Contents

Reports & Opinions

Dialogue

Features

Reviews

Copy Dates

C Vu 17.3: May 7th 2005

C Vu 17.4: July 7th 2005

Contact Information:

Editorial: Paul Johnson
77 Station Road, Haydock,
St Helens,
Merseyside, WA11 0JL
cvu@accu.org

Advertising: Thaddaeus Frogley
ads@accu.org

Treasurer: Stewart Brodie
29 Campkin Road,
Cambridge, CB4 2NL
treasurer@accu.org

ACCU Chair: Ewan Milne
0117 942 7746
chair@accu.org

Secretary: Alan Bellingham
01763 248259
secretary@accu.org

Membership Secretary: David Hodge
01424 219 807
membership@accu.org

Cover Art: **Alan Lenton**
Repro: **Parchment (Oxford) Ltd**
Print: **Parchment (Oxford) Ltd**
Distribution: **Able Types (Oxford) Ltd**

Membership fees and how to join:

Basic (C Vu only): £25

Full (C Vu and Overload): £35

Corporate: £120

Students: half normal rate

ISDF fee (optional) to support Standards work: £21

There are 6 issues of each journal produced every year.

Join on the web at www.accu.org with a debit/credit card, T/Polo shirts available.

Want to use cheque and post - email membership@accu.org for an application form.

Any questions - just email membership@accu.org

Reports & Opinions

Editorial

Pandora's Box

It seems that I have opened the proverbial Pandora's Box with my last editorial. The accu-general mailing list went almost haywire with the number of comments regarding the issue raised (for new members, see my article later in the magazine). I knew that demographically we had a large-ish chunk of the membership who would be of the age to appreciate the comments I made, but not on the scale they did!

The other change I feel I need to justify is the size of the Student Code Competition and the knock-on effect that it had on the space for the other articles, book reviews and other regulars.

Since taking over the editorship of C Vu, I've noticed that the number of people contributing to the SCC (and sending in letters/emails directly to me) has been somewhat sparse. On average, the SCC gets 2 entries, which makes the judging very unfair (in my opinion). In the last edition, we had so many entries that I felt it worthwhile publishing all of the entries and having two prizes instead of the one. The basic premise is that if you submit an entry, unless it really is awful it deserves to be in print. There is also a feel-good factor involved (it used to be called something like the "look Mum, my name's in print" factor when we had a myriad of computer magazines in the 80s); I'm not suggesting that as professionals we do such a thing, but I can remember both my first book review in C Vu and when I showed the first editorial to colleagues the buzz of pride in the hard work presented.

I want people to write in and contribute to the magazine. Has the gamble paid off? Only time will tell.

OpenOffice2, Mono and gcc 4

I hate standing still and I can't stand software which stands still either. I don't use a stable version of Linux (I use Fedora Core rawhide – "it eats babies and if it destroys trees, your computer, your mother's computer and a number of urchins, well, it's your fault" – as it was once described on a mailing list) and despite of that, riding on the bleeding edge is what makes computing fun for me. It can be a pain in the bum though at times!

I realise that beta software is not for everyone (I certainly don't have rawhide working on my machine at work or on my wife's and son's machines).

Recently, OpenOffice2 was moved into rawhide. This is the beta version of what OOo2 will become and already it has become stable enough for me to stop using OOo 1.1.3 (though it still sits on my machine "just in case"). It really is a wonderful product. While you cannot yet place an equation into a table and there are a few problems with importing old OpenOffice files with vector graphics in, it doesn't detract from the sheer quality of the product.

Having had a chance to play with MS Office 2003 and OOo2 and leaving my prejudices at the door, I can honestly say that now OpenOffice2 has a database system in place, the case for using Office 2003 on the basis of Access doesn't hold water anymore. The database form editor in OpenOffice2 is every bit as simple to use as Access and has the advantage of connecting to a raft of different database server types, so you can now just connect it to an existing MySQL server or ODBC or Oracle system.

Compatibility with PowerPoint and the other packages in MS Office as well as the user interface is so similar that when I recommended it to someone who had only ever used MS Office for home use (but was on a tight budget), he actually preferred it as it no longer felt like he was the slave to the package!

That's not to say it is all there yet. It isn't. The software last week (first week in March) only reached a beta 1 release, so there are plenty of issues left to be ironed out.

For those who have read my editorials, you'll no doubt know that I have been using Mono (one of the open source implementations of the .NET framework) quite extensively and helping out where I can. The Novell team have been busy and have released an all-in-one installer for many platforms. If you don't want the hassle of finding the bits and pieces, please feel free to try it out and see how it compares to the MS .NET framework. I did have an article lined up this edition from one of the hackers working on the `System.Windows.Forms` class, but due to his work pressures, it will be delayed.

Another addition to rawhide has been the latest flavour of gcc – gcc4. As with each update of any packages, it has quite a number of fixes, speed improvements, platform optimisations and improvements to bring it closer to the relevant standards for each language. While not new, it is good to see that gfortran is still in there and that a java compiler has been added in. I have a long term love of FORTRAN having spent more nights than enough using it for my M.Phil. Call me Mr Nostalgia if you like, but having the requirement for 6 spaces before code, dire text handing and not to mention a very linear approach to code in 1999 made me appreciate the heritage of the language (punch tapes and the likes). It is nice to see the latest version being F95 compliant.

The Conference

By the time this edition hits the door mats, the annual conference will be upon us and it should be one of the best (how do we keep improving them year on year? Simple answer is the reputation for being one of the finest in Europe as well as the sheer quantity and quality of speakers we manage to recruit). Unfortunately, due to illness, I won't be able to make it this year and so I hope you all have/had an enjoyable conference.

Getting Involved With C Vu

Despite how it looks, both C Vu and Overload usually both run short on articles and can always do with new contributors. If you have an idea for an article (either one-off, or as a series), then please get in touch.

In past issues, we've had a wide range of articles (including a lot of cross-platform material, specific libraries, Objective C and patterns), but these only last so long and being good editors, we don't rest in searching out for new writers.

You still retain copyright on all material, so you have the benefit of writing for a magazine that is read worldwide and also keep hold of the original, which means you can then publish it for yourself (should you wish to of course!).

Fancy it? If you do, please email `editor@accu.org` for C Vu and `overload@accu.org` for Overload and we'll be happy to fill you in.

And On With the Show...

Enough of me for now.

I must express a large thank you to both Pippa and Alan for helping me on this issue in its closing stages due to having to go into hospital for a rather unpleasant operation.

Paul F Johnson

View from the Chair

Ewan Milne <chair@accu.org>

I believe that by the time you are reading this, there will be just enough time to make a last-minute booking for the conference. What have you been waiting for? Stroustrup, Coplien, in fact the unrivalled quality of the whole programme: you know you'll only regret missing out. Of course, you may be sensibly booked up already, in which case can I just ask you to remind yourself of the information in the last C Vu and that you've been emailed about the AGM. It is an important part of the ACCU calendar.

Indeed, with the conference and AGM imminent, we are at the end of another year's activity for the ACCU. I would like to thank everyone who has played a part in the life of the association over the past year. Members of the committee, the C Vu and Overload editors, editorial teams and authors, conference speakers and delegates, `accu-general` posters and everyone who helps with the surprising number of administrative tasks that keep things running smoothly: your efforts are very much appreciated.

I intend to stand once again for the post of ACCU Chair for the coming year. However I have decided that this will be my last year in the job, and will stand down at the 2006 AGM. I find that covering the role of Conference Chair takes up a good proportion of my "ACCU time", in fact as much time as I will be able to spare from now on. Why the full year's notice? Well, there are some ongoing issues I'd like to see through, not least the revamp of the website and its implications. Also it is traditional for the outgoing Chair to seek a replacement before vacating the post. In fact I now believe I can depend on a candidate coming forward from the committee.

I can easily say that I have greatly enjoyed my involvement in helping to run the association, as I'm sure I will continue to in the future. I certainly recommend you seriously consider getting more involved: you stand to gain a great deal of enjoyment and satisfaction, and the ACCU always needs enthusiastic people to move things forward. Please talk to me or a fellow committee member at the conference if you have any ideas, skills or time to offer. See you then.

Secretary's Report

Alan Bellingham <secretary@accu.org>

The fourth and final committee meeting of the 2004/2005 year took place on the 19th of February 2005 in Bristol, at the house of Tony Barrett-Powell. This was our first excursion to the west of England, but since both our Chair and our host live there, it was nice to let them not have to travel as far as usual.

We normally start (or try to) at 13:30, which allows us to meet beforehand at a pub or similar and have a meal. This allows a certain amount of informal chat about what we're going to discuss, as well as letting us get up to date on whatever else we've been doing. This time, we were joined by the famous Kevlin Henney, who actually lives on the same road as Tony, though apparently not for much longer, as both of them have their houses up for sale.

At the actual meeting, we carried out the normal administrivia, officers' reports, looking over the previous minutes to check what had been done, and so on. The officers' reports should be pretty much as you read them elsewhere in this issue (allowing for two weeks of extra time since then).

The major issues facing us at the moment are the appointment of a new production editor to replace Pippa Hennessy (which is in hand), and the new web site. Allan Kelly's team had received five tenders, and he made a comprehensive report to the meeting on what each candidate wanted to charge, how they intended to do the work, what Allan and Co. felt about the tenderer, and ultimately, which tender we should go for. There were a number of questions from other members of the committee, but Allan covered them comprehensively and it's a tribute to his preparation that we agreed unanimously with his recommendation and, before the end of the meeting, signed a contract with that choice.

The other item that will be even closer when you read this is the AGM. Last issue, I supplied a draft agenda. However, there may be a couple of constitutional agenda items added by the meeting.

The first is that we are constitutionally limited to three types of membership – effectively a basic, a standard and a (single site) company type. There is a desire to allow more types.

The second is on the election of officers. At present, if a written nomination for a post has been lodged in advance, then no nomination may be accepted from the floor at the AGM itself. Although we have a tradition of therefore not making written nominations, it seems odd to have to deliberately dance round our own constitution's rules in this way.

The actual wording of these items is not yet fixed, and it is possible that neither will appear.

The next meeting is currently scheduled for the 5th of May, probably in Royston, Herts. Exact details do depend on the results of the Officers' election at the AGM. Members are not excluded, so if any of you do want to come along and see what we do on your behalf, just get in contact and let me (or my successor) know so I can send you the details in advance.

Standards Report

Lois Goldthwaite <standards@accu.org>

The UK C++ panel brings a special perspective to meetings of WG21, the international C++ standard committee. Most of the other international participants are vendors of compilers or libraries, or else people at the bleeding edge of C++ development.

In the UK, on the other hand, panel members for the most part are working programmers who use the language in their daily tasks – not vendors of C++ but consumers, if you will.

We see our mission as helping to hold a focus on how C++ and the standard committee can address the needs of ordinary working programmers, roughly defined as people who wouldn't even think about using a template template parameter oftener than, say, once in their life, and then only if absolutely necessary.

In recent discussions the panel has identified a deficiency in the C++ world, the lack of a library of easily-reusable components for what Francis Glassborow calls incidental programmers – people who write programs (frequently one-offs) in support of their real work such as scientific research. What is needed is a CPAN for C++. The Boost Library fulfills some of this purpose, but is pitched at highly competent professional programmers.

It's early days yet, and whether we will move past talk into action remains to be seen. Meanwhile, if you have an opinion on how C++ could develop better to meet the needs of the programming community, you are invited to join the panel and make your voice heard. Even if you don't want to join, your comments on standards matters are always welcome. Please write to `standards@accu.org` for more information.

Membership Report

David Hodge <membership@accu.org>

Since the last issue the membership has increased by about 50, and stands at 982.

This increase is mainly due to the conference, with people taking advantage of the discount for ACCU members and the reduced early bird rates that were available until the end of February. I expect that by the time this hits the streets in April we will be over 1000 members.

The membership secretary keeps the only database of members' details, so it is me you need to email if your contact details change. This database is used to create the labels that go on the journal mailings, so if this went to the wrong address just send me an update.

Officer Without Portfolio

Allan Kelly <allan@allankelly.net>

At the end of January the New Web committee received five tenders from organizations interested in redeveloping our website. Two came from ACCU affiliates and three from other organizations.

The committee reduced this to a shortlist of three – one of the bidders was eliminated on the ground that one part of their system required the use of Microsoft Internet Explorer exclusively. We made additional enquiries of the remaining three.

The company that submitted the lowest bid was always going to be the one to beat and in this case it was also one of the strongest bids. Together with Alan Lenton I visited the company's offices and discussed the project in more detail.

As a result the New Web committee recommended to the full committee that the ACCU accept the bid from Turtle Networks (`http://www.turtle.net/`). The committee agreed with this at the February meeting and by the time you read this contracts should have been exchanged. With luck the first elements of the new website will be in place for mid April – just in time for the conference.

Turtle is a small company based in West London. They have been established for several years and have a number of large websites to their credit. They are a major supporter of the London Linux User Group (`www.lonix.org.uk`) and use Linux for their systems. So those of you who feared a Microsoft-centric site can sleep better at night.

This is just stage one of the website project, the committee envisage 7 to 8 stages before we have a completely new site, so we won't be relaxing any time soon.

Dialogue

Francis' Scribbles

Francis Glassborow <francis@robinton.demon.co.uk>

Books

A couple of readers have responded to the item in my Bookcase editorial:

> Now to turn to something positive, and something you can all join in. I would like readers to do three things. First select the book that you have read that you think has been most underrated or overlooked. Just one, and I know that makes it hard for some but the effort of choosing can focus the mind. Of course there are no right answers but it will be interesting if some books turn up more than once (and if only three readers respond …)
>
> The second thing is to choose a category (novice programmer, newcomer to C++, embedded systems developer, games developer, etc.) and list which books you would recommend given a) a budget of £100 ($180) and b) a budget of £250 ($450).
>
> And lastly, given a budget of £2000 ($3600) list what software development tools and references you would take with you for a year's stay on a desert island. The desert island comes equipped with the essentials for life and electric power.

Before I hand you over to them, I would like to list a few books that I think are either underrated or frequently overlooked. Some of these are out of print though if people understood their quality they would not be.

Ruminations on C++ by Andy Koenig and Barbara Moo. Its present ranking on `amazon.com` of 350, 000+ is a complete disgrace. This is one of the most readable books on C++ and should be essential reading for every new C++ programmer.

Programming on Purpose (I, II and III) by P. J. Plauger

Programming Pearls & *More Programming Pearls* by Jon Bentley

Those five books should be read by every aspiring programmer. That may prove difficult as two of Plauger's books are out of print and the third only available in Facsimile as is *More Programming Pearls*. *Programming Pearls* is the only one with an `amazon.com` rank below 400,000, and even that has a miserly ranking in the 43000s.

Multi-Paradigm Design for C++ by James Coplien with an `amazon.com` sales rank of 488, 000+ is another book that deserves a much wider readership, this time among the more experienced C++ users.

Confessions of a Used Program Salesman by Will Tracz. The `amazon.com` sales rating of 949, 000+ shows how poorly known this book is. Readers almost invariably sing its praises, but still it languishes gathering dust. I suspect this book would have been my choice for the most overlooked book.

Obfuscated C and Other Mysteries by Don Libes (`amazon.com` sales rank: 869,000+). This book teaches a great deal about good C programming by exposing the reader to programs from the annual Obfuscated C competition. It is a rich source of study material for the aspiring C programmer.

C Traps and Pitfalls by Andy Koenig is only in the 160,000 on amazon.com. That is pretty poor for a book that should be read by every aspiring C programmer.

Expert C Programming by Peter van der Linden. This book is not quite so badly rated as most of the above, but with a sales rating of 101, 000+ on `amazon.com` it deserves much better. Even C++ programmers could learn quite a bit by reading it.

There are quite a few others but I offer you the above as a starter. What other books on programming and software development should be in this list? Let me give a guideline for your search. What book would you recommend to at least one category of software developer whose current sales rank on `amazon.com` is worse than 250, 000? What can we do to improve things so that these books become better known? Please do not sit back and let others do the work. Just as open software development draws in the skills of many people, we can do the same thing for other types of project.

I would like to see ACCU develop high quality reading lists for different target audiences, but it would be most unreasonable to expect one person to any one of them. For example, what should be the reading list for someone who aspires to be a games programmer? A process of progressive refinement should be applied to such a list. In other words, we need dialogue, we need you (yes I mean you, the person reading this now) to get involved. We need preliminary lists and we need others to review those lists (the lists, not the books). If we do the job right we can seriously influence the development and training of software developers. Ideally our lists should become the basis for University reading lists for the areas we target.

Now to those two reader submissions.

From Jez Higgins

Been thinking about your wretchedly difficult competition for the last couple of weeks now. Am now at the point where I have to mail you or it will haunt me forever.

Underrated or overlooked book: *Software Tools in Pascal* by Kernighan and Plauger. Now obviously this is not some amazing undiscovered gem, because you frequently see it mentioned and it is still in print after nigh on 25 years. We do work, however, in an industry where a book a year old can be past its sell-by date, and that makes it easy to discount a book that was published when many of my work colleagues were unwrapping their ZX81s. Often, when I recommend it, people say "in Pascal?" and laugh in a nervous are-you-going-mad way.

Once they actually read it, they are generally converted. There is a clarity in the writing, coupled with a strong narrative. Starting with an almost laughably simple program to copy the input to the output, it builds to a rather capable line editor, text processing and macro processing almost without you noticing. All the time it stresses good practice – refactoring, library building and so on. That the code is in an obsolete language is almost an advantage in these modern times. Everyone can read Pascal, even if they've never encountered it before, and translating to your language of choice is a useful exercise of itself. It is just super.

Books for <someone>: My category is a C++ programmer who did not actually train as a programmer, sort of fell into it by accident, and who works alone or in a small, inexperienced team of people. They are generally keen, but lack direction so have to rediscover everything for themselves. The first selection to £100 leans toward good practice.

Code Complete by Steve McConnell, £23.79 – changed how I work

The Practice of Programming by Kernighan and Pike, £17.23 – changed how I work more

The C++ Standard Library by Nico Josuttis, £29.91

The Pragmatic Programmer by Hunt and Thomas, £23.19

On up to £250, I would add in some more C++ specific advice, together with a couple of books to remind them that good practice goes wider than just the code in front of you.

The C++ Standard, £24.47

Effective C++ by Meyers, £25.49

Effective STL by Meyers, £21.69

Managing Projects with GNU Make by Mecklenburg and Oram, £15.71 – if there was one program I could have "invented" it's `make`

CVS Pocket Reference by Purdy, £6.95

Software Tools in Pascal by Kernighan and Plauger, £19.99 at Amazon

Writing Solid Code by Steve Maguire, £9.95 used at Amazon – the homely anecdotal pair to Code Complete's rigour

The Mythical Man Month by Brooks, £22.99

Castaway tools and references: This is extremely difficult. It rather depends on what I would spend my time working on if I did not have to work. Since I am finishing up an XPath engine, I suppose I might have a crack at an XSLT 2.0/XPath 2.0/XQuery implementation. In addition, maybe try a bit of games programming. I'm not sure I can actually get close to the £2000 limit. Most of the tools I use are available at no cost – emacs, CVS, grep, gcc and so on, although I might also go for a couple of Visual C++. The standard version would probably be fine.

I'd take the books listed above, plus

C Standard

C++ Templates by Vandevoorde and Josuttis

Standard C++ IOStreams by Langer and Kreft,

Java in a Nutshell by Flanagan,

XML in a Nutshell by Harold and Means,

XSLT by Tidwell,

Python Cookbook by Martelli and Ascher,

Programming Pearls by Bentley,

The XML, DOM, XPath 2.0, XSLT 2.0 and XQuery recommendations

Friedl's *Mastering Regular Expressions*,

Philip and Alex's Guide to Web Publishing by Greenspun – just terrific

XML Topic Maps: Creating and Using Topic Maps for the Web– my friend Kal is a co-author on this, and I keep meaning to find out what it really **is** he does,

Modern C++ Design by Alexandrescu

A ZX Spectrum emulator and David Webb's Super Charge Your Spectrum and Advanced Spectrum Machine Language. The latter book was the first one to really show me that you could have a library of routines to draw on again and again.

I am showing a singular lack of imagination. I really cannot think of anything else ...

All prices are from Computer Manuals, unless otherwise noted.

Even as I type this, I'm doubting myself. Should I have included Fowler's Refactoring? Booch? Peopleware? Gah. As I said at the beginning, it has been extremely difficult.

When I commented that he did not seem to have any hardware to use, Jez added the following:

For a box to work on I would take something like a Demonite Graduate. It does not have to be top of the range, because it is not like there would be huge pressure of time or enormous builds to do. The basic ~£825 version would be fine (my own slightly more expensive one does go like the clappers - they are super). I'd swap the bundled flat monitor for an Iiyama Vision Master, say the 454 (~£250). I would also throw away the mouse and get a Logitech TrackMan Marble. I am not sure they still make them, but I paid about £30 for mine. I did toy with taking two machines, but decided that would be a bit silly.

Not bothered about OS, so let me take some version of Linux – Debian or SuSE, say. Most of the tools I use are available at no cost – emacs, CVS, grep, gcc and so on. If I had gone for Windows, I would take a copy of Visual C++. Instead, I will take Comeau's compiler with libcomo ($50), along with Dinkumware's library for Comeau and gcc ($135).

From Thaddaeus Frogley

Most Overlooked/Underrated

Multi Paradigm DESIGN for C++

Note: An incredibly hard one to pick. All my favourite development books are either popular and highly acclaimed or Highly Recommended by ACCU Book Reviewers. In the end, I selected this book because I think many people are put off by the highly academic style, which can make reading it very hard work. Despite this bar to wide popularity, I think this book is well worth the effort.

C++ Programmers Essential Reference Library

(Prices Based on prices listed on `amazon.co.uk`, Dec 2004)

For £100:

£34.99 – *The C++ Programming Language, Special Edition*

£30.79 – *The C++ Standard Library: A Tutorial and Reference*

£29.39 – *C++ Templates: The Complete Guide*

For £250, as above, plus:

£24.46 – *The C Standard*

£24.46 – *The C++ Standard*

£29.39 – *Standard C++ IO Streams and Locales: Advanced Programmers Guide and Reference*

£18.89 – *Exceptional C++*

£20.99 – *Effective C++*

£23.09 – *Modern C++ Design: Applied Generic and Design Patterns*

Note: In creating a list with the theme of "Reference Library", the last three were a tough call.

Desert Island Developers Shopping List

Hardware (source: `apple.com`):

1.8GHz PowerMac G5

1GB DDR400 SDRAM (PC3200) – 2x512

80GB Serial ATA – 7200rpm

NVIDIA GeForce 6800 GT DDL w/256MB GDDR3 SDRAM

Apple Cinema Display (20" flat panel)

8x Super Drive (DVD-R/CD-RW)

Apple Keyboard & Apple Mouse – U.S. English

Mac OS X – U.S. English

Subtotal $3,423.00

Software:

OS X Development Tools (free download)

Misc Home-brew GameBoy Advance Development Software

i.e. Boycott Advance (GBA Emulator for OS X) & GBA SDK (free / Open Source)

References:

As found online (free)

I am assuming either an internet connection from the island, or time to download and burn the software and references wanted to a DVD or two.

My Comments

I found these two interesting responses. One thing I think that is worth noting is that both found the dash limits on books to be very tough to meet. I suspect that a minimal library for a serious software developer would cost of the order of £1000 ($2000).

That tells me two things; being a software developer is both expensive and time consuming. Having the books is not enough; you have to find time to study them.

I think that I am now even more convinced that we need a concerted effort to get rid of the dross from the bookshelves (virtual as well as real) and make a great effort to see that good authors (or authors of good books) get the sales they deserve.

The vast number of bad and mediocre books taking up space is making it very difficult for the newcomer to find good books. Unfortunately, when we start listing the reading requirements the costs begin to daunt. You can easily spend much more on books than on hardware and an order of magnitude more than you need to spend on software.

The Winner

Because there are only two submissions, I am not going to toss for which wins. Instead, I am going to award them both a choice of any book currently in print, costing under £45 published by Addison-Wesley, Wiley, Prentice Hall or O'Reilly.

Commentary on Problem 18

Some programmers seem to hate to use more names in their programs than they absolutely have to. Your challenge is to write a program in C++ that outputs the first n members of the Fibonacci series where the user will provide the value of n at runtime.

That is easy for most readers. However, there is a limitation, you must use `i` as the name of any variable, function, type or namespace that you declare. You may use anything you like from the Standard (such as `main`, `std::cin` and `std::cout`).

What is it about programmers that they tackle writing bad code with such enthusiasm and inventiveness? Because the problem had come to mind because of a novice's attempt at generating Fibonacci numbers recursively, I had completely missed the potential for code such as:

```
#include <stdio.h>
int main() {
  int i[] = {1, 1, 2, 0};
  puts("How many values?");
  scanf("%d", i+3);
  puts("The results are: \n\t1\n\t1");
  while(i[2]<i[3]) {
    i[1] += i[0];
    i[0] = i[1] - i[0];
    printf("\t%d\n", i[1]);
    i[2]++;
  }
  return 0;
}
```

That source code gives the lie to my contention that the problem could not be done in C. The solution I was thinking of is:

```
#include <iostream>
int i(int i) {
  if(i < 0) return 0;
  if(i < 2) return 1;
  return ::i(i-1) + ::i(i-2);
}
int main() {
  std::cout << "How many members?";
  int i;
  std::cin >> i;
  –i;
  do {
    std::cout << ::i(i) << '\n';
  } while(–i > -1);
}
```

[concluded at foot of next page]

Student Code Critique Competition 33

Set and collated by David Caabeiro <scc@accu.org>
Prizes provided by Blackwells Bookshops & Addison-Wesley

Please note that participation in this competition is open to all members. The title reflects the fact that the code used is normally provided by a student as part of their course work.

This item is part of the Dialogue section of C Vu, which is intended to designate it as an item where reader interaction is particularly important. Readers' comments and criticisms of published entries are always welcome.

Before We Start

Remember that you can get the current problem set in the ACCU website (`http://www.accu.org/journals/`). This is aimed at people living overseas who get the magazine much later than members in Europe.

Student Code Critique 32 Entries

I still wonder about the lack of knowledge (or rather awareness) among beginners of the extensive functionality offered by the standard library. Let this be reflected in your answer to the student, with a corresponding solution.

This computes the product of two N by N matrices. It works fine in cygwin compiler, but it doesn't in VC++. The strange thing is when I have N = 2 no problem, but N = 3 makes problem. I am not sure I use 'new' operator correctly in the following program. Can someone help in finding the problem here ?

```
#include <iostream.h>
#include <process.h>
void main(void) {
  int N, i, j, k;
  double **A, **B, **C;
  double sum = 0.0;
  cout << "Dimension of Matrix ?" << endl;
  cin >> N;
  A = new (double *);
  B = new (double *);
  C = new (double *);
  for(i=0; i<N; i++){
    A[i] = new double[N];
    B[i] = new double[N];
    C[i] = new double[N];
  }
  for(i=0; i<N; i++)
    for(j=0; j<N; j++){
      cout << "A[" << i << "][" << j << "] = ?" << endl;
      cin >> A[i][j];
    }
  for(i=0; i<N; i++)
    for(j=0; j<N; j++){
      cout << "B[" << i << "][" << j << "] = ?" << endl;
      cin >> B[i][j];
    }
  for(k=0; k<N; k++)
    for(i=0; i<N; i++){
      sum = 0.0;
      for(j=0; j<N; j++)
        sum += A[i][j]*B[j][k];
      C[i][k] = sum;
    }
  cout << endl << endl << endl;
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      cout << "C[" << i << "][" << j << "] = "
           << C[i][j] << endl;
}
```

From "The Cart Horse"

This is an interesting case; the student knows they have a problem and have even gone to the extent of trying the program on two different compilers. However they don't seem to know what to do with the results of their test! The first thing to do was to try and reproduce the problem.

This was surprisingly hard – I compiled the code and tested it with a matrix size of 3 and it seemed to work perfectly with Microsoft VC 6.0.

Microsoft VC 7.1 failed to compile the program – `<iostream.h>` is obsolete – so I changed it to the ISO standard `<iostream>` and added the appropriate `std::` prefixes to `cin`, `cerr` and `endl`. The code again seemed to work faultlessly.

[continued from previous page]

Obviously, the above solution cannot be implemented in C because C does not allow a distinction between local and global variables. A local variable name in C always hides a global declaration of the same name (except when the global name is declared as a type name by using the `struct`, `union` or `enum` keywords.

My thanks to Martin Stuart, James Talbut and Ian Glover who all came up with excellent C++ solutions. However, the best I think was Tim Sharrock's creative use of `std::vector`:

```
#include <ostream>
#include <istream>
#include <iostream>
#include <vector>
#include <algorithm>
int main() {
  std::vector<int> i(1);
  std::cin >> i[0];
  std::cout << "first " << i[0]
            << " terms of the fibonacii "
               "sequence are:\n";
  while(i[0]— > 0) {
    i.push_back((i.size()<3)
      ? int(i.size()-1)
      : (i.back()+*(i.end()-2)));
  }
  std::copy(i.begin()+1,i.end(),
            std::ostream_iterator<int>(
                              std::cout," "));
  std::cout << "\n";
}
```

Ian Glover also offered this interesting C solution:

```
#include <stdio.h>
#include <math.h>
/* Cheating slightly declaring I, but GCC
won't compile the rest without it and we never
use it! */
int main(int I, char* i[]) {
  while(strtol(i[1], 0, 10)) {
    printf("%d\n",
      (long)((pow((0.5 + sqrt(5.0)/2.0),
                  (double)strtol(i[1], 0, 10))
                   - pow((0.5 - sqrt(5.0)/2.0),
                  (double)strtol(i[1], 0, 10)))
                   /sqrt(5.0)));
/* The nasty hack */
    sprintf(i[1], "%d",
            (strtol(i[1], 0, 10) - 1));
  }
  return 0;
}
```

Cryptic clues for numbers

Here is last issue's clue:

Deuce, it sounds like they came for tea twice. (4 digits)

As many of you worked out, the answer is 4040

Margaret Wood came up with:

Intel's first microprocessor? No, I've heard this one is double strength.

I feel sure I had another one, but I seem to have mislaid it.

Now try this one:

A first course on C++ at the University of Rome [3 digits]

When hunting for alternate clues you might remember that we are considering strings of digits rather than numerical values. And that is about all the extra I am willing to give you.

Francis Glassborow

I next tried mingw and gcc also complained that the program used `iostream.h` not `iostream` (so I used the file fixed for VC 7.1) and it also complained that `main` should return `int`. So I'm not sure what version of gcc the student was using.

My first response in this sort of situation would usually be to try and get a better fault report from the student. I'd want the answer to three questions:

1) What version of the compiler(s) are you using?
2) What command line are you using?
3) What are the actual symptoms of the problem?

However the Student Code Critique is not interactive in this sense so I'll just press on...

What does it mean when a program works when compiled with one compiler but not with another?

In my experience this is usually because the program accesses memory it shouldn't be using. The underlying problem probably occurs with the code generated by both compilers, but there are no (obvious) symptoms with one of them. This is because the actual layout of memory is different in the two compilers (or even with the same compiler when, for example, the optimiser is turned on).

It is very often worth compiling and running a program with multiple compilers – sometimes you get more information from additional compiler warnings and other times the runtime error handling provides additional clues to the problem.

In this case we only have the program failing with one compiler, and even worse I can't reproduce it. So without more ado let's dive into the code itself.

On reading the code it is fairly clear what the problem is; the first allocation of memory for the variables `A`, `B` and `C` does not involve `N`.

`new (double *)` allocates enough memory for a single `double*` – but we actually want an array of `N` pointers.

It is at first sight surprising that the code works at all – it just goes to show that writing only a little bit off the end of allocated memory can sometimes seem to work!

So it would be very easy to suggest the student changes:

```
A = new (double *);
```

to:

```
A = new double*[N];
```

and similarly for `B` and `C`.

But is this helpful? As the saying has it, "give a man a fish and you feed him for a day, teach a man to fish and you feed him for life".

There are some other problems also lurking in this code but attempting to fix them directly leads into deeper waters.

I would recommend that most users of C++ start out by staying well away from `operator new` and letting the standard library allocate the memory for them where possible.

So, looking at the problem with a library user's hat on, what do I want?

I want a class which gives me the characteristics of a matrix.

Sadly the standard library doesn't come with one of these – but it does supply a `vector` class and we can create a matrix by having a `vector` of `vector`s. Alternatively we can search for a `matrix` class that we can download from the Internet. A quick search with "C++ matrix class" reveals several possible candidates.

If we want to stick with the standard library solution then we can define a `row` as a `vector` of `double` and a `matrix` as a `vector` of `row`s:

```
#include <vector>
class row : public std::vector<double> {};
class matrix : public std::vector<row> {};
```

Now the code to initialise the matrices is a matter of resizing the vectors rather than allocating the memory ourselves. This has several benefits over using raw pointers

- it is easier to code correctly
- the vectors will ensure the memory is freed when the function return
- if we are still having memory problems, we could use a debugging version of the standard library to trap out of range indices.

I hate writing anything twice, so I would probably suggest the student writes a simple helper function and calls it when needing to initialise a matrix:

```
void init(matrix & A, int size) {
  A.resize(size);
  for(int idx = 0; idx != size; ++idx)
    A[idx].resize(size);
}
```

This would make the code as presented work immediately and the student would also hopefully have less problems with memory allocation in future.

I might suggest, as an exercise for the student, that they move the matrix multiplication into a separate function so it could be reused and to improve the readability of the code.

I would probably initially suggest a free standing function with a prototype like this:

```
matrix operator*(matrix const & A, matrix const & B);
```

So my initial solution to the student's problem looks like this:

```
#include <iostream>
#include <vector>
class row : public std::vector<double> {};
class matrix : public std::vector<row> {};
void init(matrix & A, int size) {
  A.resize(size);
  for(int idx = 0; idx != size; ++idx)
    A[ idx ].resize( size );
}
matrix operator*(matrix const & A, matrix const & B) {
  int const N(A.size());
  matrix result;
  init(result, N);
  for(int k=0; k<N; k++)
    for(int i=0; i<N; i++) {
      double sum = 0.0;
      for(int j=0; j<N; j++)
        sum += A[i][j]*B[j][k];
      result[i][k] = sum;
    }
  return result;
}
int main(void) {
  int N, i, j;
  matrix A, B;
  std::cout << "Dimension of Matrix ?" << std::endl;
  std::cin >> N;
  init(A, N);
  init(B, N);
  for(i=0; i<N; i++)
    for(j=0; j<N; j++) {
      std::cout << "A[" << i << "][" << j << "] = ?"
                << std::endl;
      std::cin >> A[i][j];
    }
  for(i=0; i<N; i++)
    for(j=0; j<N; j++) {
      std::cout << "B[" << i << "][" << j << "] = ?"
                << std::endl;
      std::cin >> B[i][j];
    }
  matrix C = A * B;
  std::cout << std::endl << std::endl << std::endl;
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      std::cout << "C[" << i << "][" << j << "] = "
                << C[i][j] << std::endl;
  return 0; // keeps MSVC happy
}
```

We would then have two free-standing functions operating on objects of the `matrix` class; we can then move on to suggest ways in which the `row` and `matrix` classes could be enhanced by using member functions and constructors. We might then move on to discuss using composition rather than inheritance.

There are a number of other ways this solution could also be improved, for example to generalise by data type or to improve the input and output, but at this stage I suspect trying to do any more would simply confuse things for the student.

From Calum Grant <calum@visula.org>

The main problem with this code is that it is structured badly. You should always split long functions up into small simple units, which makes programs clearer and more reusable. It also makes it easier to test. Whenever you write code, turn it into generic functions or classes that can be reused. So what you really want is a general-purpose `Matrix` class that you can use in any application that requires a matrix. Rewrite your `main()`

function to look something like this:

```
int main() {
   Matrix a, b;
   int size = input_matrix_size();
   input_matrix(size, a);
   input_matrix(size, b);
   std::cout << a * b;
   return 0;
}
```

This makes the `main()` function much clearer, and all the real work is done elsewhere. We have changed the return type of `main()` function to be standards conforming, and it is conventional to `return 0` from `main()` to indicate no error. Other minor problems with your code are that it is in general preferable to use pre-increment `++j` instead of postincrement `j++` and that you should `#include <iostream>`, not the deprecated `<iostream.h>`. That will require you to put `using namespace std` in your code, or else qualify `cout`, `cin` and `endl` with `std::`. Don't `#include <process.h>`, that isn't needed. Your code should check that its inputs are valid.

The constructor of the `Matrix` class should allocate the data, while the destructor should free the data. This highlights another problem with your code: you don't free the arrays. In commercial software, forgetting to free memory can be disastrous since the system will eventually run out of memory. Therefore we need to pass the size of the array to the constructor, and since we are writing a library class, we might as well cater for non-square matrices as well. Your code did not work reliably because you did not allocate the first array correctly, it should be an array of size N of arrays of size N. Note also how the array is deleted in the destructor, it is important to use the `delete []` notation when deleting an array, since this will matter when objects in the array have destructors.

One should hide the internal data of the class by making it private, and access and manipulate the data via accessor methods. One can overload the `[]` operator to access the cells of the matrix, so that if you have a matrix `m`, you can write `m[x]` to access a column in your matrix, and therefore `m[x][y]` to access a particular cell. So here is a `Matrix` class that allocates the data, and provides accessor methods:

```
class Matrix1 {
  unsigned width, height;
  double **cols;
public:
  Matrix1(unsigned w, unsigned h)
      : width(w), height(h) {
    cols = new double*[height];
    for(unsigned i=0; i<width; ++i)
      cols[i] = new double[height];
  }
  ~Matrix1() {
    for(unsigned i=0; i<width; ++i)
      delete [] cols[i];
    delete [] cols;
  }
  double *operator[](unsigned col) {return cols[col];}
  unsigned get_width()  { return width; }
  unsigned get_height() { return height; }
};
```

So we're done right? Not by a long way! The most important problem is safety, and at the moment the copy constructor and the assignment operator don't work correctly, so writing

```
a = b;
Matrix x = y;
```

will crash the program. The compiler provides default implementations that will in this case do the wrong thing. It will only copy the pointer, so some arrays will not be freed, whilst other arrays will be freed twice. The constructor of `Matrix` is not exception safe, and could leak memory on an exception. Making the class robust and writing safe copy constructors and assignment operators would be straightforward, but quite laborious. The code does not initialize the cells in the matrix to zero, which would be a nice feature of the constructor.

You can replace C-style arrays with `std::vector`, and your code becomes

```
class Matrix2 {
  unsigned width, height;
  std::vector<std::vector<double> > cols;
public:
  Matrix2(int w, int h) : width(w), height(h),
      cols(w, std::vector<double>(h)) {}
  std::vector<double> &operator[](unsigned col)
                        { return cols[col]; }
  unsigned get_width()  { return width; }
  unsigned get_height() { return height; }
};
```

This code is much shorter, clearer, nicer and safer, since all of the functionality we need has already been implemented by the `vector` class, and all of the default methods like copy construction, assignment and destruction are all taken care of by the `vector`. The `:` notation in the `Matrix` is used to initialize its members, which initializes the array with `w` elements each a `vector` of length `h`, and the `vector` initializes the contents to zero.

The moral of the story is that one should almost never use pointers and arrays, always use STL containers and iterators. Because `Matrix` is a generic container, it should comply with the norms of the STL as much as possible. This allows it to be used with STL-compliant algorithms, and will make it easier to use and understand. So it needs as much standard functionality as appropriate, such as a `begin()`, `end()`, `iterator`, `const_iterator`, `reverse_iterator`, `swap()`, `at()` and operators like `+`, `-`, `*`. One should provide `const` and non-`const` versions of methods, so that the container is usable when `const`. Implement operators `<<` and `>>` to read and write the matrix to a stream. Like other containers, it should be templated on the type it contains, so that we could could have a matrix of any type of value, such as `integer`s, `bool`s, `float`s, `std::complex` or even other matrices. One might also perform bounds checking.

Although `std::vector` provides a perfectly acceptable solution, the C++ STL has `std::valarray`, that is intended specifically for writing multi-dimensional arrays. It is much more versatile since it can use "slices", linear subsets of an array, based upon FORTRAN's BLAS (Basic Linear Algebra Subprograms) library, offering high-performance multi-dimensional array manipulation. Stroustroup [1] provides an implementation of a `Matrix` class based upon `valarray` and slices, so I do not need to repeat it here. A slice can represent any linear subsequence of an array, so can represent both a row and a column. However it may be easier to organize the array into columns and just return the correct offset into the array.

```
T *operator[](unsigned col)
                        { return &data[height*col]; }
const T *operator[](unsigned col) const
                        { return &data[height*col]; }
```

If the size of the matrix is fixed, it may be better to specify the dimensions of the matrix in template parameters. This has the advantage that the compiler can then generate optimal code by unrolling loops, and the contents of the matrix can be stored inside the matrix object itself, rather than performing additional memory allocation and deallocation.The crucial benefit is that the dimensions of the matrices can be checked by the compiler, so that matrices of the wrong sizes are prevented from being added or multiplied, and the compiler can check that a matrix is square for certain operations. It is much better to catch errors at compile-time than run-time.

```
template<typename T, unsigned W, unsigned H=W>
class Matrix3 {
  T cells[W][H];
public:
  T *operator[](unsigned col) { return cells[col]; }
  const T *operator[](unsigned col) const
                              { return cells[col]; }
  unsigned get_width() const  { return W; }
  unsigned get_height() const { return H; }
};
template<typename T, unsigned W,
         unsigned H, unsigned N>
Matrix3<T,W,H> operator*(const Matrix3<T, N, H> &m1,
                         const Matrix3<T, W, N> &m2) {
  Matrix3<T,W,H> result;
  for(unsigned i=0; i<W; ++i) {
    for(unsigned j=0; j<H; ++j) {
      T sum = T(); // Initializes to zero
      for(unsigned k=0; k<N; ++k)
        sum += m1[k][i] * m2[j][k];
      result[i][j] = sum;
    }
  }
  return result;
}
```

References

1. Bjarne Stroustrup, *The C++ Programming Language, 3rd Ed*, Addison Wesley 1997.

From Seyed H. Haeri <shhaeri@math.sharif.edu>

The first thing which jumps out at me as I skim through the code is the lack of any (appropriate) commenting. The code has got no comments at all, which means a big drawback to any code. The student would lose a large amount of marks if he/she was a student of mine. Not only each mentally separate piece of code needs its own comment(s), but also the code needs to be commented – say at the beginning of the program – for its (potential) reader about what it's generally supposed to do. The next overall point about this code is that the student has well tested the program, yet he/she hasn't developed any diagnosis. He/she could have done that say by observing the size of what he/she allocates (using the `sizeof` operator, for example).

I then start wondering whether this code really flawlessly gets compiled under any standard conforming implementation. The student has used `std::cout` as well as `std::cin`, yet he/she hasn't anywhere told the compiler that he/she means the `cout` and `cin` of `std`.

Another matter of style is the poor way of program interaction with its user. I'll delve deeper into that throughout the criticism below. For the moment, however, especially for students, I say that it is a very good practice to get used to let the user know what the program is supposed to do. This could easily be done using a short series of initial prompts to the user before he/she starts the I/O process. Afterwards, let's go through the code. The first line:

```
#include <iostream.h>
```

should be re-written as:

```
#include <iostream>
```

I say that because the Standard has allowed the header files to be implemented with extensions other than `.h` (§16.2, Phrase 5 and §17.4.1.2, Footnote 158).

The next line, in fact, led me into doubt. I checked the Standard to see whether there really is such a standard header. And, there is not. Thus

```
#include <process.h>
```

should be fully omitted. The following line

```
void main(void)
```

although may work under many implementations, is not portable (§3.6.1, Phrase 2 of 98 Standard).

```
int N, i, j, k;
```

This line should not be written here. Variables should be declared as close as possible to where they got used. Furthermore, as far as I can see, those variables are all supposed to hold sizes. Therefore, it's quite irrational to prefer `int` to `size_t` for their type. I'll again come to that as I proceed through the code.

```
double **A, **B, **C;
```

Assuming that the decision of playing with matrices using `double**`s is a good decision – which turns out not to be so – for the mere sake of extendibility, should be replaced by:

```
typedef double** Matrix;
Matrix A, B, C;
```

This way, the code will work easily by merely changing the `typedef` as he/she decides to change the data structure using which he/she wants to play with matrices.

Another big mistake is:

```
double sum = 0.0;
```

here. I'll mention in the following where it is best to do that.

```
using std::cout;
using std::cin;
cout << "Dimension of Matrix? " << endl;
size_t N;
cin >> N;
```

As you can see, I've added statements as per what I've previously spoken about. Another point about any input is to check the input stream for (possible) errors. Hence:

```
if(!cin) { ... }
```

(Perhaps the use of some exceptions.)

```
A = new (double *) [N];
B = new (double *) [N];
C = new (double *) [N];
```

When you do:

```
A = new (double *);
```

what you get is a *single* pointer to a pointer to `double`, and not an *array* of pointers to pointer to `double`. To get the latter, you need to write what I have.

I tried a lot to find out why the student has come to the observation that it works for `N = 2`, whilst it does not for `N = 3`. The only reasonable guess of mine is the following snippet from the Standard (§18.4.1.2, Footnote 211):

> "…The array new expression, may, however, increase the size argument to `operator new[] (std::size_t)` to obtain space to store supplemental information."

That is, I think for `N = 2`, the above *space for storing supplemental information* allocated invisibly by VC++ suffices, whilst it does not suffice for `N = 3`. I guess, furthermore, that for little `N`'s – which are much likely to suffice for the student's test cases – cygwin does a similar job in allocating a space which happens to be satisfactory.

Another important point, the necessity of which may not become that obvious at academy, but somehow plays a vital role in commercial programming, is the validation of any try for allocation. This means that, under normal conditions such as that of ours, any such try should be put in a `try`/`catch` block.

```
for(size_t i = 0; i < N; ++i) {
  A[i] = new double[N];
  B[i] = new double[N];
  C[i] = new double[N];
}
```

Yep! That's right. I've defined `i` inside the `for` body. The reason is what I've already mentioned; variables should always be declared as close to their application as possible. Furthermore, they should not be present outside the scope they are supposed to function.

Anybody having a little experience of overloading in C++ knows enough why should one always prefer the prefix `operator ++` to its postfix counterpart. Although you may argue that, in this case, we're dealing with built-in types, and no modern compiler may leave optimising it off, I insist on what I told. Why? 'Cause of two important points: First, this kind of optimisation – although quite common – is not guaranteed. Second, it is a good practice to get used to that. Having that done, you'll never lose efficiency whilst dealing with objects constructing/destructing of which is much more than wasteful.

Again this `for` body should all lie in a `try`/`catch` block. Let's go further.

```
cout << "Enter matrix A: (Please enter each
        row in one line.)" << endl;
for(size_t i = 0; i < N; ++i) {
  for(size_t j = 0; j < N; ++j)
    cin >> A[i][j];
  cin.get();
}
```

There are many ways for inputting a matrix. The one chosen by the student is not appropriate however. What's wrong with it? It prompts something to the user each time, and asks him/her to enter each element again and again. One plausible way seems to be that of mine, in which I ask the user only one time about what he/she is supposed to do. This way has got another advantage, and that's the fact that it well equally works for when we want to input from files. Furthermore, human beings find it much more natural as they're working with 2 by 2 matrices. We should then check the input stream for possible errors. The next `for` body should be replaced with a similar one like above.

```
for(size_t i = 0; i < N; ++i)
  for(size_t j = 0; j < N; ++j) {
    double sum = 0.0;
    for(size_t k = 0; k < N; ++k)
      sum += A[i][k] * B[k][j];
    C[i][j] = sum;
  }
```

That is, I've defined `sum` at the best possible position. Anywhere outside this body `sum` would be meaningless. After a few blank lines in output

```
cout << "A * B = " << endl;
for(size_t i = 0; i < n; ++i) {
  std::copy(C[i][0], C[i][N],
    std::ostream_iterator<double>(cout, " "));
  cout << endl;
}
```

Here, I've suited the already-at-hand tool of the Standard, `std::copy()`. Note that this needs the addition of `#include <algorithm>` as well as `#include <iterator>` at the top of the program.

And another extremely important point which this student – like many other newbies – has forgotten is to `delete[]` the allocated memory. That is:

```
  for(size_t i = 0; i < N; ++i) {
    delete[] A[i];
    delete[] B[i];
    delete[] C[i];
  }
  delete[] A;
  delete[] B;
  delete[] C;
```

And, the rest of the code which has not been included in the journal...

Assuming that there is no better candidate than raw pointers, I recommend the student to reconsider the code. Yes, the code is quite trivial. But, it's a mixture of many different creatures. It does its input, it constructs its own objects, it performs the multiplication, it then outputs the resulting matrix, and finally, it destructs the objects. These are the major steps, not? This is a very good point showing us the necessity of splitting the code into different functions with names indicating what's intended. Here is what will be the result: (All the following code is off-hand. The deadline is close, and I've got to finish the criticism. So, please don't be fussy.)

```
  int main() {
    cout << "This programme ...";
    cout << "Dimension of matrices? ";
    size_t N;
    cin >> N;
    Matrix A, B, C;
    Construct(A, N);
    Construct(B, N);
    Construct(C, N);
    Input(A, cin);
    Input(B, cin);
    C = Multiply(A, B);
    Output(C, cout);
    Destruct(A);
    Destruct(B);
    Destruct(C);
    return 0;
  }
```

I'm wondering whether there could ever be a guy aware of OOP, whom the above code does not whet appetite for assembling a class – an ADT, in other words – which serves the job much neater.

In fact, considering the very little code above, one should have gotten quite sure that playing with raw pointers also very dangerous, is very cumbersome. A well arisen question then is that isn't there any facility in C++ which can ease the job? Oh yes, there are. You can say use their majesty `vector<>`s. This way, you get rid of all the allocation, evaluation of allocation, and deallocation stuff. All of those bothers are now settled by the aids of automatically served features of `vector<>`. This way, you should end up with something like

```
  int main() {
    cout << "Dimension? ";
    Matrix<double>::size_type n;
    cin >> n ;
    Matrix<double> A(n), B(n), C(n);
    cout << "Enter A:" << endl;
    cin >> A;
    cout << "Enter B:" << endl;
    cin >> B;
    C = A * B;
    cout << "A * B = " << endl << C;
    return 0;
  }
```

Do you see what's happened? You've ended up with nothing apart from the abstract problem at hand. Full stop. Wow!

The Winner of SCC 32

The editor's choice is:

Calum Grant

Please email `francis@robinton.demon.co.uk` to arrange for your prize.

Guest Commentary – Alan Griffiths

`<alan@octopull.demon.co.uk>`

When faced with code like this it is difficult to know where to start – inappropriate choice of headers, ignorance of what the standard mandates (`void main()`!), anti-idiomatic usage (choice of variable names and scope, memory management "by hand"), bad design and just plain bugs. There is also the question of assessing what the student understands – clearly suggesting writing a `matrix` class won't help a student that apparently hasn't even caught onto using functions to factor out repetitive code.

I'll mention the fundamental problem that often occurs with more experienced developers: too much code has been written without giving thought to finding out if it works. It appears that while the student has some test input (and expected results?) against which to run the program whole program, she is at a loss as to how to identify which parts of the program are (or are not) working. If the student exhibited a better knowledge of the language one might suggest that the program be broken down into pieces. (Indeed, the student should have been introduced to functions before this point.)

However, introducing functions and user defined types is a long road that doesn't address the immediate problem of getting the program working in a way the student understands. And this student has identified a prime suspect: incorrect use of `new` – C++ is unforgiving of developers that use language features they don't understand. It is rarely the case that arrays should be allocated using `new`, and in this occasion `new` is not an appropriate solution. So, I'm going to look at this code with a view to showing how `new` and all its pitfalls can be avoided.

First let's show the code to a compiler:

```
Compiling source file(s)...
main.cpp
In file included from C:\MinGWStudio\MinGW\include\
c++\3.3.1\backward\iostream.h:31,
from main.cpp:1:
C:\MinGWStudio\MinGW\include\c++\3.3.1\backward\back
ward_warning.h:32:2: warning: #warning This file
includes at least one deprecated or antiquated
header. Please consider using one of the 32 headers
found in section 17.4.1.2 of the C++ standard.
Examples include substituting the <X> header for the
<X.h> header for C++ includes, or <sstream> instead
of the deprecated header <strstream.h>. To disable
this warning use -Wno-deprecated.
main.cpp:4: error: 'main' must return 'int'
main.cpp:4: error: return type for 'main' changed to
'int'
scc24.exe - 2 error(s), 1 warning(s)
```

Before we continue, I'll fix these problems. The header `<iostream.h>` refers to a pre-standard library distribution, and `<process.h>` is a posix header irrelevant to the current program. Replace these with:

```
#include <iostream>
#include <vector>
```

The latter, `<vector>`, isn't needed yet, but I'll use it shortly to provide a dynamically sized array in place of the current heap allocations.

The remaining diagnostics indicate that the corrected signature for main is:

```
int main()
```

Making that change and back to the compiler:

```
Compiling source file(s)...
main.cpp
main.cpp: In function 'int main()':
main.cpp:10: error: 'cout' undeclared (first use
this function)
main.cpp:10: error: (Each undeclared identifier is
reported only once for each function it appears in.)
main.cpp:10: error: 'endl' undeclared (first use
this function)
main.cpp:11: error: 'cin' undeclared (first use this
function)
scc24.exe - 4 error(s), 0 warning(s)
```

There are several ways to fix these diagnostics. My usual preference is to use the fully qualified names, but it will probably confuse the student less to employ `using` definitions. At the top of `main`, add:

```
using std::cin;
using std::cout;
using std::endl;
```

And now we have some code that compiles – and Comeau (`http://www.comeaucomputing.com/tryitout/`) is happy with it too.

Next, to eliminate those suspicious uses of `new` – there is a far better option – `std::vector`. Replace the declarations of `A`, `B` and `C` with aliases

[concluded at foot of next page]

Features

Patterns in C – Part 2: State

Adam Petersen <adampetersen75@yahoo.se>

Every non-trivial program passes through a number of different states during its lifecycle. Describing this lifecycle as a finite state machine is a simple and useful abstraction. In this part of the series, we will investigate different strategies for implementing state machines. The goal is to identify mechanisms that let the code communicate the intent of expressing the problem as a finite state machine.

Traditional Solution with Conditionals

Consider a simple, digital stop-watch. In its most basic version, it has two states: started and stopped. A traditional and direct way to implement this behaviour in C is with conditional logic in the shape of `switch`/`case` statements and/or `if`-`else` chains.

The digital stop-watch in this example is implemented as a First-Class ADT [1].

```
typedef enum { stopped, started } State;

struct DigitalStopWatch {
  /* Let a variable hold the state of our object. */
  State state;
  TimeSource source;
  Display watchDisplay;
};

void startWatch(DigitalStopWatchPtr instance) {
  switch(instance->state) {
    case started:
      /* Already started -> do nothing. */
      break;
    case stopped:
      instance->state = started;
      break;
    default: error("Illegal state"); break;
  }
}
```

[continued from previous page]

for `vector` and `matrix` types as follows:

```
typedef std::vector<double> vector;
typedef std::vector<vector> matrix;
```

Finally, replace the memory allocation code with:

```
matrix A(N, vector(N));
matrix B(N, vector(N));
matrix C(N, vector(N));
```

Now, magically, the program works! Let's look at the whole thing:

```
#include <iostream>
#include <vector>
int main() {
  using std::cin;
  using std::cout;
  using std::endl;
  typedef std::vector<double> vector;
  typedef std::vector<vector> matrix;
  int N, i, j, k;
  double sum = 0.0;
  cout << "Dimension of Matrix ?" << endl;
  cin >> N;
  matrix A(N, vector(N));
  matrix B(N, vector(N));
  matrix C(N, vector(N));
  for(i=0; i<N; i++)
    for(j=0; j<N; j++) {
      cout << "A[" << i << "][" << j << "] = ?"
           << endl;
      cin >> A[i][j];
    }
  for(i=0; i<N; i++)
    for(j=0; j<N; j++) {
      cout << "B[" << i << "][" << j << "] = ?"
           << endl;
      cin >> B[i][j];
    }
  for(k=0; k<N; k++)
    for(i=0; i<N; i++) {
      sum = 0.0;
    for(j=0; j<N; j++)
      sum += A[i][j]*B[j][k];
    C[i][k] = sum;
  }
  cout << endl << endl << endl;
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      cout << "C[" << i << "][" << j << "] = "
           << C[i][j]<< endl;
}
```

Actually, there is still plenty wrong with this – anti-idiomatic usage (choice of names and scope, addiction to `std::endl`), bad design and bugs. In short, it is still suitable as an entry for a "Student Code Critique"! On the other hand, the student does not appear ready to deal with these problems (yet!) and should learn that there are easier solutions to attempt to manage memory by hand.

Student Code Critique 33

(Submissions to `scc@accu.org` by May 10th)

Special thanks to Richard Corden for providing us with a snippet he came across.

I'm having a problem whose cause I'm not able to detect. I sometimes end up in the `true` *block of the* `if` *statement where* `iter->first` *is not* `5`*. Could you explain me what is going wrong?*

```
#include <map>
#include <algorithm>
typedef std :: multimap <int, int> MyMapType;
// Filter on values between 5 and 10
struct InRange {
  bool operator ()(
       MyMapType::value_type const & value) const {
    return (value.second > 5) && (value.second < 10);
   }
};

// Not really important how this happens.
void initMap (MyMapType & map);

int main () {
  MyMapType myMap;

  // initialise the map...
  initMap (myMap);
  MyMapType::iterator lower = myMap.lower_bound(5);
  MyMapType::iterator iter = std :: find_if(
           lower, myMap.upper_bound(5), InRange());

  // Did we find this special criterial?
   if (iter != myMap.end()) {
    // Yup...we have a value meeting our criteria
  }
  else {
  }
}
```

```
void stopWatch(DigitalStopWatchPtr instance) {
  switch(instance->state) {
    case started:
      instance->state = stopped;
      break;
    case stopped:
      /* Already stopped -> do nothing. */
      break;
    default: error("Illegal state"); break;
  }
}
```

While this approach has the advantage of being simple and easy to understand, it introduces several potential problems:

1. **It doesn't scale.** In large state machines the code may stretch over page after page of nested conditional logic. Imagine the true maintenance nightmare of changing large, monolithic segments of conditional statements.
2. **Duplication.** The conditional logic tends to be repeated, with small variations, in all functions that access the state variable. As always, duplication leads to error-prone maintenance. For example, simply adding a new state implies changing several functions.
3. **No separation of concerns.** When using conditional logic for implementing state machines, there is no clear separation between the code of the state machine itself and the actions associated with the various events. This makes the code hide the original intent (abstracting the behaviour as a finite state machine) and thus makes the code less readable.

A Table-based Solution

The second traditional approach to implement finite state machines is through transition tables. Using this technique, our original example now reads as follows.

```
typedef enum {
  stopped,
  started
} State;

typedef enum {
  stopEvent,
  startEvent
} Event;

#define NO_OF_STATES 2
#define NO_OF_EVENTS 2

static State
TransitionTable[NO_OF_STATES][NO_OF_EVENTS] = {
  { stopped, started },
  { stopped, started } };

void startWatch(DigitalStopWatchPtr instance) {
  const State currentState = instance->state;
  instance->state
        = TransitionTable[currentState][startEvent];
}

void stopWatch(DigitalStopWatchPtr instance) {
  const State currentState = instance->state;
  instance->state
        = TransitionTable[currentState][stopEvent];
}
```

The choice of a transition table over conditional logic solved the previous problems:

1. **Scales well.** Independent of the size of the state machine, the code for a state transition is just one simple table-lookup.
2. **No duplication.** Without the burden of repetitive `switch`/`case` statements, modification comes easily. When adding a new state, the change is limited to the transition table; all code for the state handling itself goes unchanged.
3. **Easy to understand.** A well structured transition table serves as a good overview of the complete lifecycle.

Shortcomings of Tables

As appealing as table-based state machines may seem at first, they have a major drawback: it is very hard to add actions to the transitions defined in the table. For example, the watch would typically invoke a function that starts to tick milliseconds upon a transition to state started. As the state transition isn't explicit, conditional logic has to be added in order to ensure that the tick-function is invoked solely as the transition succeeds. In combination with conditional logic, the initial benefits of the table-based solution soon decrease together with the quality of the design.

Other approaches involve replacing the simple enumerations in the table with pointers to functions specifying the entry actions. Unfortunately, the immediate hurdle of trying to map state transitions to actions in a table based solution is that the functions typically need different arguments. This problem is possible to solve, but the resulting design loses, in my opinion, both in readability as well as in cohesion as it typically implies either giving up on type safety or passing around unused parameters. None of these alternatives seem attractive.

Transition tables definitely have their use, but when actions have to be associated with state transitions, the STATE pattern provides a better alternative.

Enter STATE Pattern

In its description of the STATE pattern, *Design Patterns* [2] defines the differences from the table-based approach as *"the State pattern models state-specific behaviour, whereas the table-driven approach focuses on defining state transitions"*. When applying the STATE pattern to our example, the structure in Figure 1 emerges.

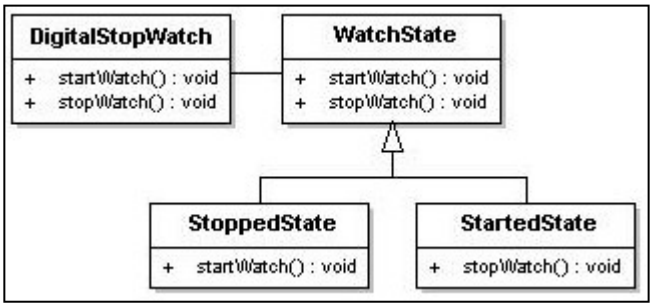


Figure 1: STATE pattern structure

This diagram definitely looks like an object oriented solution. But please don't worry – we will not follow the temptation of the dark side and emulate inheritance in C. However, before developing a concrete implementation, let's explain the involved participants.

- `DigitalStopWatch`: *Design Patterns* [2] defines this as the *context*. The context has a reference to one of our concrete states, without knowing exactly which one. It is the context that specifies the interface to the clients.
- `WatchState`: Defines the *interface* of the state machine, specifying all supported events.
- `StoppedState` and `StartedState`: These are *concrete states* and each one of them encapsulates the behaviour associated with the state it represents.

The main idea captured in the STATE pattern is to represent each state as an object of its own. A state transition simply means changing the reference in the context (`DigitalStopWatch`) from one of the concrete states to the other.

Implementation Mechanism

Which mechanism may be suitable for expressing this clearly object oriented idea in C? Returning to our example, we see that we basically have to switch functions upon each state transition. Luckily, the C language supplies one powerful feature, pointers to functions, that serves our needs perfectly by letting us change the behaviour of an

object at run-time. Using this mechanism, the interface of the states would look as:

Listing 1: The state interface in `WatchState.h`

```
/* An incomplete type for the state representation
   itself. */
typedef struct WatchState* WatchStatePtr;

/* Simplify the code by using typedefs for the
   function pointers. */
typedef void (*EventStartFunc)(WatchStatePtr);
typedef void (*EventStopFunc)(WatchStatePtr);

struct WatchState {
  EventStartFunc start;
  EventStopFunc stop;
};
```

Breaking the Dependency Cycle

After getting used to the scary syntax of pointers to functions, the interface above looks rather pleasant. However, with the interface as it is, a dependency cycle will evolve.

Consider the pointers in the `WatchState` structure. Every concrete state has to define the functions to be pointed at. This implies that each time an event is added to the interface, all concrete states have to be updated. The resulting code would be error-prone to maintain and not particularly flexible.

The good news is that breaking this dependency cycle is simple and the resulting solution has the nice advantage of providing a potential error-handler. The trick is to provide a default implementation, as illustrated in the listing below.

Listing 2: Extend the interface in `WatchState.h`

```
/* ..previous code as before.. */
void defaultImplementation(WatchStatePtr state);
```

Listing 3: Provide the default implementations in `WatchState.c`

```
static void defaultStop(WatchStatePtr state) {
  /* We'll get here if the stop event isn't
     supported in the concrete state. */
}

static void defaultStart(WatchStatePtr state) {
  /* We'll get here if the start event isn't
     supported in the concrete state. */
}

void defaultImplementation(WatchStatePtr state) {
   state->start = defaultStart;
   state->stop = defaultStop;
}
```

Concrete States

The default implementation above completes the interface of the states. The interface of each state itself is minimal; all it has to do is to declare an entry function for the state.

Listing 4: Interface of a concrete state, `StoppedState.h`

```
#include "WatchState.h"
void transitionToStopped(WatchStatePtr state);
```

Listing 5: Interface of a concrete state, `StartedState.h`

```
#include "WatchState.h"
void transitionToStarted(WatchStatePtr state);
```

The responsibility of the entry functions is to set the pointers in the passed `WatchState` structure to point to the functions specifying the behaviour of the particular state. As we can utilize the default implementation, the implementation of the concrete states is straightforward; each concrete state only specifies the events of interest in that state.

Listing 6: `StoppedState.c`

```
#include "StoppedState.h"
/* Possible transition to the following state: */
#include "StartedState.h"

static void startWatch(WatchStatePtr state) {
   transitionToStarted(state);
}

void transitionToStopped(WatchStatePtr state) {
  /* Initialize with the default implementation
     before specifying the events to be handled
     in the stopped state. */
  defaultImplementation(state);
  state->start = startWatch;
}
```

Listing 7: `StartedState.c`

```
#include "StartedState.h"
/* Possible transition to the following state: */
#include "StoppedState.h"

static void stopWatch(WatchStatePtr state) {
  transitionToStopped(state);
}

void transitionToStarted(WatchStatePtr state) {
  /* Initialize with the default implementation
     before specifying the events to be handled
     in the started state. */
  defaultImplementation(state);
  state->stop = stopWatch;
}
```

Client Code

The reward for the struggle so far comes when implementing the context, i.e. the client of the state machine. All the client code has to do, after the initial state has been set, is to delegate the requests to the state.

```
struct DigitalStopWatch {
  struct WatchState state;
  TimeSource source;
  Display watchDisplay;
};

DigitalStopWatchPtr createWatch(void) {
  DigitalStopWatchPtr instance
          = malloc(sizeof *instance);
  if(NULL != instance) {
    /* Set the initial state. */
    transitionToStopped(&instance->state);
    /* Initialize the other attributes here. */
  }
  return instance;
}

void destroyWatch(DigitalStopWatchPtr instance) {
  free(instance);
}

void startWatch(DigitalStopWatchPtr instance) {
  instance->state.start(&instance->state);
}

void stopWatch(DigitalStopWatchPtr instance) {
  instance->state.stop(&instance->state);
}
```

A Debug Aid

In order to ease debugging, the state structure may be extended with a string holding the name of the actual state. Example:

```
void transitionToStopped(WatchStatePtr state) {
  defaultImplementation(state);
  state->name = "Stopped";
  state->start = startWatch;
}
```

Utilizing this extension, it becomes possible to provide an exact diagnostic in the default implementation. Returning to our implementation of `WatchState.c`, the code now looks like:

```
static void defaultStop(WatchStatePtr state) {
  /* We'll get here if the stop event isn't
     supported in the concrete state. */
  logUnsupportedEvent("Stop event", state->name);
}
```

Extending the State Machine

One of the strengths of the STATE pattern is that it encapsulates all state-specific behaviour making the state machine easy to extend.

- **Adding a new event.** Supporting a new event implies extending the `WatchState` structure with a declaration of another pointer to a function. Using the mechanism described above, a new default implementation of the event is added to `WatchState.c`. This step protects existing code from changes; the only impact on the concrete states is on the states that intend to support the new event, which have to implement a function, of the correct signature, to handle it.
- **Adding a new state.** The new, concrete state has to implement functions for all events supported in that state. The only existing code that needs to be changed is the state in which we'll have a transition to the new state. Please note that the STATE pattern preserves one of the benefits of the table-based solution: client code, i.e. the context, remains unchanged.

Stateless States

The states in the sample code are stateless, i.e. the `WatchState` structure only contains pointers to re-entrant functions. Indeed, this is a special case of the STATE pattern described as *"If State objects have no instance variables […] then contexts can share a State object"* [2]. However, before sharing any states, I would like to point to Joshua Kerievsky's advice that *"it's always best to add state-sharing code after your users experience system delays and a profiler points you to the state-instantiation code as a prime bottleneck"*[3].

In the C language, states may be shared by declaring a static variable representing a certain state inside each function used as entry point upon a state transition. As the variables now have permanent storage, the signature of the transition functions is changed to return a pointer to the variable representing the particular state.

Listing 8: Stateless entry function, `StartedState.c`

```
WatchStatePtr transitionToStarted(void) {
  static struct WatchState startedState;
  static int initialized = 0;
  if(0 == initialized) {
    defaultImplementation(&startedState);
    startedState.stop = stopWatch;
    initialized = 1;
  }
  return &startedState;
}
```

The client code has to be changed from holding a variable representing the state to holding a pointer to the variable representing the shared state. Further, the context has to define a callback function to be invoked as the concrete states request a state transition.

Listing 9: Client code for changing state

```
void changeState(DigitalStopWatchPtr instance,
WatchStatePtr newState) {
  /* Provides a good place for controls and trace
     messages (all state transitions have to go
     through this function). */
  instance->state = newState;
}
```

The stateless state version comes closer to the STATE described in *Design Patterns* [2] as a state transition, in contrast with the previous approach, implies changing the object pointed to by the context instead of just swapping its behaviour.

Listing 10: State transition in `StoppedState.c`

```
static void startWatch(DigitalStopWatchPtr context) {
  changeState(context, transitionToStarted());
}
```

A good quality of the stateless approach is that the point of state transitions is now centralized in the context. One obvious drawback is the need to pass around a reference to the context. This reference functions as a memory allowing the new state to be mapped to the correct context. Another drawback is the care that has to be taken with the initialization of the static variables if the states are going to live in a multithreaded world.

Consequences

The main consequences of applying the STATE pattern are:

1. **Reduces duplication introduced by complex, state-altering conditional logic.** As illustrated in the example above, solutions based upon large segments of conditional logic tend to contain duplicated code. The STATE pattern provides an appealing alternative by removing the duplication and reducing the complexity.
2. **A clear expression of the intent.** The context delegates all state dependent operations to the state interface. Similar to the table-based solution, the STATE pattern lets the code reflect the intent of abstracting the problem as a finite state machine. With complex, conditional logic, that intent is typically less explicit.
3. **Encapsulates the behaviour of each state.** Each concrete state provides a good overview of its behaviour including all events supported in that very state. This encapsulation makes it easy both to identify as well as updating the relevant code when changes to a certain state are to be done.
4. **Implicit error handling.** The solutions based on conditional logic, as well as the table-based one, requires explicit code to ensure that a given combination of state and event is valid. Using the technique described above of initializing with a default implementation, the controls are built into the solution.
5. **Increases the number of compilation units.** The code typically becomes less compact with the STATE pattern. As *Design Patterns* says "such distribution is actually good if there are many states" [2]. However, for a trivial state machine with few, simple states, the STATE pattern may introduce an unnecessary complexity. In that case, if it isn't known that more complex behaviour will be added, it is probably better to rely on conditional logic so the logic will be easy to follow.

Summary

The STATE pattern lets us express a finite state machine, making the intent of the code clear. The behaviour is partitioned on a per-state basis and all state transitions are explicit.

The STATE pattern may serve as a valuable tool when implementing complex state-dependent behaviour. On the other hand, for simple problems with few states, conditional logic is probably just right.

Next Time

We'll continue with *Design Patterns* [2] and investigate the STRATEGY pattern, which is closely related to STATE. The STRATEGY pattern lets us implement different variation points of an algorithm, interchangeable at run time.

Adam Petersen

Acknowledgements

Many thanks to Magnus Adamsson, Tord Andersson, and André Saitzkoff for their feedback.

References

1. Adam Petersen , "Patterns in C, part 1", *C Vu 17.1*
2. Gamma, E., Helm, R., Johnson, R., and Vlissides, J, *Design Patterns*, Addison-Wesley
3. Joshua Kerievsky , *Refactoring to Patterns*, Addison-Wesley

A Subversion Primer

Pete Goodliffe <pete@cthree.org>

This article provides an introduction to Subversion. It lists Subversion's key features and best working practices, provides cookbook recipes for all the common activities in your day-to-day development work, and points the reader to further reading.

It is intended for software developers and doesn't matter which operating system you use (although it would help to not be scared of the command line). It's useful to have an understanding of CVS, or some other version control system.

Disclaimer: If the information in here diverges from Subversion documentation, clearly I'm wrong and they're right. Terms and conditions apply. Your hair may be at risk if you do not keep up a loan secured on it.

Terms and Definitions

SCMS – Source Code Management System (e.g. CVS, ClearCase, or Subversion)
CM – Configuration Management (development practices using SCMS)
CVS – Concurrent Versions System, the *de facto* open source SCMS

What is Subversion?

Subversion was designed from the ground up as a modern, high-performance version control system. It is intended to be compelling SCMS replacement for the (now ageing) CVS[1]. It is freely available under an open source licence.

It is gaining popularity, a stable, production-quality system. It's now used in anger by public projects including Mono, Xiph, Apache, Samba, PuTTY, Debian, and Ethereal. I have used it for personal work, and deployed it in commercial development teams. I've found it to be an excellent, streamlined SCMS that is very usable.

Of the new breed of open SCMS tools around, this is really the only mature production worthy system.

Subversion follows the same *modify-merge-commit* model of CVS, so it should be familiar to most developers.

Key Features

- CVS-like interface
- Directories, renames, and file metadata are versioned
- Commits are truly atomic, they either totally succeed or totally fail
- Commits are recorded as a *changeset* (a revision number applies to the whole file tree, not individual files)
- Branching and tagging are cheap (constant time) operations
- Efficient network usage
- Cross platform: Windows, Linux, MacOS X, with several GUI front-ends
- Good for offsite work (supports offline diff and revert)
- Scales far better than CVS
- Efficient binary file handling
- Supports Apache and the WebDAV protocol (also has own protocol, which can be tunnelled over ssh)
- MS Visual Studio integration

What It Doesn't Do

The feature sets of different SCMSs vary, although the core concepts (i.e. versioning a set of files) don't tend to differ significantly. There are a number of key facilities that Subversion does not provide that you may be used to:

- Merge support is not so rich as some tools (it is as good as CVS, though)
- There are no *dynamic views* (a ClearCase magic versioned file system)
- There are no locking checkouts (strangely, they're working on this)

Depending on your SCMS religion, the last two points are actually huge benefits.

Overview of Operation

Versioning

Subversion versioning is different from the CVS model. Unlike CVS, changes are atomic. When you check in multiple files you submit an atomic *changeset*. These file commits will either all succeed or all fail.

All the modifications made in one changeset are held together as a single revision, with a single checkin message.

1 For this reason I often compare a Subversion operation to the CVS counterpart in this document.

Revision Names

When you invoke Subversion commands, there are a number of special revision names that you can use:
`HEAD` – the latest revision in the repository.
`BASE` – a pristine copy of the file currently checked out in your working copy.
`COMMITTED` – the last revision in which a file changed, before (or at) `BASE`.
`PREV` – the file that is identified as (`COMMITTED-1`).

Important difference: a revision number applies to the *whole tree*, unlike CVS where version numbers apply to each individual file. When you check in a single file, the whole file tree gets up-versioned (the files in your working copy *do not* all automatically get up-versioned, though). A revision number identifies how a file tree looked at a position in time.

For example: you check out `foo.c` from `HEAD`, and the repository is currently at version 5. You have 'version 5' of the file in your working copy. What you really have is `foo.c` *as it appeared* in version 5 *of the repository*. The file may not have changed since version 3. So consecutive version numbers of a single file may be identical.

The changeset idea is very powerful. Using it there is no need for a manually maintained changelog (updates) file, and it's easy to merge specific changesets from one branch to another.

Branches and Tags

Committing a single file effectively creates a new copy of the entire file tree in the repository (remember: the whole source tree goes up a version number). To do this Subversion provides very cheap copy operations. Subversion branches and tags (labels) exploit this fact, and so are also very cheap and quick operations. Their implementation is surprising to CVS developers.

Both branches and tags are stored in the repository as *copies* of a file set. Therefore, they exist at a physical point in the file structure. This is very different from the CVS branching/tagging model. A Subversion tag *is not* a property applied to a particular revision of a file, although it does still uniquely identify a set of files.

Both tags and branches are created in the same way, and the difference between the two is only what you do with them afterwards. A branch is a copy created with the express purpose of performing additional development work, work that should be kept separate from the trunk. A tag is a copy that will not be worked on (you can still merge changesets into a tag copy, in order to 'move' the tag).

Branches and tags can have finite lifetimes – as versioned objects in the repository you can delete them like any other object. This will not lose the history of their existence, and is a powerful tool that helps to keep your repository neat.

Repository Directory Layout

There is no fixed directory structure in a repository, although there are conventions that Subversion users generally follow. A repository may contain a number of projects, each in their own top-level directory. A project directory has three subdirectories:
`trunk` – contains the main line of code development
`branches` – contains all branch copies of code development
`tags` – contains all tag copies of code development
The `branches` and `tags` directories both hold Subversion copies of a version of the `trunk` file set.

Terms

Repository – the central store of files under revision control.
Working copy – a set of files checked out of the repository, stored on your local hard disk.
Revision – a set of changes to the repository (file edits, directory changes, metadata changes). A revision number applies to the whole repository file tree, not to individual files.
Tag – a read-only copy of a revision of the file tree.
Property – arbitrary (possibly binary) metadata associated with a revision of a file in the repository.
Trunk – the repository directory containing the main line of code development.
Branch – a repository copy of a version of the trunk. New commits in this directory only affect the branch, not the trunk.

Miscellaneous

- When you check out a working copy of the repository, Subversion makes a physical copy of the files on your hard disk. Each directory contains a `.svn` directory (akin to the CVS directory in CVS) where admin files are held. You must *not poke* around in here.
  In that directory is held a pristine copy of the files you have checked out, so network utilisation is very efficient – file diffs are done at the client end, not on the server.
- Subversion uses URIs to define a location in the repository. There are several different repository access methods, identified by the URI's transport (e.g. `file:` `svn:` `http:` and `https:`, you only use one of these at once!)
  On Windows, URIs still use the Unix-like forward slash.
- Most operations are applied to the working copy's file tree, in which case the modification is made locally, but no change occurs in repository until you perform a commit. Some operations also accept repository URIs; these work directly on the repository files (the `copy` command is an example of this). In this case you do not even need to check out a working copy.

Subversion Cookbook

Here are some simple Subversion recipes to get you started. This should cover most of your day-to-day work. If you want to step beyond these examples, you can always type `svn help` for more information, or look at the *Further Reading* section, below.

These recipes show how to perform each operation using the Subversion command line client (`svn`). The exercise is similar using one of the available GUI front ends (for example the MS Visual Studio *Ankh* plug-in); naturally you don't type a command, you click on helpful buttons instead. But despite all the GUI goodness you still have to understand what's going on behind the scenes – these recipes will make this clear. The *Other Tools* section towards the end of this article describes the various Subversion GUI front ends available.

In the following command line examples, the text you type is emboldened, the svn response is plain text. The repository URI location is represented as `REPOSITORY`, and I assume that the URI's transport is method `svn:`. Change this as appropriate.

Subversion commands have both a long form (e.g. `checkout`) and a shortened form (`co`). I use the long form for clarity. The shortened versions are listed at the end of the document (see *Shortened Commands*).

Import a Project

Admittedly you don't do this very often, but it's a quick way to get a baselined project into a Subversion repository[2].

- You want to import the *widgetizer* project into the repository.
- You have made a top-level `widgetizer` directory structure in the repository (see below for that recipe).

```
svn import /path/to/widgetizer
svn://REPOSITORY/widgetizer/trunk
Adding      widgetizer/main.c
Adding      widgetizer/other.c
Committed revision 1
```

These changes are made to the repository immediately. Don't worry if you import to the wrong place, in Subversion it's easy to move things around afterwards (that's a later recipe).

View the Repository

Before you 'check out' files from the repository you can browse to see what you want to play with. (CVS doesn't provide this facility.)

```
svn list svn://REPOSITORY/widgetizer
branches/
tags/
trunk/
```

Clearly, if there is a ViewCVS system (a web-based repository browse application) set up, you would use that in preference.

Check Out Part of a Repository

- You've identified the project subdirectory you want to check out, and know whether you want the trunk or a branch.

Selecting Revisions

Many commands can be directed to work with a specific revision of a file (remember, though, that revision numbers actually apply to the *repository*, and not to *individual files*). In this case they take a `–revision` (`-r` for short) argument.

This can take two forms:

`-r 5` Specifies files from revision 5 of the repository.

`-r 3:5` Specifies a range of files from version 3 to version 5 of the repository

The version specified can be a revision number, a keyword, or a date. Dates are enclosed in braces, and can take many forms, for example: `{2002-02-17}`, `{15:30}`, `{"2002-02-17 15:30"}` and more. (If the date contains a space, wrap it in quotes.)

```
cd ~/Work
svn checkout svn://REPOSITORY/widgetizer/trunk
A  widgetizer/trunk/main.c
A  widgetizer/trunk/other.c
Checked out revision 1
```

The `A` above stands for 'Added'. We'll see more output like this later. You have now created a *working copy* of the repository.

You can choose to check out a whole project or just a specific file or directory.

You can edit the files immediately; there is no need to issue a Subversion command to 'open' them for editing. Of course, this means that someone else could be editing the same file as you at the same time. More on that later…

Inspect Which Files You've Modified

- You've modified some of the files in your working copy.
- Before you commit the changes to the repository, you want to see what you've changed.

```
cd widgetizer/trunk
svn status
M  main.c
M  other.c
```

The `M` above stands for 'Modified'. Files with no difference from the repository[3] are not listed. The most common status codes are:

M The contents of this file have been changed
? This file is not in the repository (you may want to svn add it)
A File is scheduled for addition
D File is scheduled for deletion
C File has conflicts which need resolving (see below)
S File has been switched to a branch (see below)
! File is managed by Subversion, but it's missing in your working copy

CVS users, note: Subversion doesn't need you to subvert the update command to do this kind of inspection. (CVS users routinely type `cvs –nq update` to list modifications because the output of `cvs status` is not at all helpful.)

You can give a specific filename as an argument to `svn status` to get information on it alone.

Inspect the Changes You've Made

- You know that you've changed a file.
- You want to look at the complete set of changes to compose a suitable checkin message.

```
svn diff other.c
Index: other.c
===================================
—- other.c       (revision 2)
+++ other.c       (working copy)
@@ -1,4 +1,5 @@
 int m;
 int n;
-int p;
+int o;
$
```

You can compare your working copy to a specific repository revision using the `–r` switch, and can compare two repository versions by specifying a range. For example, to see the last change made to `main.c`:

```
svn diff –r PREV:COMMITTED main.c
Index: main.c
   ...
```

2 Tools exist to help you import CVS projects and retain the version/branch history. See the Subversion website for details.

3 Or, more accurately: no difference from the repository version that the working copy was checked out from.

Check In Your Changes

- You've edited a file (or files), and want to check it in to the repository.

```
svn commit --message "Changed the default \
   banana count" other.c
   ... or ...
svn commit --file log-message-file other.c
Sending    other.txt
Transmitting file data .
Committed version 3.
```

If you don't specify a message on the command line, Subversion opens your default editor (as specified by the `EDITOR` environment variable) to prompt you for it.

Note: You can't perform a checkin if someone else has modified the files and checked them in before you. In this case, Subversion will moan at you:

```
svn commit -m "Changed the default banana \
   count" other.c
Sending    other.txt
svn: Commit failed (details follow):
svn: Out of date: 'other.c' in transaction 'k'
```

You must first update to the `HEAD` version (recipe below) and then try to check in.

Add Directories and Files

- You want to add a new file to the widgetizer project.
- You have created it in your working copy.

```
svn add new.c
A  new.c
svn commit -m "Added" new.c
   ...
```

Note that the file doesn't get put into the repository until you issue a `commit` command.

Move and Copy Files

- You want to modify the layout of files within a directory.

```
svn move new.c old.c
A      old.c
D      new.c
svn copy old.c copy.c
A      copy.c
svn commit -m "Changed file structure"
   ...
```

Undo a Modification

- You made a mistake when altering `main.c`, and want to back out your changes, restoring the previous revision that you checked out.

```
svn revert main.c
Reverted 'main.c'
```

Update to Latest Version of a File

Files change in the repository whilst you are working; your working copy will become outdated. Periodically (and usually before you check in) you must update your working copy to the latest repository state-of-the-art.

If a file that you are modifying has been changed by someone else then Subversion, like CVS, will attempt to merge the changes automatically into your working copy. Usually this works fine. Occasionally a *conflict* occurs, when Subversion doesn't know how to perform a merge because the changes interfere with each other. This is dealt with in the next recipe.

- You want to see what files have changed in the repository since you last updated

```
svn status --show-updates
   ... or ...
svn status -u
M      *    main.c
       *    other.c
```

The * shows that an update must be taken from the repository. The `M` shows that you have modified a file locally: Subversion will merge the repository change into your modified copy.

- You want to bring in the latest version of `other.c`, but leave `main.c` in the current working copy

```
svn update other.c
U  other.c
Updated to revision 4.
```

Notes:

- Working copies are not dynamic, so you are in control of when you pull in other people's changes. This is especially useful during builds – no files will change during your build process, so you can be assured of the build's integrity.
- Your working copy can contain a random collection of file version numbers (you can `commit` and `update` files independently).
- You can use the `-r` argument to shift to a particular file revision.

Resolving Conflicts

- You performed a `svn` update, and a file had conflicts (the `update` output message showed a file with `C` status).
- You can't check your version of the file in until you have resolved the conflict.
- Open the conflicted file in your editor. You will see the conflicts flagged between *conflict markers* that highlight the problem changes.
- Subversion has created three extra files in your working copy directory:
  1. `filename.mine` – the copy of `filename` that was in your working copy directory before you ran `svn update`.
  2. `filename.rOLDREV` – The file that was the `BASE` revision before you ran `svn update`.
  3. `filename.rNEWREV` – The new version that came from the repository.
- Merge the changes manually.
- Then type the following command to tell Subversion that you have resolved the conflict (note: this is an extra step over CVS operation):

```
svn resolved other.c
Resolved conflicted state of 'other.c'
```

Inspect File History

- You want to see how the file `other.c` changed over time.

```
svn log other.c
------------------------------
r3 | pete | Tue, 16 Dec 2004 12:23:12 +0000
                                      | 1 line
Changed the default banana count
------------------------------
r2 | pete | Tue, 16 Dec 2004 12:16:43 +0000
                                      | 1 line
Added
------------------------------
```

Creating a Branch

You want to branch the `HEAD` of your project's `trunk` into a branch called `trout`, to perform some parallel development work .

```
svn copy \
  svn://REPOSITORY/widgetizer/trunk \
  svn://REPOSITORY/widgetizer/branches/trout \
  -m "Created trout branch"
Committed revision 128.
```

Note: we didn't need to have a working copy checked out to perform this operation, since we used URIs into the repository. (You can also do this locally in a working copy, but there's not much point.)

Working on a Branch

- Having created a branch, you want to check it out and start working on it.

```
svn checkout \
   svn://REPOSITORY/widgetizer/branches/trout
A  trout/main.c
A  trout/other.c
Checked out revision 128.
```

This new branch is a separate physical directory from the `trunk`. However, it retains all the file revision history (try a `svn log` on a file).

Alternatively, if you already have a working copy checked out, and you want to switch it to view the branch, use `svn switch`:

```
svn info | grep URL
URL: svn://REPOSITORY/widgetizer/trunk
svn switch
svn://REPOSITORY/widgetizer/branches/trout
U  main.c
U  other.c
Updated to revision 128.
```

```
svn info | grep URL
URL:
svn://REPOSITORY/widgetizer/branches/trout
```

This is more efficient than checking out a whole new tree, and also allows you to switch just a single directory, or even a single file.

You can now use your branch version of the working directory as if it was the `trunk`. All checkins you make appear only on the branch, without affecting `trunk` at all.

Merge Changes Between Branches

- Your work on the `trout` branch is sufficiently advanced that it should be merged back onto the mainline of development, into `trunk`.
- Create a working copy of the trunk, then identify the range of changes that you want, merge them into the trunk, and check in:

```
cd ~/Work/trout
svn update
At revision 132.
svn log --verbose --stop-on-copy
   The final revision number listed is the
   start of the branch, let's say it's 2
cd ~/Work/trunk
svn merge -r 2:132
svn://REPOSITORY/widgetizer/branches/trout
M   main.c
M   other.c
svn commit -m "Merged trout changes r2:r132 \
  into the trunk"
    ...
```

Like CVS, Subversion doesn't record a merge history, so you have to be careful when you merge a branch into `trunk` multiple times. Always specify revision numbers in great detail when you write the log message, next time you merge you will want to merge the range `133:HEAD`, not `2:HEAD`.

Create a Tag

- You release the first version of your software and want to tag the source code that built it.

```
svn copy \
  svn://REPOSITORY/widgetizer/trunk \
  svn://REPOSITORY/widgetizer/tags/Release1 \
  -m "Created release 1 tag"
Committed revision 326.
```

This is a very simple tag operation (but probably the most common). You can tag more eclectic collections of files. Update your working copy to the set of file versions that you want to tag (using `svn update -r`). Then:

```
svn copy \
  my-working-copy \
  svn://REPOSITORY/widgetizer/tags/TagName \
  -m "Created release 1 tag"
Committed revision 327.
```

Undo a Change in the Repository

- Some idiot (you, probably) checked a blatantly wrong revision into the repository.
- You want to revert the change.

```
svn merge -r 10:9
svn://REPOSITORY/widgetizer/trunk/main.c
U  main.c
svn commit -m "Undoing change committed in r10"
    ...
```

Create a Change Log

- You want to maintain a file containing all commit messages over the project's history.
- You don't want to do it by hand.
- At each release point in your project, do this:

```
svn log -r 42:HEAD >> ChangeLog
svn commit ChangeLog -m 'Update ChangeLog'
```

Shortened Commands

Subversion commands can be abbreviated to save you some typing. The following table shows the abbreviations for the common commands:

Long form	Short (alternate) form
checkout	co
checkin	ci
copy	cp
delete	rm, del, remove
help	?, h
list	ls
move	mv
status	stat, st
switch	sw
update	up
–revision	-r
–show-updates	-u

Table 1: Shortened forms of Subversion commands

CVS Conversion in 2 Minutes

Subversion has been designed to work as much like CVS as is practical. Most of your CVS knowledge translates directly into Subversion. The following table shows the Subversion equivalents of common CVS commands:

CVS command	Subversion command
cvs co	svn co
cvs ci	svn ci
cvs tag	svn copy
cvs –nq update	svn status
cvs diff	svn diff
cvs tag –j	svn copy
cvs add	svn add
cvs update	svn update
cvs merge	svn merge

Table 2: CVS and Subversion commands compared

Other Tools

There are other tools that sit on top of Subversion. You may find them useful:

TortoiseSVN – a Windows explorer plug-in providing a Subversion RMB submenu. This is an absolutely excellent Subversion front end, and I highly recommend it. It has an excellent history viewer, merge tool, repository browser, and much more. If you use Subversion under Windows you really want to get a hold of this. You can download it from `http://tortoisesvn.tigris.org/`

RapidSVN – a cross-platform GUI front end for Subversion. In my experience this is an odd beast, and best avoided – the Tortoise front end is far superior.

Ankh – a MS Visual Studio plug-in for the Subversion client. It works quite nicely, and I recommend it. Some Subversion operations are easier to perform through TortoiseSVN, so I tend to use both clients together. Ankh is not quite as polished as TortoiseSVN, but it is an excellent tool for VS developers.

You can download it from `http://ankhsvn.tigris.org/`

ViewCVS – a web based repository viewer, originally for CVS, but now with Subversion support (at the moment there is no CVSGraph-like facility, sadly).

Subversion's **svnadmin** and **svnlook** tools. Mere mortals needn't ever use these; they are administration tools.

There are many other useful Subversion tools – I have seen very pretty Mac OS X clients (**SvnX** is a neat GUI front end and **SCPlugin** is a helpful Finder plug-in). **Subclipse** and **Svn4Eclipse** are Eclipse plug-ins. Look around for other useful Subversion plug-ins.

Pete Goodliffe

Further Reading

Subversion's homepage is `http://www.subversion.org/`

The free Subversion book is an excellent, thorough overview of the system. It is available from `http://svnbook.org/` You can buy a dead tree version of this from O'Reilly, which is recommended. It's called *Version Control with Subversion*, ISBN: 0596004486.

Another recommended book is *Pragmatic Version Control Using Subversion* by The Pragmatic Programmers. ISBN: 0974514063.

There is a Latex Subversion quick reference card in the source distribution.

I Wish Programming Was Easy Again

Paul F. Johnson <editor@accu.org>

It takes my PC just over 90 seconds from power up to desktop. It's nothing special, not really that high spec'd, okay, it has a lot of memory on it, but lots of memory is nothing that peculiar anymore.

Now, add onto that time loading kdevelop and we're up to the two minute stage. Type some code, compile and test, say 5 minutes. The code is linked to, say, OpenGL.

It's not an amazing amount of time, but say the code has a problem. Is the problem in the code or the libraries it is linked to? Check the code, hmm, nothing looks wrong there. Right, run the debugger on the binary... ah, looks like I'm passing something incorrectly to OpenGL. What is supposed to be passed in there? Where is my book with the API in? Drat, it's out of date, but looking at that I'm actually correct. Oh well, fire up the browser and search the OpenGL documentation, ah, found the problem.

My code is fine, it's passing the correct information to OpenGL, so what is the problem? I'd better check the physical information being passed and that it is exitting the OpenGL function correctly. It is. ARGH!!!!!!!

Does that sound familiar? It does from this end, and definitely will when you think back to when you started to learn C++ (or C) and tried to link it to an established library for the first couple of times (or more – I still manage to mess up when using Qt).

Is there a reason for this or is programming just becoming to darned complex these days for people to break into? And if it is too complex, shouldn't we all be doing something about it?

Lots of Questions...

My original editorial in the last issue of C Vu sparked quite a lot of conversation on the `accu-general` mailing list where it gave those on there the chance to let off a bit of steam over the issue.

However, as with most computer related topics, it is not as simple as looking through rose-tinted glasses, a more pragmatic approach is required. While I stick to every word I said in the last editorial, you'll see that some things are made better by being slightly more complex.

One of the main issues raised was how complex something as simple as making the internal speaker play a scale had become. The example cited was for the Dragon32 which just needed `PLAY "ABCDEFG"` to be typed into the machine and a scale played. But was it really as simple as that for everything?

You'll need to think back now. If I take 4 machines (BBC B, Oric, Spectrum and Dragon) and compare how different something as simple as playing a scale was, we can soon see that even then something like sound wasn't that easy! I'll not consider the Commodore range as I have no real experience of them.

BBC B
```
FOR a%=89 TO 113 STEP 4
SOUND 1,-10,a%,100
NEXT
```

Oric 1
```
FOR a=1 TO 8
MUSIC 1,3,a,10
NEXT
```

Spectrum
```
FOR a=0 TO 11
BEEP a, 5
NEXT a
```

Dragon
```
PLAY "ABCDEFG"
```

These will all play a series of beeps which produce a scale (some include the sharps, some don't). What it demonstrates that from a programming perspective, if you learn from any of these audio systems, you're going to have a problem going to a different one.

Okay, sound is just one thing. A machine is nothing without input/output and graphic handling. This is where things became really different. While most had a `PLOT` and `DRAW` method of putting lines on the screen, the way they did it wasn't the same. The BBC B and Spectrum were effectively the opposite of each other with the Oric just being strange!

Line graphics are all well and good, but what about sprite handling? None of the 8 bit machines really had any sprite handling facilities, so unless you created a whole pile of user defined graphics, you would need to fudge the screen so that it appeared there was more graphics than there was or you had to generate the graphics on the fly or you plain had (say) 16 points of rotation and depended on a frame rate to fool the eye. (Later versions of BBC BASIC (mainly for the RISC OS machines) had sprite handling facilities, however, that would also mean I would need to consider the ST and Amiga machines).

Compare this to literally a few lines to call the Allegro or SDL libraries and you can see my point. Sure, there are problems learning how to compile, waiting for the build and the such, but it is (on the whole) much easier to learn.

What's Your Point?

Simple, yes it was easier but the drawback was that moving from one platform to another was a right royal pain in the backside! Interoperability was not on the cards.

While things are more complex now (such as having to use specialist bitmap software for sprites or sound software for audio), the actual API is much simpler; you learn it once and compile it quite a few times. Sprite, audio and (to some extent) error handling is provided by the library.

You What?

Okay, at this point, you're starting to question the reason behind the argument put forward in the last editorial as this really does make me sound as if I've contradicted myself.

While the movement has been from a version of BASIC on all machines, quite a number of machines do come with languages which don't require compilation. Tcl is one such language. No compilation is required and the results are instantaneous. It just isn't as easy to learn as BASIC was or (in my opinion) as friendly to the new user.

Copyright issues aside, there is no real reason why a version of BBC BASIC for each platform isn't bundled (I know there is BBC BASIC for Windows, a Linux BBC BASIC and a MacOSX version, all but the Windows version being free). This would give the user an easy to use language within (about) 2 minutes from firing up with the advantage of seeing instant results.

DHTML and Javascript – The Way Forward?

A point which was raised in discussion was the use of DHTML and Javascript with a web browser for learning. I have a problem with this, in fact, I have a few.

1. It requires a browser happy with both. Despite there being an ECMA standard for JavaScript, not all browsers comply (quelle surprise) with how the standard describes operations. Add this to having to fudge for specific browsers, and you can't really have anything which will work with certainty.
2. Some browsers don't do one or the other (or both). Opera (up to version 7.5.2 for Linux) certainly doesn't like DHTML. Browsers used for the visually impaired have problems with DHTML and dynamic content generated in JavaScript.
3. While JavaScript is pretty close to an OO language, it is also close to being a straight procedural language. As such, it really doesn't help the newcomer as they're easily confused – are they or are they not using an OO language?
4. DHTML doesn't have a standard. It makes any real certainty over how a browser should react somewhat hit and miss.

While everyone now using one of the main platforms[1] has a browser, someone learning on (say) Firefox will need to learn how to get around DHTML problems for Opera and IE (with the same being applicable in reverse). This is pretty much the same problem as there was with the differing implementations of BASIC.

What About the Learner Though?

We are all (presumably) professionals. We know how to build our software, effectively debug, optimise the code we write and the myriad of other aspects there are to writing software, but we must remember that we all had to start somewhere.

While there is little doubt that the likes of Francis Glassborow's book is a step in the right direction, it is, unfortunately, not cross platform. I certainly would not buy a copy as I very rarely touch a Windows machine (other than tech support at work that is!); it would contain very little which would give an instant result for either myself or my son.

Have we moved so far away from having a built in interpreter with some form of common language behind it that to go back would be almost unthinkable? I have to conclude, on current evidence, that this is the case.

The learner no longer has the 3 seconds and into BASIC as we used to have. The code, build, run, debug system looks here to stay unless a common language is built in to each Linux distro, each version of Windows and OSX. I can't see that happening, more's the pity.

Sorry folks, looks like the fun days of the 8 bit machines have gone and we're left with makefiles and debuggers.

Paul F Johnson

1 These being Windows, Linux and MacOSX. I am leaving other platforms (such as RISC OS) out for the purpose of this article.

Using Qt's Non-GUI Classes

Jasmin Blanchette <http://www.trolltech.com>

In the fifth installment of our series on cross-platform programming with the Qt 3 C++ toolkit, we are going to review Qt's non-GUI classes. Although Qt is fundamentally a GUI toolkit, it contains many non-GUI-related classes that are useful when writing portable applications. Here we'll focus on Qt's support for networking, database access, inter-process communication, XML handling and multithreading.

Network Classes

Qt provides a set of classes for writing cross-platform TCP/IP clients and servers. The most important classes are `QHttp`, `QFtp`, `QSocket`, `QSocketServer` and `QSocketDevice`.

`QHttp` and `QFtp` implement the client side of the HTTP and FTP protocols. HTTP (Hypertext Transfer Protocol) is an application-level network protocol used mainly for downloading HTML and XML files, but it is also used as a high-level transport protocol for other types of data. For example, HTTP is often used for transferring purchase orders over the Internet. In contrast, FTP (File Transfer Protocol) is a protocol used almost exclusively for browsing remote directories and transferring files. (see Figure 1.)

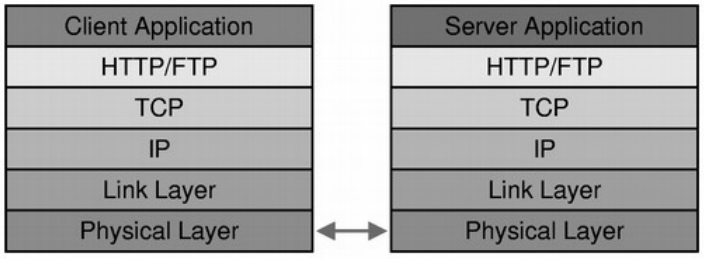


Figure 1: HTTP and FTP protocol stacks

Since the two protocols are used to solve similar problems, the `QHttp` and `QFtp` classes have many features in common:

Non-blocking behaviour. `QHttp` and `QFtp` are asynchronous. You can schedule a series of commands (also called "requests" for HTTP). The commands are executed later, when control goes back to Qt's event loop. (Blocking behaviour is usually unacceptable in a GUI application, because it can freeze the user interface for some time.)

Command IDs. Each command has a unique ID number that you can use to follow the execution of the command. For example, `QFtp` emits the `commandStarted()` and `commandFinished()` signal with the command ID for each command that is executed. `QHttp` has `requestStarted()` and `requestFinished()` signals that work the same way.

Data transfer progress indicators. `QHttp` and `QFtp` emit signals whenever data is transferred. You can connect these signals to a progress bar's `setProgress()` slot.

For example, let's assume we have a class called `Downloader` that has a `QHttp` data member called `http` and a `QFile` member called `outFile`. To keep the example simple, we'll do everything in the constructor and in a slot called `httpDone()`.

```
Downloader::Downloader() {
  outFile.setName("bookreviews.html");
  if(!outFile.open(IO_WriteOnly))
    return;
  connect(&http, SIGNAL(done(bool)),
          this, SLOT(httpDone(bool)));
  http.setHost("www.accu.org");
  http.get("/bookreviews/public/", &outFile);
  http.closeConnection();
}

void Downloader::httpDone(bool error) {
  if (error)
    ...
  outFile.close();
}
```

In the constructor, we open the output file. Then we connect `QHttp`'s `done()` signal to our `httpDone()` slot. Finally we schedule three HTTP requests: "set host", "get", and "close connection". These requests will be executed at some point in the future, when control returns to Qt. When all three requests have been processed, the `httpDone()` slot is called to close the file.

If you need TCP-based protocols other than HTTP and FTP (e.g., POP3, SMTP, NNTP or any proprietary protocol), you can use the `QSocket` class to implement your own protocol. TCP (Transmission Control Protocol) is a low-level network protocol used by most Internet protocols, including HTTP and FTP, for data transfer. It is a reliable, stream-oriented, connection-oriented transport protocol. It is especially well suited for continuous transmission of data.

An alternative to TCP is UDP (User Datagram Protocol). UDP is a lightweight, unreliable, datagram-oriented, connectionless protocol. It can be used when reliability isn't important. For example, a server that reports the time of day could choose UDP. If a datagram with the time of day is lost, the client can simply make another request. UDP is supported through the `QSocketDevice` class.

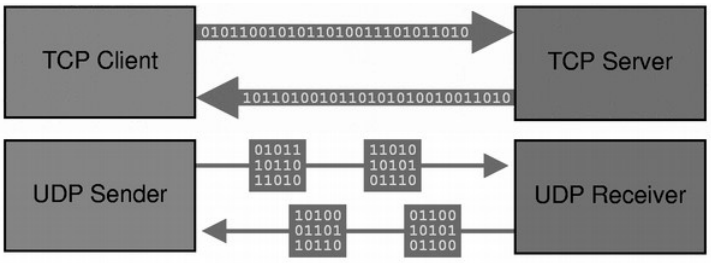


Figure 2: TCP and UDP protocols

Database Classes

The Qt database classes provide a multiplatform interface for accessing SQL databases. Qt includes native drivers for Oracle, Microsoft SQL Server, Sybase Adaptive Server, PostgreSQL, MySQL, ODBC and SQLite. The drivers work on all platforms supported by Qt and for which client libraries are available. Programs can access multiple databases using multiple drivers simultaneously.

Programmers can easily execute any SQL statements. Qt also provides a high-level C++ interface that programmers can use to generate the appropriate SQL statements automatically.

The `QSqlQuery` class is used to directly execute any SQL statement. It is also used to navigate the result sets produced by `SELECT` statements.

In the example below, a query is executed, and the result set navigated using `QSqlQuery::next()`:

```
QSqlQuery query("SELECT id, surname FROM staff");
while(query.next()) {
  cout << "id: " << query.value(0).toInt()
       << " surname: " << query.value(1).toString()
       << endl;
}
```

Field values are indexed in the order they appear in the `SELECT` statement. `QSqlQuery` also provides the `first()`, `prev()`, `last()` and `seek()` navigation functions.

`INSERT`, `UPDATE` and `DELETE` are equally simple. Below is an `UPDATE` example:

```
QSqlQuery query("UPDATE staff SET salary = salary"
                " * 1.10 WHERE id > 1155 AND id < 8155");
if(query.isActive()) {
  cout << "Pay rise given to "
       << query.numRowsAffected()
       << " staff" << endl;
}
```

Qt's SQL module also supports value binding and prepared queries, for example:

```
QSqlQuery query;
query.prepare("INSERT INTO staff (id, surname,
                                  salary)"
              " VALUES (:id, :surname, :salary)"
```

```
query.bindValue(":id", 8120);
query.bindValue(":surname", "Bean");
query.bindValue(":salary", 29960.5);
query.exec();
```

Value binding can be achieved using named binding and named placeholders (as above), or using positional binding with named or positional placeholders. Qt's binding syntax works with all supported databases, either using the underlying database support or by emulation.

For programmers who are not comfortable writing raw SQL, the `QSqlCursor` class provides a high-level interface for browsing and editing records in SQL tables or views without the need to write SQL statements. For example:

```
QSqlCursor cur("staff");
while(cur.next()) {
  cout << "id: " << cur.value("id").toInt()
       << " surname: "
       << cur.value("surname").toString() << endl;
}
```

`QSqlCursor` also supports the ordering and filtering that are achieved using the `ORDER BY` and `WHERE` clauses in SQL statements.

Database drivers usually supply data as strings, regardless of the actual datatype. Qt handles such data seamlessly using the `QVariant` class. Database drivers can be asked about the features they support, including query-size reporting and transactions. The `transaction()`, `commit()` and `rollback()` functions can be used if the database supports transactions.

Qt also provides classes that make it easy to present data from the database to the user. One of them is `QDataTable`, a table widget that displays records from a result set (see Figure 3). `QDataTable` supports in-place editing. Records can be updated and deleted without writing any code. Insertions require some code since most database designs expect new records to be created with a unique key.

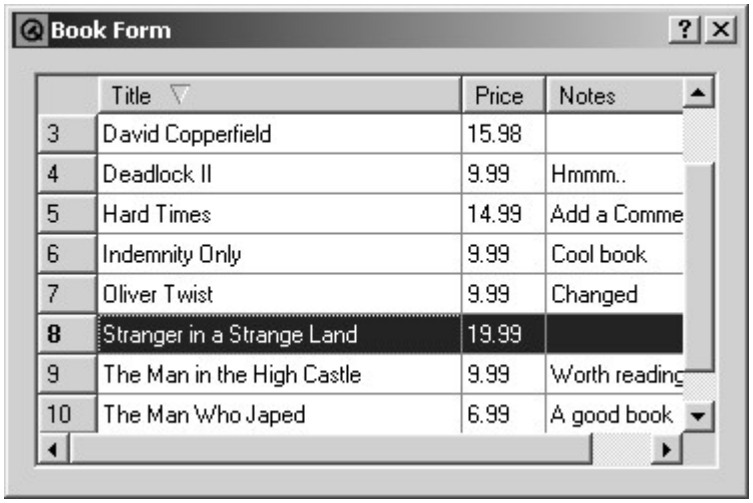


Figure 3: `QDataTable`

Qt also includes `QDataBrowser` and `QDataView` to display records as forms (see Figure 4), typically with one or perhaps a few records shown at a time. These classes provide buttons with ready-made connections for navigating through the records. `QDataView` is used for read-only data. `QDataBrowser` is used for editing, and can provide ready-made insert, update and delete buttons.

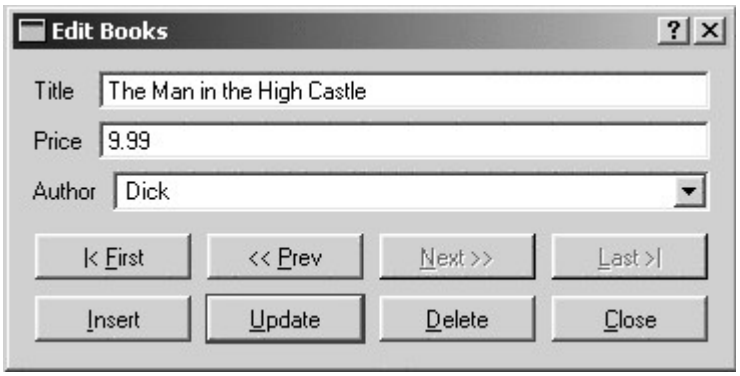


Figure 4: `QDataBrowser`

Inter-Process Communication

The `QProcess` class is used to start external programs, and to communicate with them from a Qt application in a platform-independent way. Communication is achieved by writing to the external program's standard input stream and by reading its standard output and standard error.

`QProcess` works asynchronously, reporting the availability of data by emitting Qt signals. We can connect to the signals to retrieve and process the data, and optionally respond by sending data back to the external program.

XML Classes

Qt's XML module provides a SAX parser and a DOM parser, both of which read well-formed XML and are non-validating. The SAX (Simple API for XML) implementation follows the design of the SAX2 Java implementation, with adapted naming conventions. The DOM (Document Object Model) Level 2 implementation follows the W3C recommendation and includes namespace support.

Many Qt applications use XML format to store their persistent data. The SAX parser is used for reading data incrementally and is especially suitable both for simple parsing requirements and for very large files. The DOM parser reads the entire file into a tree structure in memory that can be traversed at will.

Multithreaded Programming

GUI applications often use multiple threads: one thread to keep the user interface responsive, and one or many other threads to perform time-consuming activities such as reading large files and performing complex calculations. Qt can be configured to support multithreading, and provides four central threading classes: `QThread`, `QMutex`, `QSemaphore` and `QWaitCondition`.

To create a thread, subclass `QThread` and reimplement its `run()` function. For example:

```
class MyThread : public QThread {
protected:
  void run();
};

void MyThread::run() {
  ... // thread code goes here
}
```

Then, create an instance of the thread object and call `QThread::start()`. The code that appears in the `run()` reimplementation will then be executed in a separate thread.

The `QMutex` class provides a means of protecting a variable or a piece of code so that only one thread can access it at a time, preventing memory corruption, crashes or other race conditions. The class provides a `lock()` function that locks the mutex. If the mutex is unlocked, the current thread seizes it immediately and locks it; otherwise, the current thread is blocked until the thread that holds the mutex unlocks it. Either way, when the call to `lock()` returns, the current thread holds the mutex until it calls `unlock()`. For example:

```
QString globalString;
QMutex mutex;
...
mutex.lock();
globalString += "More text\n";
mutex.unlock();
```

`QSemaphore` and `QWaitCondition` are two other classes that make it possible to synchronize threads.

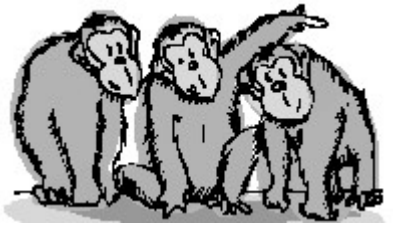
This article presented a short overview of some of the most interesting non-GUI classes offered by Qt. There are many more classes than we presented here. For example, Qt has a Unicode 16-bit `QString` data type, classes for performing text and binary I/O, classes for supporting internationalisation, and more.

Jasmin Blanchette

Professionalism in Programming #31

Code Monkeys (Part Two)

Pete Goodliffe <pete@cthree.org>



Darwinian Man, though well-behaved, at best is only a monkey shaved
Gilbert and Sullivan

Last time we began a high-paced stroll through a gallery of collected programmer stereotypes. We're looking at individual developer attitudes, to see how they can drastically affect the quality of the software we write and how they affect the whole development team.

In this concluding article we'll finish the tour, and see what makes the best type of programmer. Brace yourself: here come more Code Monkeys...

6. The Cowboy

Some would incorrectly classify this guy a *Hacker*. He's not a hacker in the classic sense of the word. 'Hacker' is a term used by geeks to proudly describe a heroic coder[1]. The Cowboy is a shoddy programmer, who actively avoids hard work. He'll take as many shortcuts as he can find.

The Cowboy dives straight into code and does the minimum work to solve the immediate problem. He won't care if it's not a very good solution, if it compromises the code structure, or will not satisfy future requirements.

A Cowboy is anxious to complete each task and move on to the next. If he's read a little about processes, he'll call this Agile Programming. It's really just laziness.

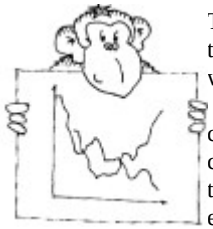
Strengths Cowboy code *works*, but isn't particularly elegant. Cowboys like to learn new things, but seldom get around to it (it's too much like hard work).

Weaknesses You'll spend ages cleaning up after a Cowboy. Their aftermath is not a pleasant place to be. Cowboy code always requires later repair, rework and refactoring. They have a limited palette of techniques to use, and no real engineering skills.

What to do if you are one Learn to *hack* code in the right sense of the word. Take a pride in your work, and spend more time over it. Admit your failings, and try to improve.

How to work with them Never go into a Cowboy's house; if their code's anything to go by, it'll be a DIY disaster! Understand that they're not a malicious breed, just a little lazy. Organise reviews of their code. Get them pair programming (they might work well with an Eager Coder, but if you want to see fur flying pair them with a Planner).

7. The Planner



The Planner thinks about what he's doing so much, the project's been canned long before he's started writing any code.

It's true, you *must* plan up front and establish a cohesive design, but this guy forms an impenetrable cocoon around himself and refuses any contact with the outside world until he's finished. Meanwhile everything's changing around him.

Terminally educated, the Planner studies and reads a lot. A common subspecies is the Process Weenie; he knows all about the 'proper development process', but is weak on hitting deadlines or getting anything done. (Process Weenies eventually become middle managers, and then get fired.)

Strengths They *do* design. They do think. They don't hack out thoughtless code.

Weaknesses When a Planner sets to work there is a very real danger of *over design*. He tends to create very complex systems. Planners are the key cause of *analysis paralysis* – where development gets more focused on methods and modelling than on prototyping and creating a solution. The Planner likes to generate endless documents and call meetings every other hour.

He spends ages thinking, and not enough time doing anything. He knows a lot, but it doesn't all make the leap from theory to practice.

What to do if you are one It is important to create careful designs up front, but consider incremental development and prototyping as methods to verify your design. Sometimes you can't commit to a design until you've actually started to implement it. Only then will you appreciate all the problems. Try to establish a better balance of planning and action. Console yourself that it's better to spend too long designing than to write awful code – the latter is far harder to fix.

How to work with them Ahead of time, agree all milestones and deadlines for a Planner's work. Throw in a *design complete* milestone; they'll be happy that it has been recognised as an important task, and encouraged to complete their design work on time. This is usually enough to crystallise a Planner into action.

Avoid meetings with a Planner. You'll spend an hour arguing about how to decide the agenda.

8. The Old Timer

This old boy is a senior programmer from the old school. Sit back and listen to him reminisce about the Good Old Days, when he used punch cards and machines without enough memory to hold the result of an integer addition.

The Old Timer's either happy that he's still doing what he loves the most, or bitter that he's missed promotion countless times. He's seen it all, knows all the answers, and won't learn new tricks (he'll tell you that there's nothing new to learn, we just repackage the same old ideas).

An Old Timer doesn't suffer fools gladly. He's a bit cranky, and is easily irritated.

Strengths He's been programming for years, and so has considerable experience and wisdom. The Old Timer has a mature approach to coding. He has learnt what qualities make good and bad programs, and how to avoid the common pitfalls.

Weaknesses The Old Timer won't willingly learn new techniques. Fed up with fashionable ideas that promise much and deliver little, he's slower and more resilient to change.

He has little patience thanks to years of corporate ineptitude. He's been at the receiving end of countless tight deadlines and unreasonable managers.

What to do if you are one Don't be too judgmental of younger more enthusiastic programmers. You were once like them, and *your* code wasn't awful.

How to work with them *You don't know how easy you have it, you young programmers.* Don't mess with an Old Timer, or you'll find out how he survived this long in the software factory. Choose your battles with him wisely. Show him respect, but treat him as a peer, not a deity.

Understand the Old Timer's motivation. Check if he's programming because he loves to do so, or because he can't scale the promotional ladder any higher.

9. The Zealot

The Zealot is a brainwashed convert, a disciple who blindly thinks that everything *BigCo* produces is excellent. Teenage girls have rock stars to worship; programmers have their own idols. In his enthusiasm, the Zealot takes it upon himself to become an unpaid technology evangelist. He'll try to incorporate BigCo products into every assignment he is given.

The Zealot follows BigCo to the exclusion of all other approaches, and rarely knows about alternatives. Anything that's not excellent in the current BigCo product line will be fixed in the next version, which we *must* upgrade to immediately[2].

Strengths He knows BigCo's products inside out, and will produce genuinely good designs based on them. He is productive with that

1 Also been subverted by ignorant people, and used mistakenly to mean *cracker* – someone who breaks into computer systems without permission.

2 Don't assume that Zealots only idolise certain software vendors. A Zealot might equally be an open source advocate, or hanker after an obsolete software package.

technology, but not necessarily maximally productive – other unfamiliar approaches might be more effective.

Weaknesses Being a Zealot, he's neither objective nor pragmatic. There may be better non-BigCo designs that he will miss. Worse, though, are the Zealot's continual rants about BigCo.

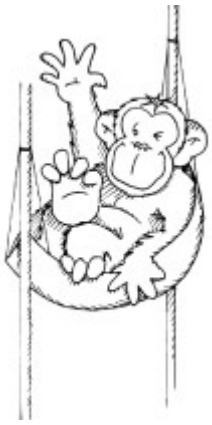
What to do if you are one No one expects you to turn away from your beloved BigCo. It's valuable to understand their technologies and know how to deploy them. But don't be a technology bigot. Embrace different approaches and new ways of thinking. Don't look at them with an air of superiority, or prejudge them.

How to work with them Don't bother getting into philosophical arguments with a Zealot. Don't try to explain the virtues of your preferred technology – he won't listen. Watch yourself – one conversation with this guy can turn *you* into a Zealot. He's contagious.

Zealots are generally harmless (and amusing to watch from a distance), unless your project is at a critical design stage. At this point, provide a clear, unbiased perspective on the problem domain and insist on a thorough evaluation of all implementation approaches. Remember: he might be right.

If you encounter silly arguments, counter them with well prepared, accurate, detailed information about the strengths of your approach and the weaknesses of his.

10. The Slacker



The Slacker is a workshy sluggard. He's hard to detect, because he's learnt to make it look like he's overloaded with jobs. His 'design' is playing solitaire, his 'research' is looking at fast cars on the web, and his 'implementation' is working on his own stuff. The Slacker actively avoids all assignments ("Oh, I'm *far* too busy to do *that*").

A more subtle Slacker will only work on the things he wants to, or the bits he thinks should be done, not what he's supposed to. Despite working all the time, he'll never get his jobs done.

The Slacker knows how to have fun. He parties too much, and can usually be found sleeping under his desk. His diet consists mostly of coffee, except for lunchtimes when you'll find him in the bar.

This guy can be a burn-out; one too many failed projects has killed his desire to work.

Strengths At least he knows how to have fun.

Weaknesses A Slacker is an obvious liability. It's hard to prove he's slacking – some hard problems *do* take a while to sort out. A programmer might not be slack, just not skilled enough to solve the problem quickly.

What to do if you are one Work on your morals, and start to put some effort in. Or learn to live with the guilt.

How to work with them It's best not to bitch about a Slacker – you have your own flaws. In good time he'll get his come-uppance.

Take measures to prove that you are working effectively, and that delays are the Slacker's fault. It might help to keep a methodical diary of your work. A clear set of deadlines is generally enough to get a Slacker working. Don't start writing his stuff too, even in desperation. He'll only expect you to do this next time.

Avoid burnout yourself, try to have fun as you work. Perhaps you should hit the bar with him one lunchtime.

11. The Reluctant Team Leader

This is the organisational classic; a developer who's been promoted to team leader when there was no further technical route for him to advance.

You can plainly see that he is uncomfortable in this role. He doesn't have the correct skillset, and struggles to keep up. He is a programmer, and wants to program. This guy is not a natural organiser or manager of people, and is a bad communicator.

Most programmers make spectacularly bad leaders. There are few genuinely excellent software team leaders; it requires a particular skillset that is both technical and organisational.

The Reluctant Team Leader is usually quite mild mannered and indecisive – how else did he get persuaded to take on this job? He gets squashed between the development team and management, taking the blame for slippage and poor software. An increasingly harrassed expression grows on his face until he finally burns out.

Strengths The Reluctant Team Leader has a real sympathy for the programmer's plight – he's been there, and now wishes he was back. Often he is far too willing to take responsibility for late software delivery, to prevent the programmers being picked on by management. Just as he's not good at delegating work, he's not good at apportioning blame.

Weaknesses When a Team Leader tries to write code it will be awful. He never has enough time to write, design, and test carefully. He naively plans himself a full day's coding alongside team leading duties. He can't fit it all in, and so the Reluctant Team Leader spends longer and longer in the office, trying to keep up.

He can't organise well, can't explain things to managers, and can't manage his team members properly.

What to do if you are one Get training. Quickly.

If you're not happy in this role, push for a career move. This is not admitting defeat; it's pointless to burn yourself out doing something you hate and aren't good at. Not everybody has the skills or passion for management. Move to the area you do have skill and passion for.

If you like herculean tasks, try to sort out the promotion path at your company. Get them to recognise that a managerial position should *not* be the next step up from senior developer. Few programmers make decent managers; their brains aren't wired up the right way.

How to work with them Be sympathetic, and do everything you can to help the Team Leader. Give him reports on time, and try to get your work done to schedule. If you might miss a deadline, let the Team Leader know early on, so he can plan around it.

The Ideal Programmer

From this tangled mess, it's clear to see that we're a strange breed. Which of these code monkeys should we aspire to? What code monkey cocktail will create the *Ideal Programmer*?

Unfortunately, in the Real World there are no perfect programmers – the beast is an urban myth. Although this question is somewhat academic, finding an answer will give us something to aim for.

The fabled Ideal Programmer is part:

Politician

They must be diplomatic, able to deal with all of these weird code monkeys and the many, many more creatures that inhabit the software factory – managers, testers, support, customers, users, and so on.

Relational

They work well with others. They aren't territorial about their code and aren't afraid to get their hands dirty if a task is for the common good. They communicate well – they can listen as well as talk.

Artistic

They can design elegant solutions, and appreciate the aesthetic aspects of a high quality implementation.

[concluded at foot of next page]

Getting Personal

This classification of programmer attitudes isn't particularly scientific. Psychologists have devised more formal personality classifications; authoritative ways of calling you a freak. They don't focus exclusively on the software development world, but do give a valuable insight into programmer behaviour.

The *Myers Briggs Type Indicator* is perhaps the most popular tool. It decomposes your personality across four axes: extrovert (E) or introvert (I), sensing (S) or intuitive (N), thinking (T) or feeling (F), and judging (J) or perceptive (P). This classification results in a four letter descriptor; ISTJ would be common for a Code Monkey.

Belbin's *Team Roles* are a taxonomy of attitudes, investigating how an individual works as a part of their team. Belbin identifies nine specific behavioural roles: three action-oriented, three people-oriented, and three cerebral personalities. Understanding these enables us to build effective teams from people with complimentary skills; if every programmer was a *coordinator* then nothing would ever get done.

Neither of these personality taxonomies have a one-to-one mapping with my programmer classifications. And they have a distinct lack of primates.

Automatically Generating Word Documents

Silas Brown <ssb22@cam.ac.uk>

I like to write my documents without WYSIWYG (what you see is what you get). One reason for this is that I want my mind to concentrate on what the document actually says, rather than on trivial details about what it looks like.

There are good typesetting systems, such as LaTeX, that can make the text look nice by putting the line breaks in the best places, putting the right amount of space between the words, and so on, so why should I let such things distract me when I'm trying to write creatively?

Another reason for not using WYSIWYG is that a document becomes more like a program. If you are editing markup, you are editing the "source code" which can then be "run" in various ways. You might transform your markup (using various tools) into different formats, such as an HTML version for online viewing and a TeX version for printing.

If you later find something wrong then you can change the source code and all versions will be changed automatically when you run your script to re-generate them. You can also use the features of your favourite editor, not to mention obtaining the assistance of your filing system by writing the different sections in files and directories and getting the script to merge them when generating the output (and the order in which they are included can be changed quickly).

Your script might also include some code to write certain parts of the document automatically, for example, test results can be generated on-the-fly by running the test program (and any changes to that program will automatically be reflected in the document). In short, you can relax without having to worry about keeping everything consistent and up-to-date, because the computer does all that for you (well, most of it anyway).

But there's one problem with this approach. What if you're writing a technical paper for publication, and the editor says, "you must submit it in Microsoft Word format"? There are many ways of converting other formats into Word, but usually the editor will go further and say "and it must look like this example" or "it must use that template". This means that, after you've got the document into Word format (by converting from HTML or whatever), you usually have to do some further work inside Word to make it look like what the editor wants. And then if you later need to make more changes, you either have to make all future changes inside Word (which means sacrificing all the benefits of the source-code approach, not to mention being more restricted in your choice of operating system and working environment, and causing problems if you want to publish in other forms and Word's conversion is not nice enough), or you have to repeat the whole process of getting it into Word and making the adjustments all over again as many times as is needed. If any of this annoys you, then that may negatively affect the quality of your work, so it's worth doing something about it.

One option is to become an expert in Microsoft Word's macro system and use that, either to automate the process of getting your document into Word, or, if you're a real macro expert, use it to drive the whole approach instead of using the scripting language of your choice. However, this does have its disadvantages. You have to be very good with the macro system in order to make your macros robust (it's too easy to make some small change that accidentally means the macro won't work any more), and if your only reason for using Word is to fulfill the requirements of some editor, why bother to learn Microsoft's product-specific language if there's some way of doing the same job using Python or whatever other general scripting language you're already skilled at?

The approach that I eventually found was this. Modern versions of Microsoft Word have an interesting way of saving documents as HTML. Enough information is encoded in the HTML for Word to re-read the document exactly in nearly all cases (if there is something that cannot be saved in the HTML then Word should tell you what that is at the time of the save). Now, there are various views about the quality of Word's HTML in terms of Web standards and portability, but if you regard Word's version of HTML as an alternative file format for Word documents, there is hope for your automatic program. Simply save the editor's example as HTML, have your program generate HTML like that example, and import this into Word whenever you need to produce a Word version; no post-editing in Word should be necessary. You may even be able to use `diff` and other utilities to find how the Word document has been changed by others (if they haven't tracked the changes) by converting it back to HTML and comparing that with what you sent.

I wasn't able to test enough versions of Word to work out exactly when this feature was introduced; the HTML functionality of Word 97 does NOT preserve all formatting, but Word 2002 does. If you do have access to a real copy of Microsoft Word, as opposed to something like OpenOffice.org, then it's best to use the real Microsoft product in order to maximise your chances that the conversion will go without a hitch, especially when converting back to Word format. If you don't have access to Word then you might be able to get away with asking the person who wants the Word

---

[continued from previous page]

Technical genius

They write solid, industrial strength code. They have a broad palette of technical skills, and understand how and when to apply them.

Reading that list again, it's quite clear what we should be. If you haven't realised yet, I'll spell it out for you. The ideal programmer is a: **PRAT**.

So What?

Only the wisest and stupidest of men never change.

Confucius

Whilst it's entertaining to stare in the cages at all of these code monkeys and have a laugh at their expense, what should you do about this? If you do nothing then it's been little more than mere entertainment; you'll walk away doing exactly the same stupid things you've always done.

To improve as a programmer you must change. Change is hard – it runs contrary to our nature. The saying goes: *a leopard doesn't change his spots.*

ACCU Conference 2005

This series has been a snapshot of my presentation at last year's ACCU conference. If you fancy staring at more monkeys then come to the conference in April this year. I'm taking code monkeys to the next level: I'll be discussing writing software *in teams*. You'll see how to survive software writing teams, and learn techniques to improve your day-to-day software writing experience.

Come to *Life in the Software Factory.* It'll be a blast.

If he did, he wouldn't be a leopard any more. Perhaps that's the key. More of us should be wildebeest or rhinoceros.

Take a moment to think about the following questions:

- What kind of code monkey are you most like? If you're honest there's probably a little of each of them in you. Identify the one or two that describe you in a nutshell.
- What are your particular strengths and weaknesses?

Look over your code monkey description again, and see what practical things you could change. What specific techniques will help you to overcome bad attitudes, and how can you capitalise on your good ones?

Conclusion

Programmers are a social species (which is odd considering their lack of social skills). They are social by necessity; you can't create excellent large software systems without a closely working team of programmers, who are knit into a larger social structure (be it a department, company, or an open source culture).

Each of these programmers has their own foibles and peculiarities. Their underlying attitudes affect how well they program – both in their approach to the code and to their place in the team.

If you want to be an exceptional programmer, you need to foster the correct positive attitudes. Remember: aim to be a *prat*.

Pete Goodliffe

Acknowledgement

Thanks to my friend and colleague David Brookes for the excellent monkey illustrations. I owe you another pint, Dave (or perhaps another banana!)

document to save their example as HTML and import your HTML reply, although this requires more skill on their part especially if you have separate image files.

Reducing the Clutter

Word's HTML may be verbose, but it's not difficult to understand which part does what (especially with syntax highlighting), and it's much easier to work with Word's HTML than it is to work with its RTF (how many text editors with special facilities for navigating around RTF source do you know?) Modern versions of Word store the document's styles in the HTML header, using the CLASS attribute to assign HTML elements to styles and using stylesheet overrides to show other formatting.

The first thing to realise is that the only markup that matters for our purposes is the body section of the document. Everything up to and including the line beginning `<body` can be included without change at the start of the output document. You can do this in Python as follows:

```
template = open(template_html_filename).read()
text = open(my_html_filename).read()
out = open(output_filename, "w")
out.write(template[:re.match(r"<body[^>]*>",
                             template).end()])
out.write(text[re.match(r"<body[^>]*>",text).end():])
```

which means you can edit (or automatically generate) your HTML without worrying about what comes before the `<body>` tag. You could adapt this so as to paste in text up to any point in the document you like by inserting special keywords to indicate that point in both the template and your document; this might be useful if the editor requires a complex header (name, address, etc, in an unusual format) that is hard to convert but that will rarely (if ever) have to change during the editing process.

The remaining "clutter" in the template may include backward-compatibility markup (you can safely replace the regular expression `<!\[if !support.*!\[endif\]>` with nothing), and markup for smart tags and spelling and grammar alerts (you can also safely remove this, but it's easier if you can tell Word not to save it in the first place by unchecking "embed smart tags" and "embed linguistic data" in the "save options" dialogue and turning off spelling and grammar check in the "language" dialogue).

The template you have to work with is still a little too cluttered though. There is usually a gratuitous amount of markup to represent the skips between paragraphs, and in some documents you will find rather a lot of markup that overrides the language of each part of the text. If you use search-and-replace to simplify this, you can then edit (or automatically generate) the simplified version and then reverse all your searches and replaces to get back to Word's version. That can all be scripted.

It may help if you are working with no end-of-line markers but with end-of-paragraph markers. You can get the text into that state by using code such as:

```
paragraph_token = "—- PARAGRAPH"+chr(0)
template = template.replace("\n\n", paragraph_token) \
.replace("\n"," ") .replace(paragraph_token, "\n\n")
```

When you are writing your search-and-replace list, one thing that helps to begin with is to see a list of the most common markup. You can do this as follows. First, get a list of all the tags that open elements:

```
openingElems = re.findall("<[^/][^>]*>", template)
```

then generate a frequency count and print it out in order:

```
freqs = {}
for e in openingElems:
  if freqs.has_key(e): freqs[e] += 1
  else: freqs[e] = 1
freqs = map(lambda (x,y): (y,x), freqs.items())
freqs.sort()
for f in freqs: print f
```

That, together with a look at the template document, should give you enough pointers to write a quick script that converts between your simplified HTML markup (or the markup that is generated by the something-else-to-HTML-translator utility you are using in your scripts) and the markup of the template. This script can be re-used as many times as is needed during the editing process. It's best if you put the list of things to search and replace into a list of tuples, rather than in the code itself; that way the program can be reversed just by reversing each tuple in the list:

```
searchList = [
  (r"<p class=Text lots-of-attributes>\(.*\)</p>",
      r"<p>\1</p>"),
  (r"<p class=Bulletedlist
                    lots-of-attributes>\(.*\)</p>",
      r"<ul><li>\1</li></ul>"),
  ...
]

for s in searchList: text=re.replace(text,s[0],s[1])
# or:
for s in searchList: text=re.replace(text,s[1],s[0])
```

Note that we are not using special XML-handling toolkits because it's often quite awkward to get them to work with Word-generated markup; they tend to find errors and throw exceptions, which is normally good but not what we want here. Note also the slight awkwardness in the above example about bulleted lists: it has to assume that each item is its own list. You can do better by writing more code, but you might not want to (it's only for one article after all); you can preprocess your HTML by stripping `<UL>`s and replacing all `<LI>`s with `<UL><LI>` (unless you're also using numbered lists, in which case you need to be more clever).

It would be interesting to see if any readers can further automate the process of creating this script. It would be nice if a program could just look at the template and guess most of the rules automatically.

Finishing Touches

The markup for Word's equations and other special objects is complex and is best left alone; if you need to include such things in your document then you can do so by typesetting them in LaTeX or whatever and including the images (hopefully not too many Word-using editors will want to adjust your equations). Incidentally, Word defaults to setting images at 96 dots per inch, and it's best to use PNG or JPEG formats.

Footnotes can be awkward too, but it's best to write without using too many footnotes anyway.

Clearly there are going to be some quality compromises in using Word as an output format. It simply isn't as good at typesetting as LaTeX is. Hopefully the journal's production editor will use something else for the final copy. However, one thing you can do is to make sure your quotes and em-dashes look nice. You can do this by writing them as Unicode entities. Don't try to be too clever and put in ligatures, unless you understand the rules of when and when not to ligature (or can extract them from TeX), and even then don't do it as you may find the final printed copy has them replaced with strange-looking characters. However, nice quotes and em-dashes are quite simple to achieve.

The LaTeX to HTML translator TTH loses the em-dashes during its translation (it replaces them with hyphens), so you need to replace them with a special token before running TTH and restore them afterwards. The following Unix commands will do this and also deal with the quotes and ellipses:

```
mv texfile.bbl .bbl; mv texfile.aux .aux; cat     \
  texfile.tex | sed -e 's/—-/SSB22ANEMDASH/g' -e  \
  's/—/SSB22ANENDASH/g' | tth | sed -e            \
  's/SSB22ANEMDASH/\&#8213;/g' -e                 \
  's/SSB22ANENDASH/\&#8212;/g' -e                 \
  's/''/\&#8220;/g' -e                            \
  "s/''/\\&#8221;/g" -e 's/'/\&#8216;/g' -e       \
  "s/'/\\&#8217;/g" -e 's/\.\.\./\&#8230;/g'      \
  > texfile.html
```

[concluded at foot of next page]

Forgetting the ABC

Orjan Westin <owestin@unilog.co.uk>

I recently bought and read "C++ Coding Standards" by Sutter and Alexandrescu[1], without expecting many surprises. Like most seasoned C++ professionals, I've read Sutter's "Exceptional" and Meyers' "Effective" series, as well as numerous articles, which coupled with long experience of using the language has made me reasonably confident that I don't commit too many sins.

Indeed, I wasn't surprised by anything I read, although I might not have agreed completely with everything either. Those are minor quibbles, though, and not really worth mentioning. What did surprise me was an omission. Of course, any collection of best practice guidelines has to tread a line between the terse and verbose, and on the whole I think the authors have made a good job. I just wish they hadn't missed what I have always thought of as the ABC principle – Always Be Conventional.

To be fair, it is covered, albeit in a couple of specific instances. Items 26 (*Preserve natural semantics for overloaded operators*), 27 and 28 (*Prefer the canonical form*[s of +, +=,++, and the minus equivalents]) and 55 (*Prefer the canonical form of assignment*) are all expressions of ABC.

However, while it is important to follow the principle when dealing with operators, I was surprised that there was no generalisation of it, since it is a common problem anywhere an established convention exists.

Here is a simple example I once found in production code:

```
template<typename A_, typename B_,
         typename C_>
struct trio {
  A_ first;
  B_ second;
  C_ extra;
  ...
};
```

This is clearly inspired by `std::pair` and while I have omitted all but the essential members, it should illustrate the point. The struct is well named, implying where the idea came from, and it can be used as a `pair` would, with a `first` and `second` member. And then there's an `extra` member... I can see the logic there, but it's such a silly thing to do. Why isn't the `C_` member called `third`?

Okay, so that's just annoying but not dangerous per se. Here is another example, again found in production code, which really boggled me:

```
template<typename A_>
class matrix {
  vector<A_> data_;
  long width_, height_;
public:
  // STL style functions mirroring the
  // std::vector interface
  ...
  void resize(long w, long h) {
    data_.resize(w*h);
    width_ = w;
    height_ = h;
  }
  ...
};
```

This implementation of `resize` does not fulfil my expectations. While the behaviour of the function does what it says on the tin, it doesn't do enough. By designing the class interface to resemble that of a standard container, the programmer has implicitly promised to follow the convention already established by the STL. If I have data in the matrix and resize it, I would expect that the elements in the cells that remain in the matrix are unchanged, which is the canonical behaviour. This is not the case here, since there is no shuffling around of existing data if I change the width.

As a user of the matrix class, I will carry with me an assumption of what the function `resize` will do in a container, based on my knowledge of an established convention. Any piece of code that draws on such a convention should adhere to it completely, or clearly document where it does not. In this example, the offending function should really be called `resize_dirty`, or be rewritten to conform to the expected behaviour.

These are just two examples – I have seen many, many more. Neither is it only applicable to operators and STL look-alikes. If I were to create a new label control using MFC[2] to show the text in rainbow colours, for instance, I would make sure that the function to provide it with a text to display would be called `SetWindowText` and have the same form as the canonical one. Sadly, judging from what I've seen over the years, the common practice would be to call it `SetText` or `SetRainbowLabelText`.

ABC is basic wisdom and common sense, just like KISS, and it bears repeating. While Sutter and Alexandrescu devote four out of one hundred and one items to special cases, they fail to point out the underlying general principle. I do not know whether this was through oversight or because they considered it too obvious, but personally, I feel it is an important principle that is neglected too often.

It's just a matter of following conventions where they exist, fulfilling expectations and avoiding putting in surprises for other developers (or oneself, in six months time). Sometimes, if all your friends are jumping off a cliff, you really should do so too.

Orjan Westin

References

[1] Herb Sutter and Andrei Alexandrescu, *C++ Coding Standards*, Addison-Wesley, 2004

[2] *Microsoft Foundation Classes*, the class framework created by Microsoft to create GUI applications for Windows, in which window classes have a `SetWindowText` member.

[continued from previous page]

Previewing the Result

Finally, when you are developing your document, it may be nice to be able to preview what it will look like in Word, just as it is occasionally useful (or at least satisfying) to preview the PostScript or DVI output when working with LaTeX. This is particularly important if you are aiming for a certain page count. If you are working on Windows then you can tell your script to run Word with the HTML file on the command line, or if you don't have Word then you can download Microsoft's Word Viewer (search `www.microsoft.com` for Word Viewer) to see what should be exactly how Word will show your document.

Sadly, the Windows emulator WINE (`www.winehq.org`) doesn't seem to be up to running Word Viewer 2003 yet (it can run Word Viewer 97, but that will not be able to render this HTML properly). You can unpack Word Viewer 2003 with the `cabextract` utility if WINE fails to run the installer (but make sure you have the latest version of `cabextract` as old versions tend to corrupt the files) but when you run it, it is liable to complain about calls to unimplemented functions in DLLs. You may be able to work around this by borrowing DLLs from a recent version of Windows, but I don't have a suitable Windows license to try this.

It is possible that the special commercial distribution of Wine called CrossOver Office from `www.codeweavers.com` will do better. However, at the time of writing, their trial version download form seems to have been non-functional for some time, so I couldn't check this. Another trialware product is TextMaker from `www.softmaker.com/english/tm_en.htm` which is more lightweight than OpenOffice.org (useful if you're short on RAM) and often gives a better idea of how Word will display your document, but it is not perfect (it managed to scramble the references in one of my papers) so I wouldn't buy it without checking alternatives. OpenOffice.org 2.0 is due to be released soon, so that might be worth checking too.

Silas S Brown

Introduction to Tcl/Tk

R.D. Findlay <bob@icanprogram.com>

There are lots of different ways to program a computer.

Computers are essentially very stupid.

Even the fastest desktop computers can only understand a hundred or so commands in their microprocessor "brains". These commands in and of themselves are very simple things like "add these two numbers", "compare these two values" or "fetch this value from memory". How then could computers possibly appear to be so "smart"?

The simple answer is that they are able to do lots of simple things extremely quickly. So quickly that the typical desktop computer can easily do several million such commands every second.

While some people still occasionally program computers by setting up these millions of microprocessor commands directly by hand, you can appreciate that writing something like a computer game or wordprocessing program in this manner would be extremely laborious and tedious. You see humans and computers don't speak the same language.

To help with this immense problem associated with programming a computer, people began to invent tools called "programming languages" and other tools to translate these "human programming languages" into the language that the microprocessor understands.

There are many programming languages used to program modern computers. The one we are going to speak about in this article goes by the name Tcl/Tk which stands for *"Tool Control Language with a Tool Kit"*. It is affectionately known by the 500000 or so users world wide by the name "tickle tee kay".

Tcl/Tk is a very good language to start to learn about computer programming because:

- the language and associated tools are open sourced and hence free
- the language has been implemented for almost every conceivable computer type (IBM clones, Macintosh, Linux to name but a few)
- it has been around for over 10 years and is widely used in the real world for everything from eCommerce on webpages to quality control in microprocessor factories
- it is a very simple language without too many quirky syntax issues

Downloading All Our Programming Tools

The amount of work you will need to do to get set up to write programs in Tcl/Tk will depend on the type of computer you are using.

For Linux users you are likely not going to have to do anything because Tcl/Tk tools are included with most Linux distributions including the very popular Knoppix live CD version.

Microsoft Windows or Macintosh users, however are going to have to download and install the Tcl/Tk tools from the Internet.

Tcl/Tk is maintained today by a company which goes by the name Active State (`http://tcl.activestate.com`). All the basic tools you will need to start exploring Tcl/Tk as a programming language are freely available for download at the Active State website following the link above.

The only other tool you will require for the code examples in this series of articles is a very simple text editor.

Once again Linux and Mac users will have plenty to chose from. For Windows users the simplest chose will be Wordpad. The important thing to note is that you'll need a text editor and not a word processor, as the latter adds lots of hidden characters into the resulting file to control fonts and formatting.

Writing Our Very First Program

At this point you should have the Tcl/Tk tools set installed on your machine.

You must be anxious to write your very first computer program.

We are going to be using a feature of the Tcl/Tk tools that are not available in every computer language: the ability for the program to be written and run interactively. In other words we are going to type something and then immediately run the program. This mode of programming is often called "interpretive mode" and it is a great way to test out simple programs and programming constructs.

The instructions below will describe how to write this program using Windows. For others the procedure will be similar.

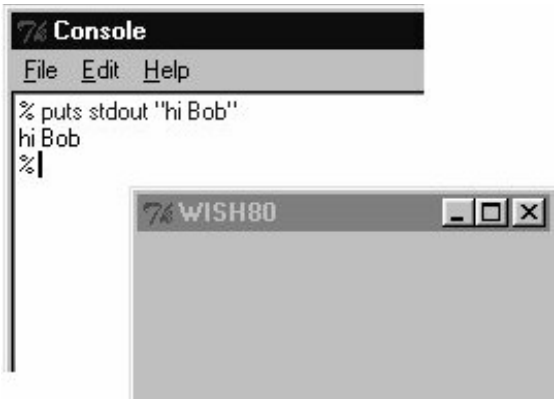
One of the tools you installed on your computer goes by the name **wish**. Which stands for **window shell**.

This should bring up two windows: one called *Wish* and the other, *Console*. The console window will usually have a % character (this may vary though) on the left hand side of the first line. This character is the prompt.

This program is going to contain the following line (to be typed after the prompt)

```
puts stdout "hi Bob"
```

If you end the line by hitting the <enter> key you should see the string inside the "" appear on the line below



Congratulations!

You have just written your very first Tcl/Tk program!

Feel free to experiment with the words inside the "" ... perhaps by inserting your name.

Notice that you can use the "up arrow" key to recall the previous line and make changes by using the "left or right arrow" to edit the line.

Explanation of the First Program

Tcl/Tk is a particularly simple language and every statement will always have the same elements:

```
command arg1 arg2 arg3 ...
```

In our little programming example the command was `puts`. This command is used to "put things" or "display things".

The first argument in our example was: `stdout`

For us `stdout` will always mean the console screen.

One thing you will find is that programmers in general are a lazy lot. The laziest programmers of the bunch are those who write operating systems or programming languages. That is why we end up often with cryptic commands like `puts` and clipped arguments like `stdout` ... it saves typing.

The second argument in our example was: `"hi Bob"`

The "" were used to group these words into a single argument ... otherwise the wish program would have taken "`hi`" as the second argument and "`Bob`" as the third.

The `puts` command would only be expecting 2 arguments and our little program would not have worked very well. This concept of grouping is in almost every computer language ... they don't always use "" as the group markers, however. The series of words inside the "" are often referred to as a "string" ... which is short for "string of characters".

In the Beginning ...

When computers were first invented the way you interacted with them was far from friendly.

Humans like colour and graphics.

Computers find those things a big pain in the chips.

The simple one line program above was an example of a command line or text based interface. The input was all done with the keyboard and the output was in the form of more text on the console.

To this day this remains one of the most economical ways to interact with a computer.

Many programmers prefer to type commands rather than use the mouse to click icons. Nonetheless, graphics and graphical user interfaces (or GUIs as they are called) are very common on modern computers.

Fortunately for us Tcl/Tk comes with very capable and powerful graphical "sidekick": Tk.

Tk looks at the graphical world as being composed sets of what it calls widgets. Like real world widgets, Tk widgets have a basic shape with a colour, texture, size etc.

The widget model is a good one for us as programmers because we don't have to concern ourselves with the thousands of details that go into teaching a computer how to draw buttons on the screen.

The Tk language looks after that for us. All we have to do is decide how our button is going to look.

- How big it will be.
- What colour it will be.
- What shape it will be.
- What text will appear inside the button.

These characteristics of a button are all modifiable from within our program. This gives us a very rich set of combinations of button attributes that we can use in our particular program.

The First Graphical Program

Just as we did above for our text program we need to be running wish.

In the console window at the prompt you want to type the following line

```
button .hello -text "hi Bob"
pack .hello
```

When you hit <enter> this time the wish window should change and a button should appear. The words "hi Bob" should appear on the face of the button.

Congratulations! You have written your very first graphical program.

Explanation of the First Graphical Program

Just to refresh our memory all Tcl/Tk statements are composed of:

```
command arg1 arg2 arg3 ...
```

In the first statement of our program the:

```
command = button
arg1 = .hello
arg2 = -text
arg3 = "hi Bob"
```

The command button specifies the type of widget we want Tk to create for us.

`arg1` specifies the name we wish to give to our newly created button. **The "." in front of the name is important.**

We will describe this more fully shortly ... but for now think of the "." as standing for the top level window shell (wish) window.

`.hello` then means name the widget `hello` and put in into the toplevel wish window.

`arg2` specifies one of the many attributes for this button that we want to set. In this case we want to set the text which will appear inside the button.

`arg3` then specifies the actual text string that we want to place in the `text` attribute for our button called `hello`.

At this point we have only told Tk to create our button. We have not told Tk where to draw the button on our toplevel window. The second statement in our little program

```
pack .hello
```

tells Tk to draw our button widget called `hello`. When this statement gets executed our button magically appears on the wish window.

Save the Program

As with all programs, it is a good idea to save them. You do this as you would any text file, just following the time honoured protocol of adding `.tk` as the extension. When you come to reload the program, you should be able to run the program directly from your filemanager – or can you?

The wish program is complaining that window name `hello` already exists in parent.

The reason for this is that widgets stay around inside the wish program until we explicitly destroy them. Because we had already created the button called `hello` when we typed our program into the wish console by hand, we are being told that we can't create another button widget of the same name.

Don't worry about the word parent ... in computereeze windows that start from other windows are referred to as "children" and higher level windows as "parents". In this context "parent" simply refers to the environment in which you are running your program.

To fix our little bug we need to add another Tcl/Tk statement to the beginning of our computer program. Here is where the file comes in really handy. You need to return to your text editor program and load up the file where you stored your previous two statement program. You need to open a new line at the beginning of your file and insert the following statement.

```
destroy .hello
```

This tells the Tk program to get rid of the previous button widget named `hello` ... because we want to recreate it.

If you now save your file again and repeat the above File/Source procedure to run the program it should work like it did in step one. In fact you can repeat this over and over again and the program will always run.

Have some fun experimenting with different strings inside the double quotes ("") on the button line in your program. For those of you who are really ambitious try adding another attribute to the end of the button line which will set the button colour. The format for that attribute is:

```
-background red
```

Some Programming Wisdom

As you may have gathered by now programming has a healthy dose of repetition involved.

In fact the sequence of steps that you have just been through is so repeated in every programming language that is often referred to as the "Programming Cycle".

While there are many books written on this subject and many opinions as to which variation of this cycle is best in large development projects ... we fortunately do not have to concern ourselves with anything but the basics. With our Tcl/Tk toolset our cycle consists of the following steps:

- entering/changing lines of code via our text editor
- saving those changes to a file
- telling the Wish console where that file is and instructing it to Open the file and run the program
- observing the resulting program behaviour

While our little program had only a simple "bug" in it and with our hints it only took one iteration to solve it, this will not always be the case for larger more complex programs. In fact it is often very desirable to attack a complex programming problem by starting with a very simple program and adding features to it step by step. At each step you would iterate through one complete programming cycle as described above. If you take nothing else away from this article remember this:

"all complex programs can be built up from simple parts which already work"

Creating Our Own Tcl/Tk Commands

Any computer language contains a relatively small set of basic commands.

If that is all that the language could do it would not be very useful or practical. It would be better if we could extend existing commands or create entirely new ones as we needed them. Fortunately Tcl/Tk has just that capability.

Before we begin we need to discuss and reinforce the relationship between the "Wish" program and our little programs.

Tcl/Tk is what is known as an "interpreted language". What this really means is that the statements in our little programs are "digested" one by one inside the "Wish" program as they execute. This has no real impact on what we are learning except that phrases such as "Tcl/Tk interpreter" or "interpreted by Wish" etc. will appear in the article as we go forward.

Suffice to remember that our programs require Wish in order to run.

Extended Command Example

Let us illustrate how to extend the basic Tcl/Tk commands by way of an example.

Using the editor you used above create the following program by typing in these statements:

```
# first extended command example

# scott - entry point
proc scott {mycolour} {

puts stdout [format "lastcolour was %s" $mycolour]

return $mycolour

} ;# end of proc scott

# main - entry point
# These statements will be executed at startup
set lastcolour red
destroy .hello

button .hello -text "change me to green" \
      -background red \
      -command {set lastcolour [scott $lastcolour]}
pack .hello -padx 20 -pady 20
```

The first thing you will notice is the statements which begin with the # character. The Tcl/Tk interpreter program (eg. Wish) will simply ignore these statements. This type of statement is used to add some description to a computer program. In addition blank lines are ignored as well with one important exception noted below.

Occasionally Tcl/Tk statements get very long. Long statements are very difficult to read because a portion of those long lines is often "off the visible screen". To make long statements more readable Tcl/Tk has assigned the \ character a special significance. It is often referred to as the continuation character.

What the \ tells the Wish program is that the line that follows is really to be treated as part of the same line that the \ appears on. As such you cannot follow a line ending in \ with a blank line. In the example above we have used the continuation character to divide this line into two lines:

```
button .hello -text "change me to green" \
       -background red \
       -command {set lastcolour [scott $lastcolour]}
```

Notice that we have also introduced another argument for the button widget. This argument is the `-command` argument. It is probably the single most powerful feature of the Tk part of the Tcl/Tk language.

What we are saying is "whenever you press the button with the mouse jump and execute the statement enclosed by the `{}` immediately following the `-command` argument".

This is what allows Tk programs to actually interact with us humans.

Before we try to understand the `-command` we have for our button we need to understand a couple of other new Tcl/Tk ideas.

In Tcl/Tk variables are set using the `set` command. An example in our code is:

```
set lastcolour red
```

In this example our variable is named `lastcolour` and the value we are assigning to that variable is `red`.

We can then use that stored value at other places in our program by adding a $ symbol to the front of the variable name. You can see a couple of examples of this in the program example above. The most interesting of these statements also merits further explanation:

```
puts stdout [format "lastcolour was %s" $mycolour]
```

In this statement we have introduced the first of our "statement within a statement" constructs. The inside statement is enclosed in `[]`. The way the Wish program handles this is that it first executes the inside statement. In this case:

```
format "lastcolour was %s" $mycolour
```

As with any Tcl/Tk statement this produces a result. The Wish program then executes the outer statement

```
puts stdout "whatever the result was from the
                                    format statement"
```

Compound statements are very important Tcl/Tk constructs to understand.

The `format` statement itself is a very powerful and important Tcl/Tk statement.

The `format` statement can be viewed as a kind of "text creation statement". It is often desirable in a computer program to display the result of a computation or the value stored in a variable as part of a human friendly phrase. For this purpose we use the Tcl/Tk `format` statement. If you examine the first argument to the `format` statement:

```
"lastcolour was %s"
```

You can readily identify a portion of the "human friendly" phrase. In this case it would be `lastcolour was`. We would like to be able to finish this phrase using the contents of the variable, `mycolour`. We do this by using a special format character called a placeholder. In this case it is `%s`. The `format` statement understands a whole range of placeholder types. The `%s` means that "the variable I'm holding the place for is a text string". The third argument in our `format` statement identifies the variable value that we wish to have stuffed into the spot held open by the placeholder. In our case that is:

```
$mycolour
```

The resulting phrase in our example might be something like:

```
lastcolour was red
```

if the value stored in the `mycolour` variable was in fact `red`.

We now have talked about all the new Tcl/Tk constructs in our little program example with the exception of the statement:

```
proc scott {mycolour} {
```

The command `proc` in this statement is informing the Wish interpreter program that we want to create our very own new Tcl/Tk command. In this case we are calling our new Tcl/Tk command `scott`.

The first matching set of `{}` tells Wish that our new command will accept a single argument ... and with our command that argument will be named `mycolour`. The single `{` at the end of the line denotes the beginning of the block of Tcl/Tk code which will constitute our new command called `scott`. This logic will continue until we encounter the closing brace `}` as in:

```
} ;# end of proc scott
```

We have introduced another form of Tcl/Tk comment here as well. If you want to add a comment to the same line as a Tcl/Tk statement is on you must use two characters `;#` together to denote the beginning of this comment string.

The `return` statement presents the result that we want our new `scott` command to give back or return. In this simple example we are simply returning the same value that we gave in as an argument.

```
return $mycolour
```

Everything between the braces constitute the logic associated with our new command called `scott`.

In our simple example the `puts` statement and the `return` statement are the only statements contained within `scott`. Our new commands can be as simple or complicated as we need to make them. This capability gives our computer language a huge degree of flexibility and richness. It is as if we can take words in our spoken language and combine them together to create brand new words of our own.

Having defined our new command we can now have it execute every time we press the button with the mouse by adding the following to the `-command` argument:

```
set lastcolour [scott $lastcolour]
```

Remember how the compound statements execute. Inner part first ... outer part next.

Go ahead and type this program in and execute it. Remember to press the button with your mouse. You should see an output that looks something like this:

A More Useful Example

Let's go ahead now and make our little example more useful.

```
# a more useful example

# scott - entry point
proc scott {mycolour} {

puts stdout [format "lastcolour was %s" $mycolour]
if {$mycolour == "red" } {
  .hello configure -text "change me to red" \
                   -background green
  set newcolour green
} else {
  .hello configure -text "change me to green" \
                   -background red
  set newcolour red
} ;#end if else

puts stdout [format "newcolour is %s" $newcolour]
return $newcolour
} ;# end of proc scott

# main - entry point
set lastcolour red
destroy .hello

button .hello -text "change me to green" \
       -background red \
       -command {set lastcolour [scott $lastcolour]}
pack .hello -padx 20 -pady 20
```

What we have done in this enhanced example is added some new statements to our `scott` command.

The first of those new constructs is what is known as the `if` ... `then` ... `else` branching construct.

Computers excel at this kind of true or false branching decisions. Most of the so called "intelligence" in computer software can be traced back to statements like these in the code. The `if` statement allows our little program to alter its execution path depending on the contents of a variable.

It begins with the statement (and it is important that in your code it is written this way):

```
if {$mycolour == "red" } {
```

In our little example this code block includes:

```
.hello configure -text "change me to red" \
                 -background green
set newcolour green
```

Here we have introduced our final Tcl/Tk concept for this article ... the ability to alter the attributes on a Tk widget even after it has been drawn on the screen.

We can alter the colour of our button. We can alter the text in our button. We can do all these things based on how we code our program logic and flow The statement which does all this magic is:

```
.hello configure ...
```

Recall that when the button was defined it was given the name `.hello`.

Thus the `.hello` configure command construct is saying "I want to change something about the widget I have called `.hello` ... in this case a button widget".

Notice in this example we are changing the text in the button as well as its background colour. When this statement executes that is exactly what will happen on the screen.

The other statement inside our true branch of our `if` statement simply allows us to remember the new colour we have assigned to our button.

The closing `}` for our true branch is found as part of the `else` statement below:

```
} else {
```

Which then moves on to the false branch as you'd expect.

It is hard to see from this very simple example, but this is how computers get much of their apparent intelligence. You can see that it isn't intelligent at all! The intelligence is all supplied by the human programmer in the form of these true and false blocks of code.

[concluded at foot of next page]

An Introduction to Objective-C

Part 5 – The Philosophy Behind Objective-C

D.A. Thomas

As mentioned in Part 1, it was Cox's idea to apply the principles of Smalltalk to a lower-level language like C, thus producing a useful hybrid which combined the advantages of both. Polymorphism is implemented by late binding and is not dependent on inheritance, as it is permitted to send any message to any object. This gives the developer the maximum flexibility and enables techniques such as delegation and notification, which are very important in the construction of complex graphical user interfaces. These techniques are very difficult to implement in C++, as the compiler needs to know at compile time about all the member functions that are going to be called on an object. Objective-C programmers tend to find C++ very restrictive in this regard, and they typically say that they are having to fight against the language instead of being able to use it as a tool. In fact, I must admit to having experienced this feeling myself. Supporters of C++, on the other hand, say that having the compiler do the checking for you eliminates at an early stage a common class of errors. The fact remains, however, that certain problems are very difficult to solve in pure C++, and vendors like Trolltech (who supply the Qt application development framework) and Microsoft provide some quite elaborate mechanisms in order to get around this.

Another area where the message-passing model of Objective-C shines is distributed objects; with NeXT/Apple's PDO invoking a method belonging to an object in another thread, address space, or even a remote host differs little from invoking one in a local object..

It has been said that C++ excels at solving closed-world problems, while Objective-C's realm is an open world. Anguish et al. write:

> **Closed-World Applications**
>
> The engine compartment of an automobile is analogous to closed-world applications. It is desirable to know in advance every component that will be inside the engine compartment and how they will fit together. Engines are carefully designed and their design is seldom modified after they leave the factory. Any variation in the connections between engine components is probably a manufacturing error. Languages such as C++ and to a lesser extent Java provide language-level features that are well-suited to solving closed-world problems. The static strong typing used by C++ and Java enables the compiler to verify that all components fit together as planned at the cost of making variation, redesign, and modification of existing applications more difficult. Some applications require the verifiability of static strong typing and can overcome the reduction in flexibility. Some programmers are just more comfortable solving closed-world style problems and might never be satisfied with Cocoa because it is designed to solve open-world problems.
>
> **Open-World Applications**
>
> The passenger compartment of an automobile is analogous to open-world applications. Any constraints on the type or number of people and things that can be placed in the passenger compartment detract from the utility of the automobile. Some constraints are inevitable, but the designer of a passenger compartment must strive for maximum flexibility. The price of that flexibility is that the designer cannot anticipate everything that might be put in the passenger compartment. The designer must work with incomplete information. Objective-C provides language-level support for solving open-world problems. Objective-C objects can operate with anonymous objects in different applications. It is possible to send messages to objects even though the receiver might not understand the message. It is possible to add behaviors to existing compiled objects without recompiling them. The flexibility provided by Objective-C aids the development and life-cycle modification of most desktop applications, but the cost is that the compiler cannot always verify that the components fit together. In some cases, errors that might have been caught by a compiler with a different language cannot be caught until an Objective-C application is running.
>
> Most graphical user interfaces are examples of open-world applications. Restrictions on the type and number of graphical components available reduce the utility of user interfaces. Sometimes it is necessary to create new user interface components that were not anticipated by the original designers and still be able to integrate the new components with the old components. Plug-ins and context menus are other examples of open-world applications. It is certainly possible to create open-world applications with static strongly typed languages, but it is more difficult. It is also possible to use strong static typing with Objective-C and gain many of the benefits at the cost of flexibility. Cocoa and Objective-C emphasize flexibility at the expense of compile time verifiability. Much of the increased programmer productivity attributed to using Cocoa results from Objective-C's flexibility. (Anguish, S., op. cit. pp. 25-26)

My first introduction to C++ was something of a disappointment. To be sure, I was greatly impressed by the power of the 'template' facility to operate with arbitrary types, though it did not seem as elegant as the strong generic typing of SML and Haskell. I expected C++ to come packaged with general-purpose and GUI class es in the same way as Smalltalk, and all I got was `iostreams`! The position is a little better today, as the standard C++ library now incorporates the Standard Template Library, a very versatile basic set of algorithms and data structures, yet I still have the impression that people who work in C++ 'cut their own code' for the most part rather than make the maximum use of code written by others.

I hope to have convinced the reader that Objective-C is very simple in terms of both its syntax and its concepts; much like Java, it eschews 'frills' like operator overloading and multiple inheritance. It has no 'templates' either, which are perhaps the source of the best and the worst of C++. However, the APIs associated with Objective-C, for example those of Apple's Cocoa, tend to have a very steep learning curve.

As the programming language of choice for new Macintosh applications, Objective-C would seem to have a future at least as secure as that of its main platform. In the wider world of object computing, however, I perceive a trend away from low-level hybrids like Objective-C and C++ towards managed object environments like those of Java and the Microsoft Common Language Runtime on the one hand and scripting languages like Perl, Python and Ruby on the other. There are several reasons for this. First, languages in the C family allow the programmer access to raw, unchecked pointers, which, while often efficient, can also lead to application crashes and buffer overruns. The difficulty of providing reliable automatic garbage collection for these languages can also cause crashes (from deallocating unallocated memory) and memory leaks. The Cocoa frameworks use partly manual and partly automated reference counting, which is better than leaving memory management completely to the programmer, but inappropriate use of `-retain` and `-release` can still result in crashes and memory leaks, and of course, there is always the possibility of the retain count never reaching zero because of circular references; in addition, the book-keeping overhead involved with reference counting can be extremely severe where a large number of objects is inserted into and taken out of container objects. Object programming seems to me to demand automatic memory management, something which is possible in C++ and Objective-C only under controlled circumstances, like the 'safe mode' of Microsoft Visual C++ .NET. As computers become ever faster, the undoubted machine efficiency of Objective-C and C++ is perceived as less of an advantage relative to the interpreted, thus slower, but also safer, easier and more flexible, scripting languages.

D.A. Thomas

`[continued from previous page]`

Our false branch in our code simply mirrors the true branch in that we will alter the text and colour to reflect a red button and remember that colour in our `newcolour` variable.

```
.hello configure -text "change me to green" \
                 -background red
set newcolour red
```

The false branch block is ended by the single closing } as below

```
};# end if else
```

If you now type in this program and run it you should see the button toggle between red and green each time you click on it with your mouse.

R. D. Findlay

Acknowledgement

The contents of this article were derived with permission from several of the online iCanProgram lessons offered at `http://www.icanprogram.com/nofeecourses.html`

iCanProgram offers all its courses online for no fees. All we ask of the students is that they make a Cancer Research donation to a local charity in memory of one of our founders.

Reviews

Bookcase

Collated by Christopher Hill
<accubooks@progsol.co.uk>

Paul Writes

It seems to me that books seem to fall into one of two categories: those worth purchasing and those which really shouldn't have made it past a competent technical editor, and unfortunately, the latter group seem to be those aimed at what can be classed as the "beginner" market; the market where the most care should be taken to not only ensure that the technical quality of the material is at the highest possible standard, but that it is simple to comprehend irrespective of the platform being used to learn on.

I am unsure if this is down to market pressures to just have a book on the book stand or to have something to be in direct competition with another book (even if it is from the same company) under the banner of "competition raising standards". What is clear is that if the offending book companies spent that extra couple of weeks to ensure that their books are of that top-notch quality, that the number of not recommended ratings would drop somewhat.

I must thank all of our reviewers for their time and effort spent reviewing books, and if anyone wishes to review books, a quick email to Chris Hill <accubooks@progsol.co.uk> will point you in the right direction.

Paul F. Johnson

The following bookshops actively support ACCU (the first three offer 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

Computer Manuals (0121 706 6000)
www.computer-manuals.co.uk
Holborn Books Ltd (020 7831 0022)
www.holbornbooks.co.uk
Blackwell's Bookshop, Oxford (01865 792792)
blackwells.extra@blackwell.co.uk
Modern Book Company (020 7402 9176)
books@mbc.sonnet.co.uk

An asterisk against the publisher of a book in the book details indicates that Computer Manuals provided the book for review (not the publisher.) N.B. an asterisk after a price indicates that may be a small VAT element to add.
The mysterious number in parentheses that occurs after the price of most books shows the dollar pound conversion rate where known. I consider a rate of 1.48 or better as appropriate (in a context where the true rate hovers around 1.63). I consider any rate below 1.32 as being sufficiently poor to merit complaint to the publisher.

.NET

.NET Framework Essentials by Thuan Thai & Hoang Q. Lam (0-596-00505-9) O'Reilly @ £20.95
reviewed by Andrew Marlow

The book contains a lot of material, providing an overview of the entire .NET Framework. I found the book to be a very helpful introduction to .NET and I do not have a Microsoft background. I rate this book as highly recommended.

Programming at intermediate to advanced level is assumed, as is some exposure to XML and web development. The book aims to help programmers make the transition from traditional Microsoft Windows programming to .NET programming. It provides an introduction to its languages, Common Language Runtime (CLR), and APIs.

Security is one of the subjects that is not covered very deeply, although it is mentioned several times in passing. The book is supposed to be concise and cannot cover everything, but this, coupled with the fact that there is no bibliography (other relevant books are mentioned as footnotes on various pages), was a minor defect, especially when one bears in mind that some of the changes of .NET 1.1 are security-related.

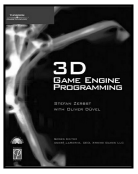
The book is concise and understandable but rather condensed, so whilst the book is small it is not an easy read, especially when it gets to the code. The style is good; the difficulty is just the sheer amount of information and the fact that it is packed into just over 300 pages. The .NET overview is particularly well written and helps to get the reader started before there are many code examples.

The CLR chapter contains a "hello world" example that clearly shows the benefits of a uniform API across languages. Meta data, assemblies and manifests, and language integration are very clearly explained. This is rounded off in chapter 3 with a mixed language example using a polymorphic vehicle class.

The "Working with .NET components" chapter is the point at which the book becomes hard going, especially for someone that is not very familiar with Microsoft technologies. The ideas come thick and fast and many code example fragments are given. It also flits between languages rapidly, partially to show how language neutral .NET programming can be. Some familiarity with all the languages used will help the reader considerably (managed C++, C#, J#, VB).

The next two chapters are harder still, with much XML involved and many pages of XML code. In my view, these chapters require the reader to have had some exposure to ADO (ActiveX Data Objects) and SOAP. The pace slackens slightly in the ASP.NET chapter, and further still in the Windows Forms chapter, where the uniform cross-language set of GUI classes are covered, and the ease of use illustrated by making .NET GUI development similar to VB development.

The .NET and mobile devices chapter covers mobile controls which are used for rendering web content on phones and PDAs, and the .NET Compact framework, which is used for developing applications that run on pocket PCs.



3D Game Engine Programming by Stefan Zerbst with Oliver Duvel (1-59200-351-6) Thomson @ £38.99
reviewed by Dáire Stockdale

Judging purely by appearances, this looks like it might be a 'serious' book. It's a large book (over 800 pages), with a cover design vaguely reminiscent of the Microsoft press covers, a blue flash in the top left corner announcing that this book is 'Thompson Course Technology' with Professional/Trade/Reference as bullet points. Only the line telling that its series editor is the CEO of Xtreme Game LLC, André LaMothe, hints at the low value of the book's contents.

I have had the misfortune to review several of the books in LaMothe's Game Development Series, and they have been uniformly bad. Worse, I believe that these types of books can be misleading and damaging to the games industry, promoting code and programming practices of the worst possible kind, and passing these off as acceptable professional standards. Young impressionable programmers are taught bad habits early on, many of which seem to last a lifetime.

This book is a good example of such a dangerous text. First of all, neither of its authors actually work in the games industry, and there is no mention that either has in the past. We are told Stefan Zerbst founded a popular German site for hobby developers and that he lectures about game programming at an unnamed German university, and his co-author Oliver Düval is a project manager in the field of support for a German software company.

Reading the book was a depressing and maddening task. It is light years apart from the mature professional texts that I suspect many ACCU members respect and enjoy. The book is muddled, uncertain about what is supposed to be teaching, and whom its audience is. I suspect the reason most 'game development' books feel the need to include ham-fisted tutorials on crucial topics like virtual functions and abstract classes is that no matter what they may proclaim on the books cover blurb, they know the real audience is young, inexperienced and unsuspecting programmers.

Many of the habits promoted within the book would jeopardise your marks if done during a university project, and should jeopardise your job if employed as a software engineer. I don't even know where to begin criticising. The code looks like it has been heavily influenced by reading Microsoft's early code, copying many of their bad habits, such as misuse of trailing underscores,

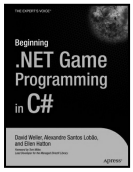
typedefining structs, hungarian notation, global variables etc, with many new bad habits added, such as mingling new with malloc, C style code dressed as C++, classes that are not copy-safe, uncommented code and much more. The author even defends some of his practices (use of globals) with a line saying "...you will find a big software company from Redmond does the very same thing, so I guess you and I can live with that quick 'n dirty solution." Well I am sorry, I cannot, and I certainly do not expect to find it encouraged in a course textbook.

Some more choice advice from the author is: "Be warned: Do not show this code to hard-core C++ object-oriented programming gurus." Very good advice! It might make them ill! He continues: "These programmers would criticize this code because when you define an interface, you should not define attributes in the abstract class. This is a code design issue." On the subject of complex numbers: "If you are really interested in this weird stuff, I recommend a doctor and a good book about analysis".

Even if the code were of a professional standard and the text pleasant to read, I am not sure what value the book would have as an industry textbook. Most authors in these series seem unaware that the console market is the focus of the game industry, and that PC games account for only a tiny portion of the market. It is the exception proving the norm. A useful book on game engine design should make the novice programmer aware of the real design issues that the industry faces: rapid development times, portable code and data, severe memory restrictions, and real-time performance. And of course there is the coming parallel processing problems of the next generation of consoles.

In chapter 6, instead of explaining the meaning of the buzzwords he is using, the author says "...you need to check a good book or tutorial about 3D graphics first..." My suggestion strongly would be to bypass the confusion and risk of learning some bad programming habits that his book poses, and go directly to the good book instead.

Not Recommended.



Beginning .NET Game Programming in C# by David Weller et al. (1-59059-319-7) Apress @ £28.50

reviewed by Paul F. Johnson

This is a very well written book, which really is written for beginners. Unlike other books, it actually takes the time to show the maths behind the likes of collision detection, the differences between a 2D and 3D game and why using OO techniques for games programming is good. The book even covers (in a very simplistic way) games engine and the design process.

The book begins using GDI+ (which makes it accessible to those using Mono) and then moves over to DirectX in later chapters for 3D programming. I would of preferred to see something on using C# with OpenGL (which, in my opinion, offers far greater flexibility and wider platform coverage to the programmer than DirectX), but the coverage and explanations give enough detail for the reader to know what is going on and to be able to use the ideas shown.

There are a number of example games that are built up over the course of a chapter. The games are pretty lightweight, but this is a book for beginners, so that is acceptable – it would be too much to expect a beginner to be writing a complete Doom4 engine at this stage!

Where the book does fall down is that it covers too much too quickly for me. Uless you have time to download a 7.5Mb zip file from the APress website, the partial listings, while useful for demonstration, are by themselves not a whole lot of good. The use of pseudo code was helpful for explaining what is going on, but the lack of comments in the source reduced the pseudo code's use.

Overall, it is a rather decent book. Recommended with reservations.

C# Web Development with ASP.NET by Jose Mojica (0-201-88260-4) Peachpit Press* @ £16.99

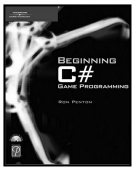
reviewed by David Sullivan

This book has a misleading title. In the introduction the author states that "This book makes no attempt to teach Web programming" and also that the examples contain very few web based examples. Also stated is that "The concepts can be applied to other application domains other than Web based applications. It is as if the publishers changed the title to generate more interest. If you want to know about ASPX pages and web controls look elsewhere. There is a short chapter at the end of the book called "Web Based C# Projects" as a token gesture.

But you might be interested in this book (one of the Visual Quickstart series) if you would like to learn from the basics the fundamentals of the C# language. Particularly if you like enjoy a practical style with plenty of examples to work through. The examples are simple but do illustrate the concepts. In the style typical of the series, the book presents a good deal of visual material, examples and practical instruction to assist the reader. It is certainly much cheaper than most technical books at 17 GBP.

The book could not be described as a thorough reference on C#, It is a book of course material but is a practical how-to reference on the C# language features. Some of the descriptions of concepts are confusing. There are many examples in the book that fall short of what is required (e.g. encapsulation, value and reference parameters), particularly for those readers who have no prior experience of such concepts. I think a professional author should really do better than this. The introduction does states clearly that the book is for beginners, and for those that have no object programming experience. However, in some cases the author makes assumptions about the reader's knowledge. If you are familiar with similar concepts (Java, C++) then you will likely understand what the author is trying to say. Not the best book for a total beginner though.

If you are interested in a quick introduction to C# with plenty of examples, ideally have some programming experience already, and you just want to learn C# (not the web based C#), then you might be interested in this book.



Beginning C# Game Programming by Ron Penton (1-59200-517-9) Thomson @ £18.99

reviewed by Duncan Kimpton

This book can largely be split into three sections: C#, DirectX, and building a game.

The C# section was depressing as it contained numerous factual errors, and oscillated between assuming in depth knowledge of programming from the reader and explaining simple concepts in baby steps. There are much better introductory books for C# and you would be well advised to skip this section and pick up one of those instead.

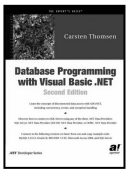
Since Managed DirectX is changing at a crazy rate the DirectX section suffers the immediate downfall of being out of date, readers should be aware that there are other books that are up to date.

I found the rather unstructured "here is my framework let's dissect it" approach very unsatisfying – I would have preferred to see an explanation of the concepts used as a basis for building up a framework. With the tear down approach I learnt a lot about the author's code, but not many of the principles behind it. Other readers may not share my preference.

All the major sections of DirectX were covered, however for a beginner it would have been better to spend space on buffered DirectInput instead of things like force feedback.

The last section relates to actually building a game, but again this feels rushed and the completed project is incomplete and unexciting, a situation that fails to inspire future study.

The crisp clean page layout, slightly humorous writing style, and the concept of teaching programming from a gaming perspective are all good points to this book. Unfortunately they cannot outweigh my overall impression that the author tried to cover too much (and yet also too little) in one book, without a lot of thought to the actual process of learning that a student must undertake. Not Recommended.



Database Programming with Visual Basic .NET by Carsten Thomsen (1-59059-032-5) Apress @ £31.20

reviewed by Mick Spence

The book is large, by the time all 26 chapters, the numerous appendices and index are finished the page count is just over 950. It attempts to cover using Visual Basic .NET and the ADO.NET layer to access a database, where "database" is mainly an SQL Server (three different providers are covered) but not exclusively as MySQL, Oracle, DB2 and Access also appear.

It is assumed that the reader does know what a database is, but most other concepts are explained. The same cannot be said for Visual Basic; few, if any, allowances are given to

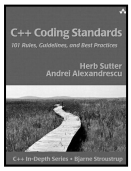
someone trying to learn VB using this book, but to be fair that is not the book's target audience.

A large number of coding examples appear at suitable points in the text. These are available on-line to save typing. Most appeared technically accurate and cover the subject under discussion reasonably well (I didn't try all examples but those I did worked). Most of these examples relate to a fictional application that gives a common theme that runs throughout the book.

The writing throughout the book appeared accurate and fairly clear, however it did feel a little verbose at times and I struggled to stay focused while preparing this review (however doing this over the Christmas holidays maybe didn't help).

This book's cover states, "Learn the concepts of disconnected data access with ADO.NET" and to be truthful it does pretty much that. I'm sure that someone trying to learn ADO.NET using the VB.NET language could do a lot worse than getting this book. I also think that the book would double as a fairly good reference book where the reading only small sections of the book would make its wordiness less of a problem.

C & C++

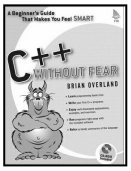


C++ Coding Standards by Herb Sutter & Andrei Alexandrescu (0-321-11358-6) Addison-Wesley @ £24.99
reviewed by Anthony Williams

The key points from the book are neatly summarised in the 12-page "Summary of Summaries" at the back of the book. This lists all 101 guidelines, with both their title and summary; the primary reason for reading the main body of the book is to understand the background and reasoning behind the guidelines, as well as any exceptions. These guidelines are not casually thrown together; they are well researched, with extensive references. It is also worth reading the introductory text to each section, as this pulls the guidelines from the section together; a nice touch from the authors is the selection of a "most valuable Item" from each section, one which you really really ought to follow.

Other books from Addison-Wesley, such as Effective STL, have an easy-to-read layout. In this book, in common with the others in the C++ In Depth series, the text is slightly too small and the margins slightly too narrow, so there is too much text on a page. This, combined with the high density of technical content, makes it hard to read cover to cover in one go. There are also a couple of errors; though nothing major, they do detract from the authority of the book, given the nature of the content.

In the preface, the authors state that they intended each item to be short, non-controversial, authoritative, and something that needs saying, and I believe they have achieved that. In any set of coding standards I usually find something that I disagree with, but there was not a single item that grated on me here. I agree with the authors that this book is something you should reference from your own coding standards. Highly Recommended.



C++ Without Fear by Brian Overland (0-321-24695-0) Prentice Hall* @ £19.99
reviewed by Mark Symonds

The book is aimed at teaching non-programmer how to write programs in C++ and comes with a companion CD which contains of the source code and a version of the GNU compiler.

There are numerous problems with the book; templates and the STL are not covered at all and the early examples are written in a very C style with all local function declarations at the start of the function rather than where they are used. The author does mention that the scope of the loop variable can be limited to the loop but since the concept of a scope has not been introduced at that point in the book it is difficult to see what the beginner would make of this advice.

Very little advice is given on good programming practice; for instance the `goto` statement is described in the section covering looping constructs but no warnings about its usage is given.

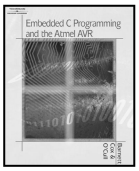
The various examples use `char*` pointers for almost all string handling. The book does describe the basics of the C++ Standard `string` class but goes on to state that all compilers may not support this.

The book contains many examples that do not adhere to the C++ standard in various ways; `main()` returning `void` and the fact that constructors return `NULL` on failure are two examples of this.

This does give the impression overall that the book is describing a pre standard C++ dialect, a very depressing state of affairs in a book copyrighted 2005.

There are also many comparisons throughout the book to the C language, which would be extremely confusing for the novice and since this is the intended audience, it seems totally unnecessary comparing C++ to another language that the reader is assumed to know nothing about.

Not Recommended



Embedded C Programming and the Atmel AVR by Richard Barnett, Larry O'Cull, Sarah Cox (1-4018-1206-6) Thomson @ £42.00
reviewed by Derek Jones

This is an all-in-one book aimed at the introductory programming student market. Is it worth buying this one book, or should more money be spent buying separate books dealing with learning C, programming the Atmel AVR processor (a very similar book by the same authors deals with the Microchip PIC), an extended programming project using this processor, a library reference, plus a CD containing an IDE and C compiler? I would suggest buying two books, a book on learning C and this book (or its companion if your lecturer targets the Microchip PIC).

The C tutorial in this book reads like it has been poorly cut and pasted from a set of more substantial notes by somebody who does not know the language very well. No attempt is made to distinguish between Standard C and the extensions commonly found in embedded compilers. A listing of the C syntax would be useful to the intended readership.

The chapters dealing with programming the Atmel AVR and the embedded programming project (which is a different one from that given in the Microchip PIC book) give lots of details. While they would be out of place in another book, the extensive coding examples are probably of use to the intended readership. Although I did not work through the examples in detail, I got the feeling they were pitched at about the right level for students taking their first programming course. The index also appeared to be extensive enough to be useful.

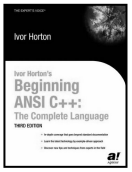
If you are a student on a course based on this book then you will not be wasting your money buying it. If you already know C and are looking to learn about the practical details of writing embedded applications, then this book is one to look at. The material is not of sufficient quality at the price it is pitched, for me give it a value for money recommendation.

Object-Oriented Programming: Using C++ for Engineering a by Goran Svenk (0-7668-3894-3), Thomson, 506pp + CD @ £38-00 (1.92)
reviewed by Mark Easterbrook

Do not waste your money on this book, it does not teach Object Oriented Programming and the programming examples are a mishmash of C and poor C++. For example:

- I would expect a book on OO, when trying to illustrate code diagrammatically, to use some form of class diagram – this book uses flowcharts.
- `float` is used almost exclusively for scientific calculations, even when using library functions that operate on `double`s.
- Pre-processor macros have no place in C++ code; the author seems to like them.
- The STL is not introduced until 80% of the way through the book.

I strongly suspect the author is Fortran programmer and has just translated to C++ with the addition of OO to increase the buzzword count. Not Recommended.



Ivor Horton's Beginning ANSI C++ 3ed by Ivor Horton (1-59059-227-1), Apress, 1090pp @ £37-50 (1.60)
reviewed by Malcolm Pell

The book's intended audience is someone with little prior programming knowledge or experience.

The book starts well, and I had no trouble understanding the basics of C++. Even though I have previous C experience, I feel that someone without C experience can still use this book to gain familiarity with C++.

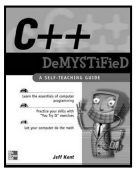
The first 11 chapters cover the basics of C++, which map quite well to the features provided by C, so should not present any major difficulty to either a C user, or someone with little programming experience.

Chapters 12 to 20 cover features which are pure C++, and thus new to someone like myself coming from a C programming background.

I was surprised that 'Input and Output' is not properly discussed until chapter 19. Given that most of the example programs produce some sort of output, I would have thought that an early chapter on some basic I/O code would be beneficial to inexperienced readers.

There are plenty of sample code chunks in every chapter, and lots of useful exercises which readers are strongly encouraged to undertake. There is also a Code ZIP file that can be downloaded from the APRESS Web site.

Overall, I would suggest that this book is considered by someone who desires to learn C++. Do not be put off by the number of errors found by myself and other readers. In some ways, finding these errors gave me confidence that I have understood the subject material. See the Long review for a list of errors.



C++ Demystified by Jeff Kent (0-07-225370-3), McGraw Hill Osborne, 348pp @ £12-99 (1.54) reviewed by James Roberts

This book advertised itself as 'simple enough for a beginner, but challenging enough for an advanced student'. I would grudgingly agree on the former, but strongly disagree with the latter.

The author is strong on explanation, writing in a rather chatty and jokey style. I suppose this might be useful for readers who like the technical input leavened slightly. I found it irritating.

After a little investigation, it turned out that this was in-fact a book on C, with a little bit of C++ thrown in. For example, using `cout` rather than `printf` for terminal output. However, the word `class` only appears in a 'what to study next' chapter at the end. I think that this is a fatal weakness. I have no idea how a reader could be expected to understand a description of the `ofstream` functionality, without knowing about methods or classes.

Another criticism that I have is that the author expects the user to have Visual Studio available. Although he says that alternatives are available, it might have been nice if he had recommended one or two, or even just gave a URL for download.

Some examples will not compile, including this one that I found slightly amusing:

The following...are different in syntax, but identical in effect:

```
char name[] = {'J', 'e',
               'f', 'f',
               '/0'};
char name = "Jeff";
```

Yup, both fail to compile on my system too.

There are a number of other problems, including: no coding exercises; 'rules' which are given on one page and broken on the next; leaving function parameters as unnamed because the names 'serve no purpose'.

To end on a more positive note, there were some areas that were not badly covered (e.g. dangers of `cin` where a user might type a character when a number is expected). However, this does not make up for the inadequacies of the remainder of the book. Not recommended.

Java

Concurrent Programming in Java by Doug Lea (1-201-31009-0) Addison-Wesley* @ £34.99 reviewed by David Caabeiro

When I got this book my intent was to improve my understanding of the concepts of the language regarding concurrent programs design. I must say I got a nice surprise, given that this turned out to be quite good material for concurrency in general, regardless the language used. On the other hand, this implies that the book's title is not the most appropriate, as it suggests to be restricted to the Java language. Far from it, anyone with an interest in this topic will benefit from this book.

The book is divided in four parts, starting with basic concepts to get you ready for the next topics, which include concurrent patterns, design forces behind concurrency, synchronization, state dependency and threads. It also contains extensive code examples.

The author expects you to have a good background in OOP, patterns and Java (though knowledge of other strongly typed languages such as C++ will suffice). So if you are a newcomer you would better look at other places first.

Covert Java: Techniques for Decompiling, Patching, ... by Alex Kalinovsky (3-672-32638-8) SAMS* @ £21.99 reviewed by Silvia de Beer

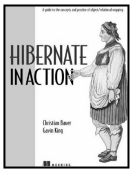
This is a very interesting book if you are a Java architect or developer. It makes you reflect on topics which are inherent to the Java language like the fact that Java is compiled into bytecode. This impacts heavily on the security aspects of your Java programs. This book is not advanced enough for people who already regularly use decompilation, obfuscation of code and profilers. It rather forms a useful and understandable introduction to all those topics where a lot of developers would be ignorant.

The book contains nineteen easy digestible chapters which cover the various topics. The book is well written, always clearly understandable, with code examples, with concise summaries and review questions.

It is interesting to see how easy bytecode can be decompiled, using for example the JAD or JODE decompiler which return the source code, almost exactly the same as the original source code, only without comments. The logical consequence for a commercial package is thus to use an obfuscater of the code, to make the decompiled code difficult understandable. The various obfuscating techniques are explained. The fourth chapter explains how you could call package or protected members of another package.

The book covers useful techniques, as for example using an omniscient debugger, which differs from a normal debugger in that it records the history of an execution. The section on the security manager is worth reading as well, because it shows how easy it is to replace the configuration of a security manager. A security manager can very easily bypassed by ill-wanting people if you do not take any precautions. The last chapter is explains how to implement a license manager in your java code, using encryption techniques.

There was only one unrealistic example in the book, about patching native code in C. It briefly talked about changing assembly code, not very realistically in my opinion, but interesting to see the difference between patching java bytecodes and assembly code.



Hibernate in Action by Christian Bauer & Gavin King (1-932394-15-X) Manning @ £40.50 reviewed by Peter A. Pilgrim

This book is about a next generation Java database persistence technology. It is about one particular framework called "Hibernate", which allows developers to work directly with POJOS (plain old Java objects) instead of programming basic create, retrieve, update and delete (CRUD) operations traditionally with the standard JDBC API.

The framework achieves transparent persistence by using XML mapping files to associate the fields & methods in your POJOS to the tables and columns that exist in a RDBMS.

The book is divided into nine chapters, three appendices, a reference and an index. The first chapter launches the reader into esoteric world of object relational mapping (O/RM). The authors provide an overview of the contemporary thoughts of object persistence in terms of the Java platform.

After 31 pages we are into second chapter that properly introduces us to Hibernate and its integration. We start with the simplest POJO class. Persistence, however, is not a free lunch. Hibernate requires an XML mapping document. The authors discuss the main objects and interfaces and, then, explain how to perform queries. One large piece of advice I can recommend when using Hibernate is to switch on the debuggable logging facility to see generated SQL commands.

Mapping your own persistence classes is the theme for chapter three. The authors have their own working example application, available on-line, called 'Caveat Emptor'. Here is the key differentiator for people who feel they need to be hands-on and experimenting at the keyboard with the O'Reilly Associates "A Developer's Notebook" book. Bauer and King describe the fundamentals of associations, and definition of XML meta data, for each example POJO implementation. They explain naming conventions, controlling inserts and updates, how to retrieve objects, property and class name mappings, and most importantly how to declare SQL schemas. Finally they discuss entity and value types.

The fourth chapter is about working with persistence objects. The author describes the life cycle of the objects as they move from 'transient' to 'persistence' states and back again. Hibernate has one major advantage over other O/RM solutions, it supports

detached objects, which are especially helpful to web application developers. In the latter section of the fourth chapter, the famous HQL makes an appearance for the first time (page 139).

The fifth chapter covers transactions, concurrency and caching. The authors begin with a refresher on subjects you probably already know such as ACID database transactions, the basic two types: direct JDBC and JTA, and isolation levels. From here onwards the content is aimed at practising intermediate developers.

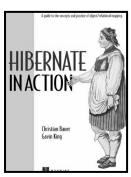
The title of the sixth chapter is called, advanced mapping concepts. It starts with a revision of the Hibernate type system, introducing the built-in mapping types. This chapter explains how to extend the framework with custom user types, including composite user types.

The seventh chapter is about retrieving objects efficiently. Hibernate has three ways for the developer to express queries. The first is through the HQL dialect, and the second is by the Query API: query by criteria and query by example. The third option is to program list with relevant business details, direct code using the SQL. In my opinion some of the material in the beginning of this section should be a part of the third chapter. The authors also discuss sub-queries and executing against native SQL.

The eighth chapter is about writing Hibernate applications. Bauer and King provide advice for designing multi-tier applications that use the framework as their persistence tool. J2EE developers will find the material in this chapter particularly helpful. Here you will find good tips for using Hibernate in a web container, including a well explained example of resource management store a Hibernate session in a ThreadLocal variable.

The final chapter, the ninth, describes the tools that available for the Hibernate developer. In comparison with the O'Reilly Developer Notebook much of this discussion on the hbm2java, the XDoclet generation, and the hbm2ddl is pretty similar, but the O'Reilly book obviously provides more detailed source code and examples.

In conclusion, Hibernate in Action is comprehensive guide and a solid reference to a complex, wide-ranging, and contemporary Java platform to database persistence solution. This book, then, comes strongly recommended, because a contemporary and state-of-the-art topic is very well explained, and especially, because the voices are literally from the horses' mouths.



Hibernate in Action by Christian Bauer & Gavin King (1-932394-15-X) Manning @ £40.50 reviewed by Christer Lofving

The concept of Enterprise Java Beans (EJB) has both advantages and drawbacks.

One big plus is the persistence mechanism that will protect your business data from almost everything; even from the disaster of power failure. A drawback is the sometimes troublesome handwork of creating and mapping entity ("database") beans; another is the costly overhead in handling the lifecycle of such a bean.

Hibernate seems to be the cure for much of this.

It builds on a much simpler persistence mechanism than EJB, and use single-filed POJOS ("Pure Old Java Objects") instead of monstrous EJB objects; the EJB container has to generate a head of a hydra with different files before being able to start up.

Being a new open-source based product, the best information about Hibernate so far has been available on the web. I think this one is among the first printed titles about the subject.

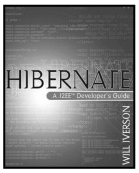
The author takes a pedagogic approach with a basic starting chapter about object/relational persistence. Good for the beginner true, but I think 99% of the potential readers already know this stuff. Rest of the book is hands-on code with explanations. The code samples are small, clean and easy to follow.

Another plus is the nice format. The less than 400 pages (excluding index) suggests that it contains no redundant material.

I had one real disappointment. The cover promises "Getting started". But where in the book is that chapter? I finally found a "Hello World" headline, but that text deals with analyzing the code and configuration settings, simply supposing the reader already has Hibernate up and running.

Who is the book aimed at then? I think it is for anyone already on his way with Hibernate and who wants to summarize the concepts for him/herself. To find out what might be overseen. I do not recommend it if you are starting to learn Hibernate and/or if you are new to the concept of persistence-relation objects itself.

If you are just after a taster, the Web provides you with many samples, tips and free documentation



Hibernate: A Developer's Notebook by James Elliot (0-596-00696-9) O'Reilly @ £17.50 reviewed by Peter A. Pilgrim

Hibernate: A Developer's Notebook is one of the first titles in a new series of O'Reilly books. The notebooks are supposed to contain just enough information to let you quickly learn about a new API or project; only just what you need to "make it work". The book is divided into nine chapters, three appendices and has 178 pages.

The first chapter is short and describes installation and set up of Hibernate. The examples require the Ant, a build tool, in order to generate the executables. The examples are based on the Java based embeddable database HSQLDB. The author describes where to download Hibernate and install it within a project hierarchy.

After a short introduction into the world of object relational mapping O/RM, the second chapter starts working with Hibernate properly. We are already at page 13! The first topic is the mapping document. Hibernate frameworks require a XML mapping file to define meta data of plain old Java objects in terms of database persistence. The author demonstrates that mapping documents can be written quite practically with a text editor. Contrast this "mining at the coalface" approach in this notebook to the reference style of Bauer and King Hibernate in Action, and we are only at page 18 of the O'Reilly book. By the way the business domain of the book examples is a MP3 album / track information database, which works with Java based embedded database, HSQLDB.

The third chapter explains how to create persistent objects in the database using the Track POJO class, which was generated in the previous chapter. In the following sections of the chapter, the author carefully explains the life cycle of mapped objects, and how they move from transitive to persistent states. Moving in the reverse direction of state is about finding persistent objects. If you want a fuller understanding of Hibernate state management, then the Manning book is the perfect reference. Elliot describes the process of finding saved Track objects, the retrieval make use of the infamous HQL. In contrast to the Manning book, HQL is introduced early on to demonstrate how similar the query dialect is to ANSI SQL. Of course, HQL is semantically different from SQL. The last sections of the chapter tell how to delete a persistent object, and describe the named parameters of the HQL query engine.

A database that only could persist simple POJOS would be next to useless. Typically any data structure in an application required a type of collection to manipulate individual elements or to describe relationships between two or more different classes of objects. Chapter five is about mapping collection and declaring associations and it introduces a new class Artist. The only criticism of the this chapter is that it completes missing out or fails to explain that Hibernate can map complex Java collection data types such as bags, lists, and ordered types.

The fifth chapter expands on the previous one with richer associations, starting with lazy associations. The author justifies the reason for lazy loading of associations especially for linkages between big tables. There is a section dedicated to ordered collections. The associations between tables (Java classes) can be augmented as much you would expect in a business schema.

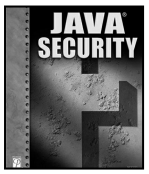
Chapter six describes how to persist enumerated types as Hibernate data types. In order to add persistence for enumerated data, then your POJO class must implement the PersistentEnum class.

Chapter seven is about custom value types, which are types that the developer defines and thus extends the default Hibernate data types. The author explains that complex and highly nested object structures can be persisted to a database using the framework.

Chapter eight discusses criteria queries with the major example to find MP3 tracks shorter than a specified length. Programming with the Criteria API is much like building functional objects and constructing them as an expression.

The subject of the final chapter is the Hibernate Query Language, which appear several times earlier in the book.

In conclusion, HADN is a great book to have on the coffee table, if you want are in hurry to get a project up and running with Hibernate. There are enough working examples for the developer to build functionally software without taking shortcuts. The only criticism is that such a book will not teach you advanced tips and tricks, so you may grow out of it pretty quickly if you suddenly become a seasoned Hibernate Expert. This latter important point will be sorely important if you need to optimise Hibernate for an enterprise quality application. None the less on average you will be sort of person who just wants to see best practice and actually view some code that just works as it says on the tin. This book is recommended.



Java Security by Harpreet Ganguli with NIIT (1-931841-85-3) Premier Press @ £37.99
reviewed by Alistair McDonald

This book is structured in three parts : an Introduction to Java Security, Advanced Security Concepts, and Appendices.

I found this book very difficult to read. The writing is often confused and concepts are rarely explained well. Within a chapter, there is a tendency to jump around between different areas. I got the feeling that the author did not know the subject well, and was occasionally confused by similar terms.

On the other hand, this book does not suffer from the overly verbose code listings of Helton's "Java Security Solutions", in fact the examples in this book work well in highlighting the discussion in the text. Most chapters have short question and answer sections.

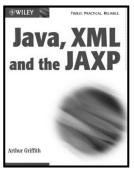
This book is not as comprehensive as Helton's "Java Security Solutions". It does not cover the more advanced topics, nor does it cover the basics as well as Helton's book. I would recommend Helton's book over this one.

Java Security Solutions by Rich & Johennie Helton (0-7645-4928-6) Wiley @ £38.99
reviewed by Alistair McDonald

This is a thick book, at almost 700 pages. It is broken into nine sections, each a few chapters long. One feature with nearly all of the chapters is the excess of code listings. These days, any good book should have their code samples available for download and there is little need to clutter up a book with page after page of listing. It is far better to discuss the salient points with short code excerpts than to reproduce entire classes, complete with verbose comments.

The book covers most aspects of security, in particular those that use cryptography in some way. In general, the writing covers Java well, but the book also attempts to touch on more general material, and it makes too many assumptions to be of any use.

Overall, this book attempts to do too much. Rather than being a Java security book, and covering security as used in Java applications, it is attempting to be a security book that uses Java to illustrate the concepts. Helton would do better to cut out a lot of the code samples, drop the more esoteric subjects, and pull the remainder into a more cohesive text. However, for someone who knows security basics, or has a more general security text to hand, then this book will be of great help if they need assistance in implementing security in Java.



Java, XML, and JAXP by Arthur Griffith (0-471-20907-4) Wiley @ £22.50
reviewed by R.D. Hughes

For a relatively small book, this has quite large ambitions – the author states that his intention is to provide an introduction to XML and to using XML in Java. In fact, the author has done a very good job in meeting these ambitions in a manner that is extremely succinct yet still very readable.

Other books of comparable size dedicate themselves entirely to introducing the syntax and semantics of XML. Here the author spends only the second chapter, some thirty pages, tackling these issues and yet I came away with at least as much understanding of the basics from this book as I have from others.

The rest of the book is largely dedicated to introducing DOM and SAX programming in Java. It succeeds in providing an excellent overview of these areas and enables anyone with even basic Java experience to get going with XML extremely quickly. A wide range of topics is covered; from basic parsing of documents to modifying DOM parse trees and building parse trees from scratch. Another chapter looks at XSL within Java and shows some very simple applications.

The author even finds time at the end to discuss Ant, an XML based build system designed for Java applications (although this is far from its exclusive use), as a practical and highly relevant way of showing XML in action.

I would certainly recommend this book for anyone who wants a good introduction to XML and its use within Java code.

Python

Python Programming for the Absolute Beginner by Michael Dawson (1-59200-073-8) Premier Press @ £19.99
reviewed by Alan Griffiths

As I've been programming in various languages for over thirty years I don't fit the target profile of an "Absolute Beginner", but I do have two children that do: Blake and Simon assisted me in reviewing this book.

For this purpose the underlying structure of the book is ideal: it is written around a series of increasingly complex computer games – something the boys could relate to immediately. And the first program in the book is a welcome variation on the ubiquitous "Hello World!" – it prints that essential element of a real game – the words "Game Over!"

Blake found that there was a little too much background discussion in the book to retain his attention and I spent some time with him to assist in skipping from one example or exercise to the next. This isn't really a problem with the book, as I do not think it is targeted at his age group (Blake is 9). In contrast, I got up one morning to find that Simon (12) had read the next chapter by himself and was enjoying working through the "challenges" at the end of it.

The book follows a sensible progression through text output, text input, variables, flow control, functions, object classes and instances, packages, GUIs and graphics each with a well chosen selection illustrative games and challenges.

There are some limitations: the approach is informal – so I was often left wondering exactly what the language supports for various constructs. While the book suggests trying things out – which is what the boys were happiest doing, I have a professional's awareness of the distinction between what happens with the current configuration and that which can be relied upon. This was most apparent when a "modified version of livewires" – an add-in library – is introduced: nowhere is it explained what the modifications are or what they affect. (I ended up replacing the latest versions from the Internet with the versions shipped on CD with the book as the easiest way to complete the later chapters – in part because not all the add-ins were available for the latest Python version.)

In summary, the book is a well thought out introduction to programming in Python. It is pitched at a level of experience below that of the typical ACCU member – but that is exactly what it claims to be.

Recommended.

General

Data Structures for Game Programmers by Ron Penton (1-931841-94-2) Premier Press @ £36.99
reviewed by Alan Lenton

The information on the back of the book'User Level: Beginning/Advanced' says it all. This book mixes the ludicrously simple with advanced topics and little in between.

The author also suffers from confusion about whether he is writing for game library authors or game application programmers, so many of the topics contain details of data structures without any information about how to use them efficiently. For instance, there is little point in covering minmax trees without covering alpha-beta pruning or other higher-level heuristics. Minmax trees are of no use on their own except for extremely trivial games, like noughts and crosses.

The author's decision not to use the Standard Template Library has only the weakest justification, and much of the C++ programming is also fairly primitive. I get the impression that the author had only just moved from C to C++ when he wrote the book. That is unfortunate, because the lack of fluency in the C++ idiom will not help his readers.

I cannot, unfortunately, recommend this book, and that is rather sad because there is potential showing through, especially in the

material on trees, graphs and finite state machines. It is just that they do not really provide enough depth to be really useful.

Not recommended.

Designing for the User with OVID by Dave Roberts et al. (1-57870-101-5) MTP @ £30.99
reviewed by Andrew Marlow

This book talks about UI, OO and basic software engineering principles. It does so in a way that makes it seem to be aimed at the beginner. At an elementary level and superficially, it seems to do this quite well. The table of contents is also impressive, making it seem like the book will pack a powerful punch in just a few pages. However, a more careful examination shows that the book has several faults, which is why it is not recommended.

The book is comprised of four parts: (1) UI and OO foundations, (2) the methodology, (3) prototyping and evaluation, (4) case study and exercises.

Much of the foundation section is standard good software engineering practise and is used whether or not OVID is used. Most of the discussion is clear and well presented but it does lack precision and is simplistic in a number of places. For example, the book does not make a clear distinction between classes and objects. The description of how to identify classes, objects and instances are closely tied with examples drawn from file metaphor applications. The section on views contains less than one full page of text on what an OVID view is.

The methodology section does not provide an overview of the OVID methodology. This, coupled with the way in which the foundation material is handled, makes it hard to distinguish between standard good practise and OVID. The finding views section is three pages and is the only part where there is any detail on views yet still there is insufficient detail to explain what a view is.

The discussions in the prototypes and usability section are basic but well presented.

The realCD case study has task and action tables forced onto single pages using microscopic print. They were almost unreadable. Screen shots are small, blurred and monochrome.

The book fails to attribute mental models to Donald Norman and does not provide enough references for hierarchical task analysis and Harel diagrams.

The book was published in 1998 and mentions a few things in such a way as to date the book. For example, throughout the book it mentions Windows 95 and 98 rather than just referring to Microsoft Windows(TM).

Many of the good points in this book do not owe anything to OVID in particular, and there are enough negative aspects about the book to outweigh the good points. In conclusion, this book is not recommended because in the opinion of this reviewer is simplistic, imprecise, dated, poorly presented typographically, and gives insufficient detail on what makes OVID different from other approaches.

Extreme Programming Explained by Kent Beck (0-321-27865-8) Addison-Wesley* @ £24.99
reviewed by Anthony Williams

This is Kent Beck's second attempt to explain eXtreme Programming to the world; it is not just a revised edition, rather a complete rewrite. He has a vision, a philosophy of how we should go about software development, which he describes in terms of values, principles and practices.

There is a new value, Respect, in addition to the values of Communication, Simplicity, Feedback and Courage from the first edition. These values, along with the underlying idea that we are trying to write software to please the customer, drive the principles and practices; Communication is important, so talk to your customers all the time, show them what you've done so far and discuss what can be changed. Feedback is important, so write tests for everything, so you know if you break something, work with the customer to write acceptance tests, so you know when you're done, and give the customer frequent releases, so they can try things out for themselves.

Many of these ideas are old, tried and tested ideas with a good track record, but the combination makes them even more powerful, and it is called "Extreme" programming for a reason – Kent wants to drive the ideas as far as they can go, "turn the dials up to 10"; if testing is good, let's test all the time, after every tiny change; if code reviews are good, let's program in pairs and do continuous code reviews as the code is written. "10" is further up on some of the dials than even Kent thought it was when he wrote the first edition; people release more often, and integrate more often than the first edition suggested, for example.

Kent acknowledges that one of the criticisms of the first edition was that it was too inflexible – you were only "doing XP" if you did every practice, followed every principle and held every value. The second edition is far more inclusive – he acknowledges that people may have other values in addition to those listed, and that these will shape the principles and practices. He also acknowledges that each team needs to decide on the set of practices that is most appropriate for their situation; that said, many of the practices will help to improve the software development process of any team, regardless of whether they decide to "follow XP".

This book is very easy to read; the writing is compelling, and it is well laid out. That said, I'm still not sure that it can be comfortably read in one pass; Kent's ideas are deep and far-reaching, and really need thought and contemplation to get the most from them. I expect to gain more insight into what Kent is getting at with each reading.

You might not agree with what Kent says, but this book will certainly make you think about the way you develop software. It is not just aimed at developers, but their managers and executives too – developing software XP style can have beneficial implications for the whole organisation.

Highly recommended.

Game Programming All on One by J. Harbour (1-59200-383-4) Thomson @ £36.99
reviewed by Paul F. Johnson

I wanted to like this book as it combines what I really like in a book; platform independent library, support for many compilers and a non-partisan approach to development.

Very quickly though this falls apart. Dev-C++ is used throughout, headers are missing and there are no instructions for compilation from the command line (many linux programmers use the command line over an IDE for small projects).

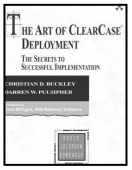
This means that while there are plenty of examples, they will not compile cleanly (if at all) and, given the intended audience of Beginner – Intermediate, this is plainly unforgivable.

Confidence isn't very high of the true cross platform nature of the book with the early examples – conio.h doesn't exist on my machines, the likes of time() are used without time.h being included. Okay, let us move onto the program examples.

There is very little in the description of the library functions in use or what the sections of the code actually do. The book also has full listings of the source contained on the CD. Now, I am a tad lazy when it comes to typing in code, I use the CD. Lump it into kdevelop and hit compile. Then puzzle why it will not compile. Check the makefile. Right, the makefile is broken – as they are for quite a number of the source files. Putting my "beginners" hat on, first thing I do – give up. I do not have the knowledge to fix that sort of problem. Chase through the book and see if I can compile via the command line. There are a couple of instructions, but they are incomplete.

This really is a pity – the book could have been so much more. Instead we have a second rate book which for the beginner will cause many difficulties further down the line.

There seems to be something in common here. I've looked back at my past reviews of books and the majority of the not recommended books have all been from the same publisher. It really does look to me as if there is very little in the way of technical editing or quality control. I hope this improves soon.



The Art of ClearCase Deployment by Christian D. Buckley & Darren W. Pulsipher (0-321-26220-4) Addison-Wesley* @ £30.99
reviewed by Pete Goodliffe

ClearCase is a high-end expensive version control and configuration management tool sold by Rational/IBM. The aim of this book is to explain how to set up and administer a ClearCase installation, and how to integrate it into your company's development process. In a nutshell: it is a disappointing offering.

The book certainly readable, if verbose in places. It is clearly laid out. If your organisation is large and has very chaotic CM and development processes it might be of some use.

And yet it is blinkered and presumes ClearCase is right for you – it does not help you to choose the right tool. There is a small comparison with other CM tools, but only really to slate them. It is biased and sometimes quite incorrect, dismissing the alternatives.

This book gives the overall impression of a work not quite finished yet, with countless errors and omissions. There are even wholesale duplications – a sure mistake. There are large structural problems, and the book opens and closes with some very odd "change agent" spiel that does not gel with the rest of the contents.

ClearCase is a good system, and will be a great asset to a large organisation: when deployed correctly. Many organisations can't support such a behemoth, and blindly moving to a ClearCase system could be very destructive. This book will not help you to understand these issues.

The Data Compression Book by Mark Nelson & Jean-Loup Gailly (1558514341) MTP, 526pp 1996 @ £36.99

reviewed by Paul Colin Gloster

This is a book which accessibly reviews the prerequisite theories underlying most conventional forms of compression and which quickly and intelligibly progresses onto fully working practical programs. It does not concern itself with the file formats of established, refined compression file formats (not even those used in the widely distributed free source programs written by one of the co-authors) and as just the basic core concepts appear, the programs in this book are comparable to but do not rival third party compression programs: not even matching the performance of programs by Jean-Loup Gailly on mirrors of GNU-licensed FTP servers.

However, I would not hesitate in recommending this or a similar book because it is good to read such a clear domain-specific text explaining many of the main approaches taken in both lossless and lossy compression algorithms.

Even so, the code printed in the book has mistakes, though fortunately the accompanying disk is perfect. The code presented is very portable but I have found that due to varying numerical precision, the lossy programs are not binary compatible across platforms.

Desktop computing power was an important factor for lossless coding when this book had been written, but now it is feasible to use lossless compression on embedded targets, which is perhaps where some people will want to deploy the disk's code as opposed to merely understanding the science and moving on to superior industrial strength compression.

Unfortunately, not all embedded C compilers whose vendors lie about C89 conformance are able to use a complete program unmodified.

Being from the then-publisher of "Dr. Dobb's Journal", the book contains an essay against the patenting of innovations for which the patent applicants played no role in discovering.

Linux

Linux Kernel Development by Robert Love (0-672-32720-1) SAMS, 496pp £32.99 Dec 2003

reviewed by Ian Bruntlett

This is, in my opinion, a "wow" book, detailing the Linux Kernel 2.6 as it applies to 80x86 hardware. It has a website `http://tech9.net/rml/kernel_book`. At the time of writing, Kernel 2.6 is the current kernel it is unusual to have up to date kernel documentation. I heard a rumour that the latest Linux Device Drivers (3e, O'Reilly) also covers Kernel 2.6. If this is true then we are experiencing a rare renaissance in Linux Kernel documentation.

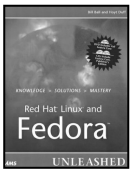
The author is an experienced kernel and GNOME hacker who now works as senior kernel engineer in the Ximian Desktop Group at Novell.

Prospective readers should be familiar with C, processor architecture and operating system design – if you are a bit rusty, check out the bibliography or, for processor specific information, using Google can unearth some very interesting processor datasheets.

This book aims to equip the reader with the skills necessary to become a Linux Kernel Developer. One item on boot loaders, on page 16, discusses LILO instead of the more recent GNU GRUB. The special requirements introduced by kernel mode development are discussed.

In some way this book is like the QL Advanced User Guide – I didn't understand it completely on my first reading although with time and effort I'm likely to understand it.

Highly recommended for: 1) People who want to know how their system works; 2) People who want to write kernel/supervisor mode code. If you are only going to use the standard C library (libc) then you are better visiting the Linux Emporium website and buying the GNU libc manuals.



Red Hat Linux Fedora Unleashed by Bill Ball & Hoyt Duff (0-672-32629-9) SAMS* @ £36.50

reviewed by Silvia de Beer

I have mixed feelings about this book. The book consists of four parts. The first part covers how to install Fedora Linux on a PC (previously called the Red Hat distribution). It explains what to pay attention to and investigate before installing Fedora, and how to make a dual boot machine. I found this first section useful to read. The second part treats System Administration, and contains chapters on services, package management with rpm, managing users, managing the file system, backing up. The third part treats system services administration, and the last part is about programming and productivity.

Most of the topics would need to be mastered by a system administrator, so in that point the book would be excellent, but because the topics are all covered with such little detail, the book is almost worthless. It could for example be perfectly useful to read a little bit about configuring printing services, network connectivity etc, but I think the book should have been more exhaustive on fewer topics. The book is overly verbose and I did not learn very much by reading the 1000 pages.

I did miss topics that I think would need to be covered in this type of book on Linux. All the command line commands are given without any further explanation and options. It will be rare that you can use the command in the exact format that it is given. For example, I looked in the index to find the sections in the book which cover basic Linux commands like ps, ls, grep, but they are all just given without any explanation about the rich wealth of options.

The book comes with 2 CDs and a DVD, containing the Fedora Core distribution. This is may be one of the reasons that you could consider buying the book, because downloading such a distribution over a normal telephone line is burdensome.

Management

Managing Software Acquisition by B. Craig Meyers & Patricia Oberndorf (0-201-70454-4) Addison-Wesley @ 37.99

reviewed by Mark Easterbrook

My first impression of this book is that someone had selected as many buzzwords as they could and then strung them together with padding to make grammatically correct, with the result that a lot of the book is indigestible. The text is full of paragraph after paragraph starting with "A manager may say" or "A systems consultant may promise" and numerous anonymous quotes, resulting in a book that feels like it is a collection of third party hearsay – as if the authors are reluctant to present any ideas of their own.

The authors claim an audience for the book of project managers and their staff in both private and public sectors, but there is very little that is not aimed squarely at those working on US government contracts, of which much is military related. As a result, this book is of little use to anyone who does not work in this specific market segment.

As part of the SEI series in Software Engineering I expected a reasonable level of quality, applicability, and usefulness, but I have been disappointed, Addison-Wesley have devalued the whole series by letting this one slip through their vetting process. Not recommended.

Managing Software for Growth by Roy Miller (0-321-11743-3) Addison-Wesley @ £26.99

reviewed by Mark Easterbrook

The overall theme of this book is that software is grown, rather than manufactured, which is why attempts to manage or control software projects using traditional management or manufacturing techniques have been generally less than successful. The book is divided into four parts. The first part explores applying traditional and contemporary software management techniques to software projects and why they so often fail.

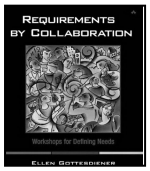
The second part moves away from software and explains the science of complexity such a chaos theory as a background to the third part,

growing software, or how to plan and control something as complex as software development that tends to defy planning and control. The final and largest part explores how managers can change their organisation from the unreliable manufacturing of software to a more successful growing of software.

The book is unfortunately let down by the choice of language which is distinctly American and makes frequent use of U.S. idioms and culture, which reduces its potential audience and may make it difficult for readers who do not have English as their first language. If you have to stop and think what IRS stands for or the significance of Apple Pie comparisons, then some parts will be difficult to comprehend.

This is not a book that introduces any radically new ideas but borrows extensively from previous works, with reference to classics such as DeMarco & Listers "Peopleware" right up to the latest on Agile Development. This, combined with an Annotated Bibliography at the end, makes it a good starting point on the wrongs and rights of software management, although the bibliography is far from comprehensive and I would have liked to see books such as "The Mythical Man Month", "Crossing the Chasm" and some of the Microsoft Press publications mentioned.

Although the book does have its merits, and deserves a place on a recommended reading list on software management, the language and lack of original material would place it closer to the bottom rather than the top.



Requirements by Collaboration by Ellen Gottesdiener (0-201-78606-0) Addison-Wesley @ £34.99
reviewed by Ivan Uemlianin

Requirements By Collaboration is a very practical book about planning, conducting and evaluating same-time/same-place requirements workshops. It will work very well as a handbook; less well but still useful as an introduction. I recommend the book for anyone running such workshops.

There are three parts: Overview (chapters 1-3), Framework (chapters 4-9) and Design Strategies (chapters 10-12).

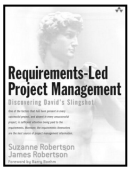
In the Overview, Chapter 2, "Workshop Deliverables: Mining Coal, Extracting Diamonds", looks in fairly practical detail at Requirements Models. Nineteen models are outlined (e.g., Actor Map, Business Rules, Event Table, Use Cases) and characterised in terms of how they address the problem domain. Chapter 3 goes on to give some useful "Ingredients of a Successful Requirements Workshop".

Framework consists of six chapters which each look at a different aspect of the workshop: the six P's – Purpose, Participants, Principles, Products, Place, and Process. For example, chapter 9, "Process: Plan the Work, Work the Plan" looks at different activities, the focus being (a) appropriately linking activities to types of requirements model and (b) forging a fruitful path through a workshop from one activity/model to the next.

Design Strategies steps back from the chalk face to look at case studies and presenting results.

The book is rigidly structured and its language is highly rhetorical (from the punchy end of the advertising spectrum). This is all a bit heavy going for a cover-to-cover read, but the rhetoric is not empty: it allows the author to be very terse when she needs to be.

Requirements (in part 1) and workshops (in part 2) are each approached from many different perspectives. A complex, multi-faceted picture is built up which might be difficult to translate into a more conventional linear-through-time conception. However, used as a handbook for planning a workshop from your initial questions, through kitting out the room, to running activities on the day, this book will help you develop intense, dynamic and stimulating (and probably reusable) requirements workshops.



Requirements-Led Project Management by Suzanne and James Robertson (0-321-18062-3) Addison-Wesley @ £34.99
reviewed by Ivan Uemlianin

"Requirements-Led Project Management" is aimed at people familiar with the requirements process; it claims (on the back cover) to "demonstrate" how good requirements can impact positively on processes further down the development lifecycle.

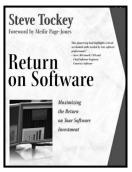
The book consists of eleven chapters, each discussing a different aspect of the requirements process, and how it can improve the effectiveness of your project management. Chapter summaries are provided in "What Do I Do Right Now?" and "What's the Least I Can Get Away With?" sections. Two appendices give outlines of a "Requirements Knowledge Model" and the "Volere Requirements Specification Template".

The back cover says the book shows "how to use requirements to manage the development lifecycle" and I interpreted this to mean that it would be shown how aspects of requirements process could improve other parts of project management. This holds true for at least the first three chapters, but for the rest of the book the link is much less clear. For example, the chapter on Measuring Requirements does not show how good requirements can mean more accurate function point counts and consequently more accurate labour-time estimates; it shows how good function point counts can lead to accurate labour-time estimates for the requirements gathering process.

Much of the book is an ostensibly pleasant meander through the requirements process. The table of contents does not provide subheadings and the book has not been structured to promote efficient navigation. My impression is that the authors (and editors) were very relaxed throughout its production. I have not read the Robertsons' Mastering the Requirements Process, which may well be seminal, but if it is they are resting on their laurels.

There are good ideas dotted through the book, but they are all too fleeting when they appear. For example, a few interesting pages on requirements reuse (p38f), could have been turned into a very useful chapter.

This is not a first requirements text; neither can it be used as a handbook. It is not an especially pleasant read. If you liked Mastering the Requirements Process, you may like this one.



Return on Software by Steve Tockey (0-321-22875-8) Addison-Wesley @ £37.99
reviewed by Alan Griffiths

This book sets out to tackle a serious subject and one that could well do with treatment: how to address the technical choices that software development presents within the context of a business. I was hoping that the book would present techniques for translating between the world of scope, development effort, timescales and alternative technologies and that of business value, cash flows and return on investment. I was very disappointed.

It starts well with a discussion of the reasons why forces from the business are important to software developers. But, having once explained why the financial view of software development projects is important, the book focuses entirely on that view – for example, there are several chapters on methods of comparing different cashflows. (Which, to me, was tedious – given my training in mathematics, these different methods would obviously give the same result as the difference was usually a scale factor – this scale factor reflected the interest that money would accumulate between different reference dates).

For most of the book, there is no reference to software development – and especially not how one might present a choice between, say, using .NET or Java as a cash flow. The closest one gets is the use of purchasing computers in various examples, for instance why tax law requires such a purchase to be depreciated over several years.

Not recommended.