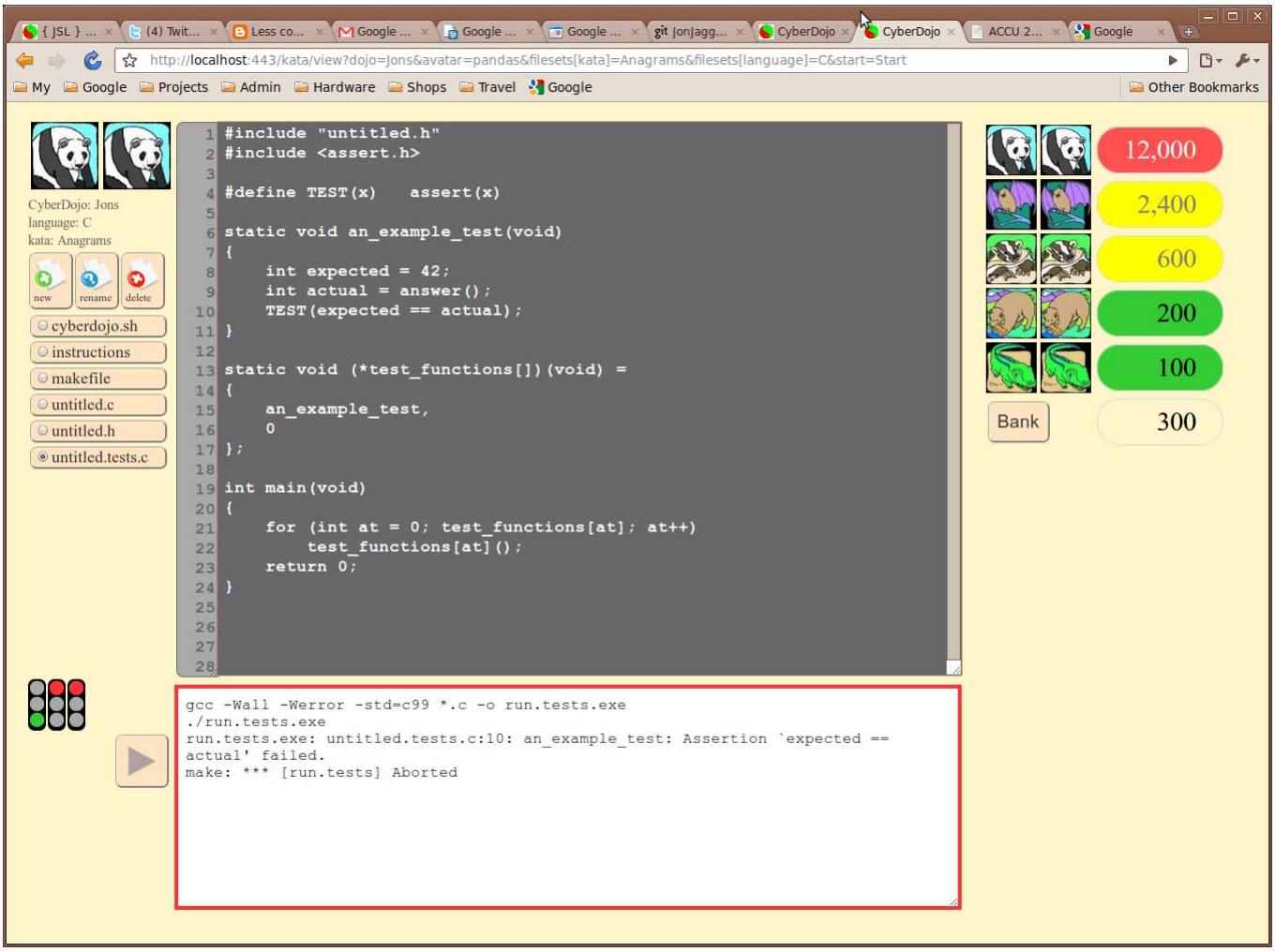
Jim had to get this fixed. And fast. The pressure was building.

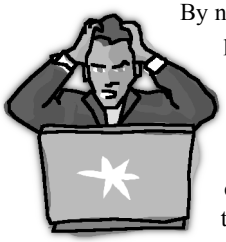
PETE GOODLIFFE

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

What is Code-DoJo? (continued)

Figure 1





By now they surely should have rolled the code back to a previously known good release and taken more time over the diagnosis and the fix, but at every turn Jim assured Julie that he was 'almost there'. And he truly believed it. But each time he got close to the cause of the problem, each time he thought he had it cornered, it seemed to back off into a darker recess of the system.

The problem was clearly now not solely in Jim's code. All his unit tests showed that the module functioned as well as had been expected. No, this was a gnarly integration issue; something strange happening at the boundary of a number of software modules. And it was intermittent, too. The problem was related to some subtle timing or ordering of events flowing round the system.

Jim's prey, like a shy deer, was evading his sights. He just couldn't quite find it.

'I think I know where it is now. It's not in the event dispatcher itself. I think there's some nastiness in the communication between it, the database and the processing backend' said Jim. 'I've got it down to those three components. I haven't fixed it yet, but the next fix really has to work.' He tried to sound more certain than he really was.

'Really? Are you sure?' chimed Julie. There was a hint of mocking in her tone. It wasn't missed. Normally Jim would play along, but he wasn't in the mood today. He gave her the glare that he normally reserved for traffic wardens and turned his gaze back to the scores of source code windows open on his screen.

'If I could just...'

'Wait a minute' interrupted Julie. 'Seriously, just wait. Stop and think about what you're doing.' Her calm voice cut across Jim and he looked up again. He looked tired. And stressed. 'Come on, walk with me to the coffee machine. Tell me what you think the problem is.'

Jim had been thinking. All day. But he did need a coffee, so he acquiesced. He'd been too proud to ask for help; increasingly so as the day wore on. But now he realised that he needed a listening ear and a fresh perspective. He was out of good ideas, and was now operating on educated guesses and adrenaline.

Your first route into a problem is rarely the best

Jim had been too close to the problem. He'd tried every first thing that entered his head without seeing (or yet understanding) the bigger picture. He had started with a preconception of the issue, and hadn't focused on detecting the fault before applying sticking-plasters. Each one was a 'little fix' that should have been the solution, but just masked, or moved the problem around – like smoothing an air bubble trapped under wallpaper.

And he'd spent a whole day doing it. He felt no closer to the solution, and he felt the rest of the team's eyes boring into the back of his head as he worked frantically on a fix.

'Don't worry' said Julie. She had seen it many times over. She'd done it herself enough times in the past. And she knew well enough that she was still perfectly capable of doing it again. 'Tell me what you've found so far.' Jim started to describe the situation.

One coffee and one chat later Jim felt refreshed and had a new focus. As he had explained the full problem to Julie, without her saying a single word, it dawned on him that he'd missed a large piece of the puzzle. As he described what he was going to do next, he realised how he'd not seen the real problem. He described what he'd do instead.

'That makes perfect sense' said Julie encouragingly. 'Do you want me to pair with you?'

'I think I've got this covered now,' said Jim. 'But do come back in ten minutes and check that I'm not going off on one again.' Then he added, thoughtfully 'And, when I'm done, would you mind reviewing the fix?'

'Of course not' said Julie. She smiled.

Jim was like her in many respects. She knew that he'd only learn by making mistakes. At the end of the day, she'd ask him to reflect on what had happened, a little personal retrospective. Hopefully he wouldn't be doing that again in a hurry.

Jim fixed the issue, they reviewed the fix, and deployed it by the end of the day (which was spent celebrating over a drink, when it could have been spent hunched over a keyboard working late into the night).

Desert island development

No developer is an island. Beware the peril of getting so narrowly focused, and so close to a part of the problem that you're not really able to see the whole issue, or to be able to effectively work on it.

Watch yourself. Check whether you're going down a coding blind alley, and make sure that you will notice and can get back out. How can you do this? Work out some practical mechanisms. Set yourself short time limits and deadlines, and review your progress as you go. Make yourself accountable to someone else as you work, either by pairing, reviewing, or informally reporting progress to them.

> Be accountable to another programmer. Review progress with them regularly.

Never be too proud to ask for help. As we saw above, it's often in describing a problem that you'll explain to yourself how to fix it. If you've never tried doing this, you'll be amazed at how frequently it happens. You don't even need to talk to another programmer. It could even be a rubber duck [1].

Stood at the bottom of the mountain

Many software design issues, coding decisions, or bug fixes feel like a huge mountain you have to climb. Running directly to the foot of the mountain and starting to clamber up is often the wrong approach.

Often it is better (i.e. easier, more time/money effective) to approach the mountain in a team. The team can help each other up. One person can point out when another is going to climb into a difficult situation. The team can work together in ways that an individual can't.

It always pays to take a step back first and plan a route before starting your ascent. Round the far side of the mountain there may be a far easier route to get up, if you'd only look for it. Indeed, there may already be a path laid. With signposts. And lights. And an escalator. Your first route into a problem is rarely the best.

> When facing a problem, make sure you've considered more than one approach to solve it. Only then start work.

This is one example, of many, of how software development is often more of a human problem than a technical problem. We have to learn how to enable ourselves to solve problems most effectively, and overcome our natural instincts to solve problems quickly – but ineffectively. ■

Questions

- How effectively do you work with others in your team?
- Can you ask for help, or to discuss problems?
- How often do you 'code yourself into a dead end?' When did you last do this? How long did it take you to notice?
- Are you *accountable* to others? If not, to whom could you become accountable?
- Do you think sharing your progress and discussing problems would make you look like a weaker programmer to others in the team?

References

[1] *The Pragmatic Programmer*. Andrew Hunt and David Thomas (1999) ISBN-10: 020161622X

Desert Island Books

Alan Stokes chooses to take old friends with him.

I have just got back from the first Agile Cambridge and while I was there a prominent ACCU member asked if I was always going to tease (although he didn't use that word) people in my Desert Island Books introductions. I was a little taken as back as I see it as part of the fun and I believe I only tease those people who I know well enough....

Although I have known Alan Stokes for a number of years, first on ACCU general, then at the conference, then at the ISO C++ panel and finally at the banks at Canary Wharf, I do not feel I know him well enough to tease. However, in all the time and right from the beginning I have known him to be a skilled, solid and highly talented developer.

Alan Stokes

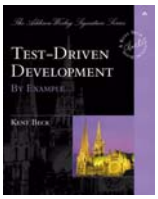
The hard thing about this task of course is not deciding what to put in, but what to leave out. I'm a hoarder by nature, particularly of books. The other difficulty is deciding how to present them; I've decided to go for chronological order, or at least the order in which I encountered these books.

I think I'm supposed to focus on technical books (but I must include a novel), but my first choice is neither. I read (and re-read, and re-re-read) the *Mad Scientists' Club* as a child, long before I'd ever seen a computer. It's a collection of stories about a group of American kids in the Fifties, who apply science (and engineering) to the situations they face. They have access to an impressive array of hardware (radio sets, scuba gear, helium cylinders) and manage both inspired mischief (creating a fake Loch Ness monster, haunting a house) and heroic rescues (correctly working out where a lost pilot must have come down and going to get him). Something in this really chimed with me – some combination of their ability to make things happen and their sheer geeky excellence. Unfortunately I lost my copy many years ago and would probably never have seen it again if it had not been mentioned on Slashdot. The book (and several sequels) are now back in print, and I've had the pleasure of reading them to my two boys. (It'll be interesting to see if this subliminal propaganda has any effect!) I still enjoy the stories, and would take this with me to my desert island for pure escapism, and as inspiration to take charge of my own rescue.

I was very lucky to have access to various microcomputers at school from the age of 12 onwards – initially a Commodore PET (4k of RAM! Addressable screen! Cassette deck!) and subsequently an array of BBC Micros. I'm tempted to take *The PET Revealed* to my desert island (an annotated disassembly of the firmware and general treasure of arcana). I'd happily spend a couple of years discovering how the PET actually worked. (My reaction on encountering a computer program for the first time was a determination to find out how it was done, and to be able to do even better myself. I'm still working on both.)

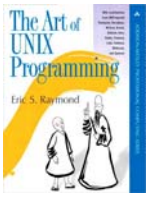
The book I will include from that part of my life is *Compiler Construction for Digital Computers*. This was the first book I'd read that covered how large programs could be put together. At that stage I'd dabbled in a few languages beyond Basic and 6502 assembler – PL/1, Fortran, even a strange thing called C – but was still more interested in how the compiler worked (and how to make a better one) than in anything else. This book was a revelation – there were data structures, algorithms, formal language and parsing theory. It was also amazing in that it was clearly written using, and printed by, a computer system – which nobody else seemed to have thought of at that time. (While writing this article I've discovered that the original punched card deck containing the book's source and its formatting program is on display at Stanford University, which is wonderful.)

After this historical interlude I'm going to leap forward a long way, through my early professional career. There's a few books I'm tempted to include – Knuth's *Art of Computer Programming* (I read volumes 1–3 a long time ago, and really should re-read them and the newer fascicles), or Bruce Schneier's *Applied Cryptography*, which brought some depth to my long standing interest in codes and ciphers. (I'm really looking forward to his talk at the autumn conference.) I wish I'd read the *Pragmatic Programmer* in those days – by the time I did so I'd learned a lot of its lessons the hard way, through doing it all wrong. And *Design Patterns*, of course – that had a massive impact on how I thought about code, but I don't think I'd take it with me – it's much more of a starting point than the final word. And then there's *Refactoring* and *Refactoring to Patterns*, which helped me to see how to incrementally change code and make its design better, but I don't feel a need to re-read them.

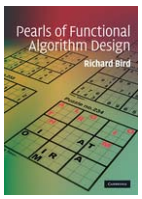


However, my next choice has to be Kent Beck's *Test-Driven Development By Example*. I've read other good TDD and Agile books since (e.g. anything by Uncle Bob Martin, Steve Freeman and Nat Pryce's *Growing Object-Oriented Software*), but Beck's book was the one that changed my life. It's not a big book; I devoured it in a couple of days. And I realised that I'd been doing it completely wrong all those years; how could I have been so stupid? Equally clearly, from now on I'd be writing and designing my code using TDD, because I wanted it be correct, clear, and elegant (at the small scale to the large – great programs exhibit 'fractal attention to detail', as DadHacker put it). And because it was obviously the only way that made any sense.

Not very long after that revelation I changed jobs and found myself learning many new ideas and perspectives. I was very lucky to be working with an extremely talented programmer who opened my eyes to more of those simple and compelling ideas that are obvious in retrospect. As a nod to that period my next book is *The Art of Unix Programming*. This is highly opinionated, deeply partial and thoroughly argumentative. It also illuminates many profound truths. I learnt some good techniques – threads should be the last resort not the first, the immense power of plain text – but more importantly it changed my attitudes to Unix, open source and programming in general. This is a book I'd like to re-read, and it'll give me something to argue with while I'm alone on my island.



My final technical book is a difficult choice. I want to include something with a functional perspective. I've always been interested in functional programming – I chose to go to Imperial College as an undergraduate partly because they had a strong functional focus. I've not yet written serious code in a functional language, but I have dabbled, and the functional perspective has influenced my style for years. I could go for a language book (I've read books on Haskell and Erlang, and would like to branch out into Scala or Clojure), but I think it'd be better to have something on the underlying ideas. I'm tempted by *The Fun of Programming*, a joyful festschrift for Richard Bird. This has some wonderful applications of functional programming (pricing options, describing musical notation) and some very high-level abstract constructs that I might even understand after a few years study on the island. But I think I'll take a gamble on a book I haven't read yet but have had highly recommended by someone whose judgement I have great respect for – Bird's own latest book, *Pearls of Functional Algorithm Design*. (That also means I have



Inspiration (P)articles

Frances Buontempo continues her quest for positive experiences.

Having spent years trying to make his Windows PC behave like Linux, by using Gvim and unix tools, our esteemed editor, Steve Love, blats Windows in favour of Linux and then tries to make his machine behave like a Windows box.

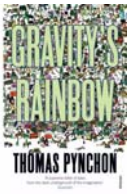
I've been playing around with various flavours of Linux for several years, but never really delved that deeply into it until I recently decided to go head-first and make it my primary platform at home. I still want to be able to do C# development, and Mono (www.mono-project.com) looks good. One of the aims of Mono is to allow compatible binaries built for .Net to also run natively under the Mono runtime without recompiling. I decided to try three such libraries: the MySQL .Net connector, NHibernate and xUnit (a lightweight NUnit-like facility available from xunit.codeplex.com). I also wanted to try out the relatively new support for Linq in NHibernate. Linq is part of the .Net 3.5 framework, and is supported in the latest release of Mono. In spite of the very sparse documentation for Linq support in NHibernate, when I eventually managed to get my code to compile, it did indeed "just run" against my database with NHibernate. Furthermore, the xUnit GUI program ran my tests and (eventually) gave me the green bar! I am very new to the magic of NHibernate, and so getting this to run – with Linq support – was quite a breakthrough. That Mono just did the right thing with .Net binary downloads, including the GUI of the unit test framework, is a testament to the sterling job the Mono developers have done. I am not a Windows or Microsoft hater (or lover, for that matter), but I've found it inspiring being able to do many of things I've got used to in Windows without being tied to one platform. And, I'm looking forward to learning how to do the rest of them.

If you've read something in C Vu that you particularly enjoyed, you disagreed with or that has just made you think, why not put pen to paper (or finger to keyboard) and tell us about it?

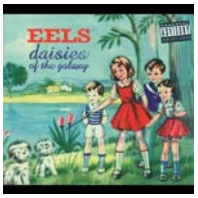
Desert Island Books (continued)

something specifically on algorithms, which I seem otherwise to have neglected from my choices.)

Right, we can stop being technical now. I need to choose a novel. I find it impossible to choose a single book from all the ones I've read (I'm tempted by my oldest favourite, *The Hitchhiker's Guide to the Galaxy*, but I can nearly quote that from memory so there's no need to take it). Instead I'm going to take a punt on a book I haven't read yet. I read Thomas Pynchon's *Mason & Dixon* (The inspiration for Mark Knopfler? - Ed) a few years ago – it took me over a year to finish it, and was hard work but rewarding. My working assumption is that I will be stuck on the island for some time, so a book that will take a fair bit of reading seems a good plan. So I'm going to choose the same author's magnum opus, *Gravity's Rainbow*.



There have been times in my life when music was very important to me. I like to think of my taste as eclectic – it sounds better than disorganised or old-fashioned. So I'm going to simply pick some music that has meaning and associations for me. My first choice is *Essential Fats Domino*. I love all of the songs on this; 'Blueberry Hill' always sends shivers down my spine. My second choice has to be something by the Eels – and perhaps the best would be their atypically optimistic album *Daisies of the Galaxy*.



I'm not going to choose a luxury – the choices all seem too obvious (internet connection, small but well-stocked and -staffed kitchen, nuclear submarine, the final C++0x standard). But I should like to have with me a photo of my wife Mandy and sons George and Ben, who have always managed to put up with me with panache.

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.

Next issue: Rachel Davies.

Code Critique Competition 66

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

State a problem together with a solution and ask for improvements (which, in some cases, might lead to a full rewrite due to taking a different view or to generalising and in others to the conclusion there's no way to improve it).

The problem: Print all pairs of indices that bracket the maxima in a circular list of numbers.

E.g.

```
0 1 2 3 4 5 6 7 8 9 <- indices
2,1,4,3,5,6,6,6,1,1 <- numbers
^   ^     ^ ^ ^     <- maximums (three 6s
                       count as one).
(9,1) (1,3) (4,8)   <- brackets (output in any
                       order).
```

No doubt this can be made more efficient. Simply, after we print a pair we could set `j` to `k`, couldn't we? Well, we'd have to detect the case where `k`'s been reset to zero and, if so, return immediately. I guess `k`'s only going to be reset half the time, so we'll be setting a `k_was_reset_flag` to no avail half the time, so is it really more efficient? It certainly makes the code more complicated – less easy to follow.

Maybe it's better to write two functions, one brackets the maximums in a non-circular list and the other looks for a wrapped maximum?

The code is in Listing 1.

Listing 1

```
// The code to improve
void bmax(int const a[],int const n)
{
  int i = n-1;

  for(int j = 0; j != n; ++j)
  {
    if (a[i] < a[j])
    {
      int k = j;

      do
      {
        if (++k == n) k = 0;
      } while(a[j] == a[k]);

      if (a[j] > a[k]) print(i,k);
    }

    i = j;
  }
}
```


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

Critiques

Huw Lewis <huw.lewis@hlsoftware.co.uk>

Efficiency?

As the problem itself suggests, the obvious inefficiency is the way the algorithm finds a maxima, then continues its search from the incremented index rather than the right hand side of the maxima previously found. This isn't difficult to correct and doesn't leave much else to optimise. However, one is left with the feeling that it could be better.

Re-factor for cohesion

The function provided is not very cohesive i.e. it is doing more than one thing: managing a search through a circular buffer, finding maxima and printing them. It might be better if there was a simpler function to find the next maxima in the sequence from a given point and fail if it reaches the end of the buffer.

Following this line of thought the result is a couple of small simple helper functions (`incrementIndex` and `previousIndex`), a simpler `bmax` algorithm (renamed to `findNextMaxima`) and a simpler 'application' function `printMaxima` that uses the `findNextMaxima`.

In order to pass the result of the maxima detection out to the calling function, I have defined a maxima as `std::pair<int, int>` to represent the indexes either side of the maximum value. The function findNextMaxima works by looping through the array from a 'from' index to the end looking for an increase in value. Once found the next value transition type must be a decrease in order to be a maxima.

```
// Rather than find all maxima, the function should
// find the next maxima
typedef std::pair<int, int> Maxima;

bool findNextMaxima(const int* a, const size_t
  len, Maxima& result, const int from = 0)
{
  bool b_success = false;

  int i = from;
  int prevIndex = previousIndex(from, len);
  while ( i < len && !b_success )
  {
    // See if this element is an increase...
    if (a[prevIndex] < a[i])
    {
      // Search for the next change of value
      int nextChange = incrementIndex(i, len);
      while (true) // loop cannot be infinite
           // due to increase detected above.
      {
        if (a[nextChange] != a[i])
          break;

        nextChange = incrementIndex(
                       nextChange, len);
      }

      if (a[nextChange] < a[i])
      {
        // the next change is a decrease -
```

```
        // this is a maxima!!
        result.first = prevIndex;
        result.second = nextChange;
        b_success = true;
      }
    }

    // move loop indexes
    prevIndex = i++;
  }
  return b_success;
}

void printMaxima(const int * a, size_t len)
{
  Maxima result(-1, -1);

  int from = 0;

  while (from < len &&
      findNextMaxima(a, len, result, from))
  {
    print(result.first, result.second);

    // move search index (from) to the end
    // of the maxima
    from = result.second + 1;
  }
}
```

A generic algorithm?

The previous solution provides a more readable and flexible implementation, but what we have there is specific to arrays of integers. This could be re-factored again to provide a generic solution for any unsorted sequence (vector, list, queue etc) of any type that has a usable **operator<**.

Furthermore, the solution can go further by replacing the **operator<** with a template argument predicate functor. Now the algorithm finds not only maxima but minima also, and custom predicate types can be used for detecting maxima/minima in application specific class types.

I have framed my solution in a class template so as to provide a container for typedefs and helper functions. The class **CriticalPointDetector** takes 2 template arguments:

- **C** – a STL container type (unsorted) e.g. **std::vector<int>**, **std::list<double>** etc.
- **Pred** – a predicate functor type to determine the 'less than' or 'greater than' characteristics of the contained type (**C::value_type**).

Use **operator<** semantics for finding maxima

Use **operator>** semantics for finding minima

I then provided 2 simple template classes **MaximaDetector** and **MinimaDetector** that inherit from **CriticalPointDetector** using the appropriate **std::less** or **std::greater** predicates. Finally, the **printMaxima** function manages the loop around the buffer and uses the provided facilities to find and print maxima. It is trivial to write a corresponding **printMinima** function using the **MinimaDetector** template class instead of **MaximaDetector**.

The full solution is given below. It is certainly true that the original code was a lot smaller, but did it provide any hope of re-use?

```
// Template Parameters:
// C - the unsorted sequence type e.g.
//   std::vector<int>
// Pred - the predicate that is used to search
// for maxima or minima
//   i.e. std::less for maxima,
//   std::greater for minima
template< class C, class Pred >
class CriticalPointDetector
{
public:
  typedef typename C::iterator iterator;
  typedef typename C::const_iterator
    const_iterator;

  // A critical point is defined as a pair of
  // iterators outlining the point in the
  // sequence as [ iterator before CP,
  // iterator after CP]
  typedef std::pair<   const_iterator,
    const_iterator > CriticalPoint;

  static bool findNext( const C& seq,
    const const_iterator& from,
    CriticalPoint& result )
  {
    bool b_success = false;
    const const_iterator endIt = seq.end();

    if (seq.empty())
      return false; // EARLY RETURN

    const_iterator prevIt(
      previousIt(seq, from));
    const_iterator it(from);
    while (it != endIt && !b_success)
    {
      // If this is a hit...
      Pred pred;
      if ( pred(*prevIt, *it) )
      {
        // Search forward for the next value change
        const_iterator nextChange =
          findNextChange(seq, it, pred);
        if (nextChange != endIt &&
            pred( *nextChange, *it ) )
        {
          // this is an anti-hit, and
          // therefore is a critical point
          result.first = prevIt;
          result.second = nextChange;
          b_success = true;
        }
      }
      // move iterators on for next loop
      prevIt = it++;
    }
    return b_success;
  }

private:
  // return the previous iterator to it
  static const_iterator previousIt(
    const C& seq, const const_iterator& it)
  {
    const_iterator result = seq.begin();
    int n_distance =
      std::distance(result, it);
    if (n_distance > 1)
    {
      std::advance(result, n_distance - 1);
    }
    else
    {
      // the distance is 0 - we need the last
      // iterator
      std::advance(result, seq.size() - 1);
    }
```

```
    return result;
  }

  // return the next iterator
  static const_iterator nextIt(const C& seq,
    const const_iterator& it)
  {
    const_iterator result(it);
    ++result;
    if (result == seq.end())
    {
      result = seq.begin();
    }
    return result;
  }

  static const_iterator findNextChange(
    const C& seq, const const_iterator& from,
    const Pred& pred)
  {
    const_iterator result = nextIt(seq, from);
    size_t count = 0;
    const size_t seqSize = seq.size();
    while (count++ <= seqSize)
    {
      if ( pred(*result, *from) ||
          pred(*from, *result) )
      {
        // a change is detected
        return result;
      }
      // go to next iterator
      result = nextIt(seq, result);
    }
    // the entire sequence has been looped
    // through and no changes detected.
    throw std::logic_error("No changes "
      "detected in sequence values");
  }
};

// Define a shortcut to the maxima detector
template < class Seq >
class MaximaDetector
: public CriticalPointDetector<Seq,
    std::less< typename Seq::value_type > >
{
};

// Define a shortcut to the minima detector
template < class Seq >
class MinimaDetector
: public CriticalPointDetector<Seq,
   std::greater< typename Seq::value_type > >
{
};

template< class C >
void printMaxima(const C& seq)
{
  typedef MaximaDetector<C> MaximaFinderType;
  typename MaximaFinderType::CriticalPoint
    result;

  typename C::const_iterator from
    = seq.begin();

  while (from != seq.end() &&
      MaximaFinderType::findNext(seq,
        from, result))
  {
    print(std::distance(seq.begin(),
        result.first),
      std::distance(seq.begin(),
        result.second) );
    // move search index (from) to the end of
    // the maxima
    if (from < result.second)
    {
      from = result.second + 1;
    }
    else
    {
      from = seq.end();
    }
  }
}
```

Commentary

This was an interesting problem to consider as the code looks inelegant and a bit inefficient. However I was a little disappointed to only receive one entry: perhaps the example was harder that it looks to improve.

If you have an algorithm of your own that could do with improvement please consider sending it in for a future CC!

The first question I had was whether the code was really that inefficient. I knocked up a quick test harness to populate an array of items of various sizes and test the algorithm against them. I was able to process around 20,000 arrays with 1,000 elements in a second, so it didn't seem to be an obvious performance bottleneck; although this would of course depend on the actual use.

I tried the same dataset using the generic solution given above and the performance dropped significantly: about 6 times slower! (I didn't try the non-generic solution as the two helper functions weren't provided.)

What has happened? It's hard to tell without doing more analysis, but it seems that the algorithm might be more generic and clearer to read, but at the expense of efficiency. This may or may not matter in practice, but it doesn't really address the problem setter's brief!

It is always important when addressing performance to keep (at least) two key things in mind: firstly that you need to measure and secondly that performance usually depends heavily on usage.

The Winner of CC 65

There was only one entrant, so Huw gets the prize.

For this issue I've set another code critique with a program that presents a couple of problems with implementation but also one that raises some questions about design too.

Late critiques for CC64

I received a late submission for CC64 ('I'm trying to write a simple quadratic equation solver for the equation "a * x * x + b * x + c = 0" that writes output to a file but I am having problems getting it working. It's OK for some inputs but I'm having problems, in particular with equations that have no (real) solution.'). I've added an editorial comment (Ed) in a couple of places.

Paul Floyd <Paul_Floyd@mentor.com>

1. Original code won't even compile. Poor show!

   ```
   CC -library=stlport4 -o cc64 cc64.cpp
   "cc64.cpp", line 32: Error: The function "sqrt"
   must have a prototype.
   "cc64.cpp", line 33: Error: The function "sqrt"
   must have a prototype.
   ```

   Error(s) detected.

   `sqrt` should be in the `std` namespace.

Ed: This is true, but the code does compile with two different compilers (msvc and g++). One of the real world problems with trying to write portable code

is that different ways implementers of the library use to pull in the C runtime functions means sometimes they leak into the default namespace.

2. Problem hopelessly underdefined. Are real-valued quadratics the only valid input or are complex quadratics allowed as well? Are only real solutions valid, with complex solutions producing a warning/error message, or are complex solutions allowed? If so, should they be displayed as cartesian or polar values? Let's guess real quadratics, complex solutions allowed and displayed as cartesian values.

   Ed: It may simply reflect the business domain I've worked in for so long: almost all the work I do is underdefined.

3. No error handling for errors reading **`std::cin`**, errors reading `file.dat` or writing `file.dat`.
4. **`tofile`** writes using default precision of 6 digits. verify then performs comparison at machine precision (~16 digits). I would recommend using **`std::setprecision(16)`** when writing to verify.
5. That comparison again. This needs to have some sort of tolerance; absolute, relative or both. Go with this advice:

   http://www.boost.org/doc/libs/1_32_0/libs/test/doc/components/test_tools/floating_point_comparison.html
6. I don't like camel case **`fromFile`** and flat case **`tofile`**. Use the same coding standard.
7. No validation that the roots work back to the original coefficients.

Code Critique 66

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

This issue's problem is also a bit of a design critique. I've got a 'log' function (an externally provided function I can't change) that takes a **`char const *`** argument and I want to wrap it in a C++ layer so I can stream to it. The function itself is thread-safe and I want to be able to use streaming in multiple threads.

My approach here is to use a temporary helper object containing an **`ostringstream`** and build up the string in there. The helper object is created when the streaming starts, and is passed along the streaming operators until the end of the statement when it is destroyed. The destructor of the helper object passes the contents of the **`ostringstream`** to the log function. It seems to *nearly* work, but I'm getting some odd characters in the output. I found that adding an & (where it says: `/* Needed?: & */`) seems to fix it, but don't know why. Are there any problems with this approach – or better ways to do the same thing?

(To give a bit of background, following a recent discussion in the ISO C++ standards meeting about destructors that throw exceptions, I was keeping a look out for examples of code where work was done in the destructor. That provided the initial input to this example, but it also raises questions about the lifetime of temporaries and the order of their destruction. Finally the code in the critique compiles with visual studio but not with g++ – making the change mentioned to add an ampersand fixes the compilation problem, so we have a compiler difference to consider too.)

The header file `log_wrapper.h` is shown in Listing 2.

Listing 3 contains an example of its use: `test_log_wrapper.cpp`

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

Listing 2

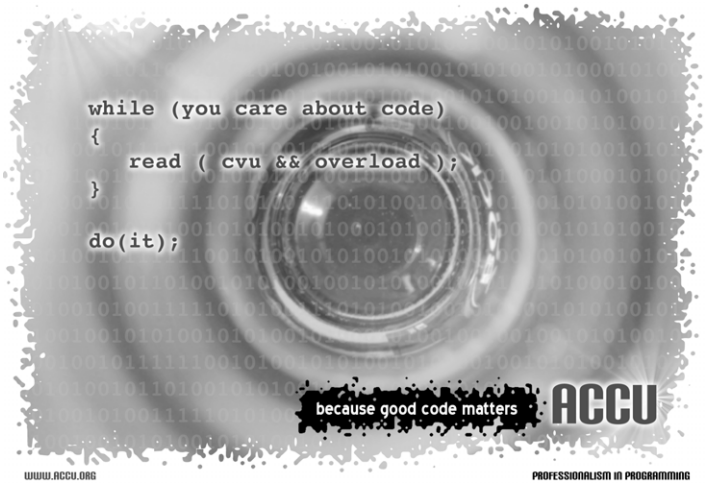
```
#include <memory>
#include <sstream>
// third party log function
extern void log(char const *);
// Add C++ style streaming to the log function
class logger
{
public:
  class helper
  {
    friend class logger;
  public:
    template <typename T>
    helper /* Need & here? */ operator<<(T t)
    { *oss << t; return *this; }
    helper() : oss(new std::ostringstream) {}
    // When helper destroyed log the message
    ~helper()
    {
      if (oss.get())
      {
        log(oss->str().c_str());
      }
    }
  private:
    std::auto_ptr<std::ostringstream> oss;
  };
  template <typename T>
  helper operator<<(T t)
  { helper h; return h << t; }
};
```

Listing 3

```
#include "log_wrapper.h"

// Dummy 'log' for testing
void log(char const *p)
{
  puts(p);
}

int main(int argc, char **argv)
{
  logger l;
  l << "Starting";
  l << "Testing number: " << argc;
  l << "Testing string: " << argv[0];
}
```



Regional Meetings

A round-up of what's happened at an event near you!

Agile Cambridge 2010

Paul Grenyer outlines the programme at a regional Agile conference.

Mark Dalgarno, with the support of Redgate Software, is using Software East to do great things for the software community in Cambridge. Software East has been running frequent evening events on topics including Agile and iPhone development and attracted a number of ACCU speakers including Allan Kelly, Pete Goodliffe and everyone's favourite Mac pusher, Phil Nash. Not content with rivalling ACCU London's events, Software East has gone further and put on their own Agile conference over two days in the heart of Cambridge. I was pleased to be asked to speak and enjoyed both thoroughly exhausting days. What follows are some of the highlights for me.

Agile Cambridge was the first conference I have attended as a Twitter user. It's an amazing tool for not only communicating with the the attendees and passing on words of wisdom from one session to another, but also for keeping those who could not attend up-to-date and whetting their appetites for next time.

I do not know what it is about the last day of a conference, and it does not seem to matter how long or short the conference is, but I always leave completely spent on that last day. Maybe it is the activities of the night before the last day? Agile Cambridge was no different and I cannot wait to go again next year!

Test Engineering at Google

by James A. Whittaker

James Whittaker is the second incredibly enthusiastic and charismatic tester I have had the pleasure of seeing speak this year, following James Bach at ACCU 2010. He took us through the technology of the last twenty years, including a picture of Michael Douglas on a phone about the size of a small loaf of bread and demonstrated how things have changed. He seemed very surprised that a lot of people in the audience still used a paper phone book and this became the subject of a number of jokes throughout the presentation. One of the main points James made was that with a lot of software now being web based, it is no longer cheaper to fix bugs in a product before it ships. He went on to describe the role of a tester as a consultant in a hospital and told us that all software is broken and on life support until the testers can get in and diagnose the diseases.

What Does It Take to be an Agile Company

by Allan Kelly

I have seen Allan Kelly speak a few times and have always enjoyed his direct style. He passionately believes in what he is doing and I enjoyed this presentation as well. I think Allan has a very complete view on what agile is, where it came from and where it is going. He explained that he believes that Agile has arrived and is here to stay and that it will only get better. I am very much inclined to believe him. Another of the stand out points for me was that companies should use experimenting over planning and make sure that if and when failure comes, they fail fast and fail cheap. This became a much repeated line throughout not only the presentation, but the whole conference.

Building Effective Habits Through Peer Group Learning and Assessment

by Jason Gorman

Jason Gorman has been doing some very interesting work. I think it is best summed up when described as: coaching Test Driven Development (TDD) at the BBC. I still find it amazing that some developers still do not see the clear benefits of TDD and have to be encouraged with incentives. Jason Gorman has been at the BBC teaching and assessing those who have been incentivised and it sounded like very interesting work! He described some of the exercises that the developers went through, their assessment and certification and some of the problems he faced. One of the techniques used was pair programming, where developers would mark off on a sheet which TDD rules their partner had broken, in order to help them learn and improve. At the end of the session he wrote some code and unit tests while the audience marked off the rules he had broken on a similar sheet to demonstrate how it worked. Of course he broke (almost) every rule. This was another very enjoyable session.

Five Years of Change, no Outages

by Steve Freeman and Andrew Jackman

Again, I have seen Steve Freeman talk a couple of times before and he has a lot of very interesting things to say. In this presentation Steve and Andrew Jackman described a very successful project that they both worked on at different times. It solved a particular problem that had been attempted by different teams in the past who had failed. The main secret to the project's success was the use of Agile methods, including regular and automated deployments. They described a lot of the methods I am using every day and I am again amazed that more teams do not see the obvious benefits of regular automated deployments. Maybe this is why this sort of conference is needed so much.

Cyber-Dojo

by Jon Jagger

I attended Jon Jagger's Cyber-Dojo as a participant at ACCU 2010. On this occasion I was delighted to be asked to help out alongside Jon. The setup had changed significantly as Jon has been developing his Cyber-Dojo since the last time. There was a greater choice of languages and problems to solve; and the teams were not given an objective until part way through. What was most interesting to me was the teams being told that their only objective was to get every team's build to green and the number of them that then proceeded to break their build after having reached green, before the others had caught up. These Cyber-Dojos get more and more interesting and I'm looking forward to the next one.

Building Trust in Agile Teams

by Rachel Davies

This was one of most enjoyable presentations, not least because I got to take part. Rachel spoke about how important it was for members of Agile teams to trust each other and the ways in which trust could be gained and lost. Rachel has spent many years coaching Agile teams and had a lot of very useful things to say. They certainly made me think about my team in new ways. Unlike ACCU conferences, the Agile Cambridge bunch took a while to start interacting with the speakers. There was plenty of interaction throughout Rachel's presentation and a whole lot of laughter. I am very much looking forward to seeing Rachel speak again soon.

Creating a Development Process for Your Team: What, How and Why

chaired by Giovanni Asproni

The final session at the conference was a panel. I really enjoy panels, both as a member of the panel and a member of the audience. You never know quite where it is going to go and by their very nature there is lots of interaction between the audience and the speakers. Those who joined Giovanni on the panel included Allen Kelly, Steve Freeman, Rachel

Davies and Nat Pryce. It was a chance for the audience to ask about some of the recent Scrum bashing and get to the bottom of the concerns some of the industry experts have with it. There was a long and very interesting discussion on pair programming and another on how to convince an organisation that Agile was the way to go.

ACCU London – September 2010

Chris Oldwood shares his experience of a real code-dojo.

Once again I found myself at one of the London offices of SkillsMatter for an ACCU London get together. This time it was a Cyber-Dojo hosted by Jon Jagger. I didn't know exactly what one of those was, but there were good reports about the one he ran at the ACCU Conference this year so that made it sound promising. Also there had been some chatter on accu-general about the system he was building for it that intrigued me.

we all know that Lines of Code is no measure of success and what really mattered was the team dynamics

The basic premise is that we were initially split into groups of three to solve a simple programming task. There would be one person allowed to use the keyboard (the driver) and two aides. After ten minutes Jon would ring a bell and the driver had to move to another group, but he wasn't then allowed to drive – one of the remaining two had to take up the role. Due to the time constraints of the event this part only lasted an hour with a few minutes afterwards to discuss what happened. Naturally a few of us retired to The Slaughtered Lamb to do some Pair Drinking.

I'll be honest and admit that I was somewhat nervous at the thought of Pair Programming with a bunch of ACCU members. If there is one thing I have come to realise since joining the ACCU it's how high the bar really is and I probably would have felt somewhat intimidated if some of the bigger names were there. In contrast I barely knew any of the 10 or so people that turned up and that made it all the more exciting as I had some unexpected social barriers to contend with instead. Still, rather than throw caution to the wind and join two other new faces I spotted Allan Kelly was also a man down and so took the safe approach and joined his group instead.

The setup was quite simple. Each group had a laptop with some custom browser based software [1] that provided a simple text window into which you could write either unit tests or production code. Once you were happy with each you hit a button and the code would be compiled and the tests executed. You would then either get a green or red light. Sadly I arrived late and missed the explanation about how the groups were connected behind the scenes, but there was some kind of gaming system under the hood that turned your actions into a pot of money that gave you an additional goal.

The programming problem itself was quite simple – format up an integral value into Roman Numerals. It's the sort of problem you would expect to be pretty easy, at least if you've been through a round of job interviews in the last few years; but it quickly becomes tricky. I'm not sure if it was stated as a goal, but each group took a Test-First Development (TFD) approach and wrote small failing tests before adding new production code. The choice of initial tests was quite interesting as some started with the low numerals, I, V etc, whilst others picked the opposite end, M, D etc. It didn't take long for the easy ones (1, 5, 10 etc) to be working and the teams to stumble on the key difficulty with Roman Numerals – how to deal with the numbers just less than the main numerals, i.e. 4 (IV), 9 (IX) etc. At this point, for some, the 'simplest thing that could possibly work' changed from a simple step to a much larger prospect as the teams started to discuss algorithms to generalise the solution; others seemed happy to keep solving each little case and try and continue taking small steps.

The 10 minute rotation now became more of a distraction as you were in the middle of a 'design meeting' when you found yourself needing to stop and bring the newest member of the team up to speed, or you became the outsider and had to integrate with a new order. Near the end word spread of a fresh lead involving special cases of numeral pairs and some members were trying to explain their findings to their team mates when Jon called time. Throw into the mix a subtle change in programming language (Java/C#) and the fact that The Driver may not even have written code in one of these curly brace languages, or even be used to the Imperative paradigm for that matter and you can see why so little production code was written in those short 60 minutes.

Of course we all know that Lines of Code is no measure of success and what really mattered was the team dynamics. For example one team was initially isolated in the opposite corner of the room and no one noticed when it came time to swap – so they were left with the same pairing until Jon stepped in. Surprisingly there were no big egos, if anything people were all a little too polite; perhaps that's a symptom of a largely anonymous group whereas if we all worked together some of us would have felt more comfortable taking a more leading position.

Perhaps if more teams switched one of their team-building events from paint-balling or bowling to one of Jon's Cyber-Dojos they might build more than just morale.

Note

[1] For some insight into the "browser-based software" mentioned, be sure to read Jon Jagger's article on page 9.

Bookcase

The latest roundup of book reviews.

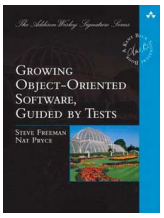
If you want to review a book, your first port of call should be the members section of the ACCU website, which contains a list of all of the books currently available. If there is something that you want to review, but can't find on there, just ask. It is possible that we can get hold of it.

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-glamourous '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)

Growing Object-Oriented Software Guided by Tests



By Steve Freeman and Nat Pryce, published by Addison-Wesley, ISBN-13: 978-0321503626

Reviewed by Alan Griffiths

Highly recommended

I've known Nat and Steve for some years through the 'Extreme Tuesday Club' and various conferences. They are a significant part of the UK's community of developers interested in advancing the state of the art (or craft, or science according to your prejudice). We've also worked at the same client: their team was one of the users of the software my team produced and gave some of the clearest and most constructive feedback on our product that I've had.

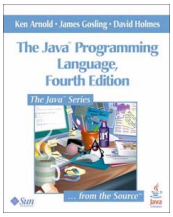
Having discussed the issues of software development (and other less relevant topics) with them over this time it was nice to see that they've found time to produce a book presenting their vision of software development. It was also nice to see that they discuss many of the issues we all encounter when trying to put such a vision into practice – I have to admit they've been more successful than I have in addressing some of them. This book inspires me to try harder!

The book itself covers the motivations for incremental, test driven development and the technical and people issues encountered in applying it. This isn't just about unit test, or system tests it is a consistent vision of testing all the way down from system features to class interfaces. There is a worked example illustrating both the process in miniature and the tools (they use their own JMock along with JUnit). It is a well chosen example in that it is easy to understand but also incorporates many of the issues that exist in real developments and presents solutions that ought to scale.

Potential difficulties are not ignored and are presented with the air of someone that has encountered them and knows the pain they can cause. The solutions offered are often of the kind that are obvious – but only after someone else points them out. Many times I found myself thinking 'that is so cool!'

This is one book I'll leave lying around hoping my colleagues will borrow it.

The Java Programming Language



By Ken Arnold, James Gosling, David Holmes, published by L Prentice Hall, ISBN: 978-0321349804

Reviewed by Paul Grenyer

I decided to read this book about 18 months after having to learn Java in a hurry after discovering what I had been lead to believe was a C# role turned out to be a Java role.

Despite several years of programming in C++ and C# I figured there must be lots of stuff that was different in Java and that this book would be a good way to find out. The first chapter is a 40 page general introduction to Java and I found it such a dry read that I gave up. Then, twelve months later, I decided I really should read it cover to cover to find out what I was missing and it took me five months to do it!

Although solid and reasonable well written the book is a dry read most of the way through. The chapter on streams is especially hard going. Although in many cases each method of a class being discussed is described in detail, this book represents a medium level (as opposed to high or low level) introduction to the language. As you would expect it covers classes, interfaces, inheritance, polymorphism, enums, control flow, generics, threading, reflection, etc. so you do get a good overview of the language. However it does not go into any detail about how to compile or run Java programs, neither does it mention ANT or describe how to create JARs. If you want a practical tutorial, this is not the book for you.

I didn't agree with a lot of the 'good coding style' suggestions such as using a fall through comment in switch blocks to tell the reader of the code what he should already know the language does or using continue in a bodiless for loop as a semi-colon may get deleted or forgotten. Many people, including myself and the creators of the Spring library, believe that checked exceptions are a bad thing in most cases and should not have been added to the lanuguage, so I find bad advice like 'Nearly all exceptions you create should extend Exception, making them checked exceptions' appalling. There are a number of examples of poor practice throughout the exceptions chapter.

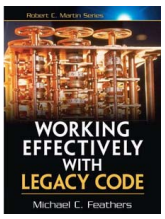
Overall I think most Java developers would benefit from reading this book just to plug a few of the inevitable gaps in their knowledge. It is hard going, but worth it. As it covers Java 1.5 and Java 1.7 is nearly upon is, it would certain benefit from an update and a review of good programming practice.

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

Working Effectively with Legacy Code



By Michael C. Feathers, published by Prentice Hall, ISBN: 978-0-13-117705-5

Reviewed by Paul Floyd

Recommended

Working with code in practice means refactoring it. According to Feathers, the key to refactoring is getting it into unit tests. He's well placed to have such an opinion, being the original author of CppUnit. The purpose of the unit tests is not so much to prove correctness, Feathers argues, but more to ensure that there are no changes to the behaviour. The emphasis is on C++ and Java, with one chapter covering non-OO languages, specifically C.

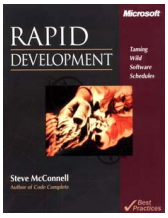
In the first part of the book, Feathers describes two fundamental issues with unit testing legacy code: getting the code to compile and link, and reading values in some way so that you can detect any changes. He calls these seams and sensing. Part I also covers some of the main tools that can be used (harnesses and mock-object libraries).

Part II then goes into the details of techniques that can be used for seams and sensing. Each chapter covers a class of problem, containing sections in a similar style to *Design Patterns*, with a catalogue of methods to solve the problems. As an example, Chapter 10 'I Can't Get This Class into a Test Harness' deals with parameters that are difficult to instantiate, hidden dependencies, breaking up large constructors and global dependencies. There are a couple of chapters covering more abstract ideas for getting to learn what legacy code does. The last part of the book is a catalogue of 24 techniques for breaking dependencies.

I found that Feathers treads the balance between theory and practice well. He could have regaled, or bored, us with endless tales from the trenches. In fact he spares the blushes of his clients, all the example code is made up for the book.

If you are working with legacy code and trying to get it under unit test, then you are likely to benefit from at least some of the ideas that this book describes.

Rapid Development



By Steve McConnell, published by Microsoft Press, ISBN: 978-1-55615-900-8

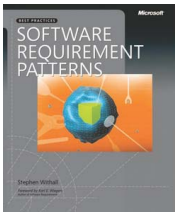
Reviewed by Paul Floyd

I'll start out with the bad points. This book was written in 1996, so it predates the start of the Agile/XP movement. If you're a fan of Agile, then you probably won't want to read this book, which is more of a 'classic process' book. Another bad thing, for me, was that I've already read a few of McConnell's books (*Software Project Survival Guide*, *Professional Software Development* and a long time ago the 1st edition of *Code Complete*). There's a fair bit of overlap, which didn't help me keep my concentration up. The most annoying part of the book was the little fictional anecdotes. Basically these are all along the lines 'Ken used my methods, and the project was a success' and 'Joan didn't use my methods, and the project was a failure'. I don't know what purpose they serve.

McConnell is clearly well read. The bibliography for this books runs to 14 and a bit pages. The book is in 3 parts. The first is the shortest, and covers the basics – high level strategy, errors, fundamentals and risk management. The second part is where the meat of the book is. It covers all direct and indirect aspects of projects, from planning to project recovery for projects that fall off the rails. This is where all that research must have paid off, with plenty of references to to the work of people like Capers Jones and Barry Boehm. The last part of the book is a catalogue of best practices, 27 in all. Some of these just refer back to the earlier chapters. For each best practice, an indication is given as to its likely effect on schedule reduction, visibility, risk, chance of 1st time success chance of long term success.

In conclusion, a professionally written and well researched book, but perhaps a little dated with respect to Agile methods.

Software Requirement Patterns



By Stephen Withall, published by Microsoft Press ISBN: 978-0-7356-2398-9

Reviewed by Paul Floyd

I've read a trio of books on requirements recently (the other two by Karl E. Wiegers). Book reviews are subjective things, and with me being a mere code monkey, and perhaps I aimed a bit above my station in reading this book.

The premise of the book is to help people write requirements patterns with the aim of making writing the requirements themselves easier. As with all such attempts at raising the level of abstraction in order to increase the amount of reuse, the problem domain needs to be sufficiently large that the extra work in the abstraction pays off in the reuse. The example patterns in the book are oriented towards corporate IS database centred projects. If you're developing embedded software or COTS then the examples won't be directly applicable to you, which means that you'd have to write your own patterns from scratch rather than 'leveraging' the examples in this book.

The first two chapters cover the basics of requirements. There are extended versions of these chapters available for download, grrr, as a Windows setup.exe file. Even on our Citrix Windows machines, I don't have the rights to run an installation program. What's wrong with pdf?

The next two chapters cover the concepts of requirement patterns and how to use and create them. The patterns are made up of 9 sections: Basic details, Applicability, Discussion, Content, Template(s), Example(s), Extra requirements, Considerations for development and Considerations for testing.

The bulk of the book, about 280 pages, consists of a catalog of requirement patterns. Overall this was clear enough, though a few times I felt that Withall was slipping a bit into tutorial mode and explaining some of the basics of databases and how to tackle requirements like support for disabled users.

The Principles of Successful Freelancing

By Miles Burke, published by Sitepoint, ISBN: 978-0-9804552-4-3

Reviewed by Robert Jones

I received this slim volume (185 pages) way back in January 2009, and only now have I managed to get around to writing up the review, which tells its own story really! Needless to say I have not found this a gripping read. Despite both a natural desire to see it through, and a certain moral pressure since I was provided with the book for free, I have not actually read it from cover to cover, in over 18 months.

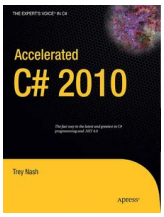
The book begins with a superficial, and faintly obvious, coverage of the qualities required of a freelancer, which frankly seem to be qualities required of any slightly successful professional, briefly tilts at accountancy and bookkeeping obligations (which bearing in mind the author is Australian somewhat lacks a relevant focus), before hitting its main stride with a series of case studies.

Perhaps if you found your exact clone among the case studies there would be some value in them for you, but I found them contrived and lacking in any real insight.

The latter part of the book, which due to induced hypersomnia, I've only skimmed covers longer term issues of freelancing such as general wellbeing, intellectual property, work/life balance, etc.

All in all, not a book with pride of place on my book shelf.

Accelerated C# 2010



By Trey Nash, published by Apress, ISBN : 978-1430225379

Reviewed by Allan Kelly

Browsing the ACCU list of books for review it occurred to me that never having coded in C#, and having been away from the code-face for a couple of years, I was well positioned to review what, from the title, appeared to be a rapid introduction to a language.

The first thing I noticed about the book is that it is big, 600+ pages, and the font is small. Actually it is more than the font, the typesetting leaves a lot to be desired. The lines seem very close together and the side boxes aren't boxed – they just kind of appear (in a larger font) before you realise you are reading one. Even the notes which are boxed sometimes appear out of position. This is certainly not a book to read on a train or a plane.

Reading the first chapter I was a little confused about who the book is aimed at. It seemed to be saying it's for the beginner in C# and then says 'you probably know about .NET already', an odd mix. Elsewhere the author says 'I suggest you read this with the C# reference close by because I can't explain everything'. In which case what does it explain?

The constant references to C++ led me to decide it is aimed at ACCU members, or perhaps people who would benefit from joining the ACCU. People who know how to program, know C++, and now need to learn C#. This might be why the book occasionally presents IL assembler to illustrate a point.

The ordering of the material seems at times strange. There is a big discussion of types before classes are introduced; control structures (all of half a page) come after generics are mentioned; sometimes sections seem to change direction half way through.

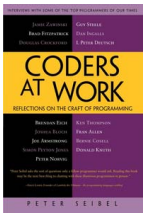
Somewhere the author states this is the third edition of the book – usually publishers make a big thing of this because it's a sign of a good book. But I get the feeling the text has been added to without regard for the flow of the book and cohesiveness of the sections.

As a result the book comes across as a stream of consciousness – types, ref types, value types; O generics are new and cool, now about .... Which is a shame because the author knows his stuff and he has a relaxed chatty style which would be quite readable were it not for the typesetting.

Overall this is not a book for beginners. It doesn't introduce programming, or C# syntax. Nor is it a book for advanced readers; some material is too basic for that. It's for intermediates. This is a difficult group to aim at because intermediates each know more about some things than others.

In the end I gave up on the book after a few chapters because it hurt my eyes, literally. There is a good book in here, but it needs to decide what it is trying to be. Then someone needs to restructure the material and get the thing properly styled. As it stands I suspect that it would be a useful reference book – even if it wasn't written as such.

Coders at Work – Reflections on the Craft of Programming



By Peter Seibel, published by Apress, ISBN : 978-1430219484

Reviewed by : David Pol Ahonen

Verdict : Highly Recommened

*Coders at Wor*k by Peter Seibel captures eighty hours of interviews with fifteen universally renowned programmers. You may not know all the names in the book by the time you start reading it, but very possibly you know or have used some of the software they have created.

The structure of the book is simple: each interview is preceded by a short introduction to the programmer that sets the context for the reader and then comes the interview itself. All the interviews share some common questions ('How did you learn to program?', 'Do you ever write down anything before you start writing code?', 'How do you read a big program that you didn't write?', 'When you're debugging, what tools do you use?', 'Do you still find the kind of programming you do interesting?') that Seibel uses in a smart way to later introduce specific questions that are more tailored to the interviewee at hand (Peter Norvig and Google, Simon Peyton Jones and GHC/ Haskell, Joe Armstrong and Erlang, etc.). I think this approach makes the book a joy to read: you learn historical anecdotes of the field (as most of these programmers started in their craft in the 50s or the 60s), you get to know their programming practices and you learn about what they are doing now, how they are doing it and what they think about the future.

It definitely helps that almost all of the interviewees answer Seibel's questions with such lucidity and sincerity. A lot of what is written here can make for interesting quotes. I was certainly surprised to learn that many of these computer science heroes prefer print statements over a debugger, and relieved to confirm that they are all big pragmatics and therefore tend to avoid big designs and prefer to 'just do it' and iterate as needed. Of course, not everyone agrees on the same things, and that is also part of the richness of this book: it will expose you to points of view that may conflict with yours and may even trigger a change in your own thinking. That is what this is all about in the end: what makes a great programmer working in a team in the real world? This book is a good first step to answering this question.

In summary, *Coders at Work* is a very entertaining read and, if you find the content as interesting as I did, you will probably finish it in a sitting or two. Anyone with an interest in programming will have a great time with it.

Perl Fundamentals (booklet and DVD)

By Peter Scott, published by Addison Wesley / livelessons, ISBN : 978-0137001279

Reviewed by : Ian Bruntlett

This package consists of a 51 page booklet and a data DVD of 4 hours of instruction. It is a data DVD with audio-video files (`.flv` files, I had to install VLC media player to watch the tutorials on a Ubuntu 9.10 system). It has a folder with the examples source code in them. I found that to be of particular help when typing in the examples by hand and trying to run them. I found the Perl error messages to be as helpful as a C++ compiler's STL error messages. Because it is a data DVD, the files can be copied onto a folder and run directly from hard disk. This was particularly convenient. This course requires commitment – I spent about 20+ hours studying this course and making notes (24 sides of A4 paper). The course is split into 8 lessons and an appendix. I will structure this review accordingly. I'll overlook the similarities that Perl has with C/C++, concentrating on some of the surprises in store for newbie Perl programmers.

Lesson 1 [23m:37s] Introduction

Basics. Includes an overview of articles that can be accessed by the perldoc command.

Lesson 2 [53m:18s] Basics

Simple variables (scalars) for string and arithmetic operations. In particular, interpolation is a powerful way of handling strings. Conditional statements e.g **`if else elsif`** and the less commonly found **`unless`**.

Lesson 3 [32m.11s] Arrays, Lists, Looping Statements and Command Line arguments

The heading pretty much says it all. Perl arrays are more flexible than C's arrays – arrays can be printed in **`print`** statements by

the use of string interpolation – something akin to `printf` on steroids :)

```
my @Beatles = qw (
  John Paul George Ringo);
print "The Beatles are
  @Beatles\n";
```

Lesson 4 [30:30] Hashes (associative arrays) and logical shortcuts

Hashes need no explanation. In this chapter there are some interesting examples of how to use hashes. There are code fragments like:

```
my $date = shift or die
  "Usage: $0 date\n";
```

which use default parameter for the `shift` function and a strange looking conditional expression. Most programmers would normally code the above with something like:

```
if ( ! my $date )
  die "Usage: $0 date\n";
```

For me, the things that confused me in the past were fragments of code which were so alien to C programmers.

Lesson 5 [30m:52s] Subroutines for reusable code

Perl functions (subroutines) don't have prototypes. They look something like this:

```
sub print_cost
{
  my ($pet, $cost) = @_;
  # put parameter values into
  # pet and cost
  # rest of function goes here.
}
```

Lesson 6 [36m:21s] File handling. The implicit variable $_

This chapter deals with hard core Perl magic. The diamond operator <> is a Perl iterator. For instance, this is a way of finding out the longest line in a group of text files:

```
my $max_length = 0;
while ( my $line = <> )
{
  chomp $line;
  $max_length = length($line) if
   length($line) > $max_length;
}
```

The <> defaults to reading in all the text in all of the command line parameters. This isn't the shortest way to count file lengths – that honour lies with the use of the implicit variable (spelled `$_` but more commonly left out.

Lesson 7 [37m:16s] Regular expressions (regexes) – searching and changing text

This is very much a strong point of Perl. This lesson goes into the details of using regexes and has a 6 step algorithm that explains how a regular expression thinks – simplified because it doesn't cover backtracking. There are different standards for regular expressions so if you're hopping from one language to another, you may have to take care that you're speaking the correct regex dialect.

Lesson 8 [11m:38s] Perl modules (libraries), CGI and web programs

Perl modules are stored in an online repository called CPAN. It's Perl's equivalent of C++'s boost libraries except that CPAN is more of a repository than a peer-reviewed library.

This is the worst part of the tutorial. It says 'download Perl modules' but doesn't list whereabouts in the perlmod reference material there is a FAQ about downloading CPAN modules.

There is an example in this lesson, `mech.pl`, which uses the `WWW::Mechanize` module. This wasn't present on my PC. I tried to install it using the CPAN command with root privileges but that didn't help. In the end I installed the module using Ubuntu's Synaptic package manager and got it to work.

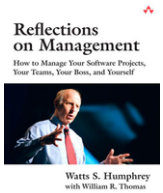
There is also an example CGI script (for web server side processing) but I didn't check it out because I'm still a newbie when it comes to networking and web servers (in my case, Apache).

Final notes:

When studying this tutorial, I found my local Linux User Group's mailing list a good way to get help. While watching the video clips, I had to use VLC media player on my PC (running Ubuntu 9.10).

Verdict: If you want to learn Perl and are willing to put the hours in, you should buy this tutorial.

Reflections on Management



By Watts Humphrey, published by Addison Wesley, ISBN: 978-0-321-71153-3

Reviewed by Paul Floyd

Recommended.

To give the book its full subtitle *How to Manage Your Software Projects, Your Teams, Your Boss, and Yourself*. Quite an ambitious range.

For those that don't know about Watts Humphrey, he's the man behind much of CMMI. He earned his stripes at IBM, managing the development of OS/360.

The book has something of a *Readers' Digest* of the other works of Humphrey (on PSP – Personal Software Process and TSP – Team Software Process). That's not a bad thing, as it makes the book refreshingly short and accessible. The tone throughout is brusque, positive and upbeat. I don't think that Humphrey has much time for people that have doubts and hesitate. Issues are tackled head on, usually with a step-by-step plan.

Broadly, the themes covered in the book are software quality, making yourself and your team efficient, and managing yourself and your boss.

Personally, I'm not entirely convinced by the software-process-as-industrial-process thesis, but clearly it does have merits. The book contains plenty of plain common sense, like 'define your objectives' and 'always start with a plan'. What I found most interesting (and true to life) in this book were the human insights. For instance, what managers at various levels expect and aim for: a Vice President with strategic long-term goals compared to a line manager with tactical short-term goals.

Final word, I found the book interesting enough for me to get a copy of *PSP: A Self-Improvement Process for Software Engineers* by the same author.

View From The Chair

Hubert Matthews
chair@accu.org

I've been thinking recently about the ACCU, where it's come from and where it might head in the future. The ACCU has changed over its 24 years and I think it is likely to change considerably more in the years to come. It started as a geographically focused technology interest group (and it still retains the official name of the C Users Group (UK)). It then took on board another core technology (C++) and spread outside the UK. In recent years it has started to shed its technology-specific origins and has moved towards being an organisation for software developers wanting to learn and improve. I think it is clear to most of the members that software development in general has changed and their jobs have changed too. Few people now sit in functional silos and we have all had to become multi-skilled and more aware of the overall needs of our projects and companies. Few have the luxury of single-language development. Just as in leading-edge science, software is now multi-disciplinary with cross-functional skills and teams.

What does the ACCU's 24-year history tell us? What can we learn and what can or should change? I think the key points are these:

- Technology changes over time but our general goals and aims change less
- We have moved from a technology focus to a developer focus
- Members know why they are members but find it hard to explain to others
- We have been primarily inwardly focused

Some years ago the name 'ACCU' officially stopped being an abbreviation but old habits die hard and people still ask what it stands for. Perhaps it is time to consider whether the ACCU should rebrand itself and consider changing its name. When I discuss this with members they almost always think it's a good idea – they know all too well the problems they have talking to potential members ('well, ACCU used to mean...') – but they are understandably concerned about losing the brand we have built up and they also have no idea what a better name might be. Names are extremely important (think of the power of variable and function names in code) and we need to consider the overall effects such a refactoring would have. We certainly wouldn't want to suffer the fate of Consignia, Monday, Inprise or any of a long list of other failed and forced business renamings. In our own industry the BCS has recently attempted to change its brand to CIIT but people still think of it as the BCS. Brands, names and clear messages are important and the ACCU needs to sharpen up these aspects if it is to grow and thrive in the new broader world of software development. There is a need for the long-term community and opportunities for growth that we offer; we just need to work out how to express that clearly and succinctly to others so that they too can recognise the value. We also need to start thinking about being less internally focused and more aware of how people find out about us, what they think we stand for and why they might want to join.

In summary, I think the ACCU is approaching a crossroads in its history and we have to start thinking about where we want to go, how to get there and who we want to accompany us on the journey. I also think that the membership realises this even if they've not voiced it explicitly and they are ready and eager to tackle these issues. Let the debate commence!

ACCU website attack

Tony Barrett-Powell

Some of the regular visitors to the ACCU web site may have noticed a recent rendering problem and short outage of the site. This was a consequence of an attack on the web site and the subsequent restore of the site to a good state.

We know the following things:

- An attacker logged into the website using an Administrative user's credentials
- A script was added to the home page of the site in an attempt to gather sensitive configuration files at the server level
- The administrative functions were used to browse data held in the website database

We can only speculate how the attacker obtained the administrative credentials. A compromised PC containing the stored password in a browser is an obvious candidate, but we have been unable to identify any such system.

The attacker's attempts to add the script were technically poor and caused the home page of the site to fail to render. Thus we are certain no passwords were obtained using this script.

More concerning is that we cannot rule out the possibility that the attacker may have downloaded the entire database backing the site, which does contain information about the members of the site, including names, email addresses, postal addresses and the passwords for the site. We have never stored any credit card or information about any other payment form on the site.

The site was designed with the possibility of a security attack in mind, so we had already taken the precaution of salting the passwords stored in the database, so there is no chance of decryption in this case.

However, if the database was downloaded the attacker will have access to member's names, email and postal addresses.

We took some immediate actions:

- Disable Administrative access
- Ensure all passwords of any Administrative users were changed
- Establish a finer grained set of privileges for those users contributing content to the site who previously had administrative rights

It is also our duty to inform the members about the attack and our assessment of the extent of the breach, which is the intent of this report.

If you have any questions please contact me on webeditor@accu.org.