Contents

Copy Dates

C Vu 17.5: September 1st 2005

C Vu 17.6: November 1st 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

Paul F. Johnson <cvu@accu.org>

According to my C Vu folder, this is my 8th edition at the helm. It's been a gas and has taught me a lot about deadlines and what the word "professionalism" actually means. Sure, we can all hold our hands up and say, "I am a professional", but as Pete Goodliffe points out issue upon issue, there is more to being a professional than a trade or a methodology. It is more a state of mind over anything else. Let me explain what I mean.

While in our everyday life we strive to produce the best possible product (whether that is documentation, code or design) we can, if we are not in the correct frame of mind, then it is unlikely that irrespective of how amazingly good you are that you will turn out your best – I'm sure everyone has woken up midweek for the Lurgi to hit; you feel dead on your feet and the prospect of having to battle against your project fills you with fear and loathing. Sure, you still go into work, but without the correct mental attitude, you might as well not go in.

Professionalism is Gestalt. Everything has to come together; mind and body. I've known people with quite a low level of programming ability but do have a professional mental attitude which is more important in some respects. When you improve your programming/design/documentation/whatever what is it that is actually improving? The entry of code will certainly be more rapid and you may have adopted a more agile method in your project, but surely the two most changed aspects are knowledge and attitude. We've all done it, learned a new skill (such as an optimisation technique or a generic method of template execution) and felt that buzz when applying it. This is the "professionalism buzz" - and it reflects on your day to day operation. Really. I'm not joking – it can be seen and reflects upon most aspects of any job you do.

I've seen this most recently on the ACCU mailing list where a proposed new mentored developers group for MUD (multi-user dungeons) programming was brought up. There was an initial sustained burst of energy while the idea was discussed and then slowly and surely as the realisation dawned on the proposer of some of the (not to insubstantial) problems, things slowed down and an analysis began. It was interesting for me to view. Not so much as the editor of C Vu, but as someone who has never really been involved in design. Okay, I know the programming side: the fun that can happen with asynchronous servers over synchronous servers, how best to preserve data on crashes and fun with threading models, but not the design side.

Those with knowledge and understanding of the subject injected a needed quantity of reality. They didn't kill the idea, but through their professionalism in dealing with people they have probably never met, moved the project past the lots of talk and no action stage to something which can be considered to be approaching a real project.

You Don't Know What You Have Until It Goes Phnutt!

Unlike previous issues, this one has been quite problematic from my point of view. My email went phnutt! Try going without any access to your most frequently used email account for a week and see how much you miss out on!

The problem occurred when the hosting service I used had problems with the hosting service they use. Essentially, just about all of the essential services (web, ftp and email) died on a Sunday and the backbone refused to either restart the server or trace the problem and report back to the sys admin (me). The people I use for hosting moved server on the Tuesday and my email was finally sorted out by the Friday. Five days without email coming up to the submission date is not to be taken lightly!

Thankfully, all is well again.

It Should Be Banned!

I'm sure everyone has, at one time or another, decided that they no longer wish to work for their current employers. After gaining my PGCE (in education, certificates are everything!), I decided that it was time to move on. "I shouldn't have much of a problem" thinks I, "after all, with my experience and degree in Chemistry, it should be a doddle". Like hell it is! While I am very good at what I do and somewhat competent within programming, I am completely unable to get a job away from where I currently am.

I have two things going against me

1. I have only ever worked in education. Arnold Rimmer (of Red Dwarf fame) had a large H on his forehead to signify that he was a hologram. Anyone who works in education, at any level, automatically have a large E on their forehead which only prospective employers can see.

2. There is one question on every application form which I can never answer. It is that "Please provide additional evidence in support of your application". This can be viewed in one of two ways. Either as your attempt to fib your way into an interview or to sell yourself. I am hopeless at both, so I rarely get through to an interview.

While I can understand the need for this section on an application form, unless you're working in sales, it's pointless! The application form, employment criteria and quality of covering letter should suffice. If the three are met, then the short list can be drawn up and the interviews go ahead.

Problem is that it is quite likely that a lot of people would qualify for an interview and what would be the fairest way to see if someone was suitable? Yes, you guessed it - "provide additional evidence in support of your application". Argh! I think my problem is that I'm just a nice chap. Dang. I need to be a ruthless fiend able to smite other CV's at 50 paces and remove from the space-time continuum those who would stand against me. But where is the fun in that? Oh well. Another day, another application form done. Pity I never studied Chemistry as there is a shortage of qualified teachers. Hold on.... I did study Chemistry...

An Apology

"Everyone makes mistakes". It seems in the last edition's Book Review section an incorrect price crept in. The book in question is "C++ Common Knowledge: Essential Intermediate Programming by Stephen Dewhurst". The review had the book listed at £29.99. While this is the price on Amazon (prior to their discount), the actual price according to the publisher (and backed up on the publishers website) is £21.99. I'm sorry for this mistake and any problems it may have caused.

A New Service from C Vu

One aspect you may have noticed has been apparent over my time as editor has been the promotion of both best practise and education. After consideration, I've decided to add a new semi-regular feature which I'm sure you'll all appreciate.

If you know of a conference (or are running one), then C Vu will publicise it for you free of charge within C Vu and what makes it even better is that it's completely free for what I'm offering.

How to Submit a Conference.

Send the details via email to me in the following format and in it shall go. All I ask is that a short report of the conference is submitted for future editions (or for the website).

Submissions should be of the form:

- Name of conference
- Date, Time and Location
- Costs
- Email contact / Web address
- Synopsis (max 300 words)

Simple as that. I'll run the conference call for 2 editions prior to the conference.

Interested? Email me.

Okay, let's get on with it then

Enough of me – let's get on with the magazine!

Paul F Johnson

View From the Chair

Ewan Milne `<chair@accu.org>`

Summer finally seems to have arrived, you probably don't want to think about winter, but it's never too early to think about next spring: and of course with spring comes the ACCU Conference...

ACCU Conference 2006: Call for Participation

The ACCU Conference 2006 will take place in Oxford on the 19th-22nd April. We would like to invite you to lead a session at this leading software development conference. Presenting a session is a highly rewarding experience: the quality of our conference audience means that you will come away from it having learned as much as anyone in the room.

Sessions are 45 or 90 minutes long (see below), and may be either tutorial-based, presentations of case studies, or take the form of interactive workshops. We have a long tradition of high quality sessions covering many technical aspects of software development. We are particularly interested in sessions covering any aspect of C++, Java, C# and Python - from beginner's to advanced level. Other technologies and issues including (but not limited to) XML and scalability are also welcome.

ACCU is keen to include sessions about the wider development environment beyond languages and technologies and facilitate dialogue between developers, analysts, planners and managers. To this end we would encourage speakers who wish to address the subjects of:

- development process
- design
- analysis
- patterns and softer aspects such as team building
- communication and leadership

We are particularly interested in the issues around distributed collaboration: tools and techniques, which will be the subject of a full day track.

If you would like to run a session please let us know on `accu2006@accu.org` by the 30th September at the latest.

Please include the following to support your proposal:

- Title (a working title if necessary)
- Duration (45/90 min)
- Speaker name(s)
- Speaker biography (max 150 words)
- Description (approx 250 words)
- Intended audience

If you are interested in knowing more about the format and style of the sessions you may like to consult the website for previous years' conferences at: `www.accu.org/conference/` for background information.

Speakers running one or more full 90 minute sessions receive a special conference attendance package including free attendance, and assistance with their travel and accommodation costs. In addition to 90 minute sessions, we are running 45 minute sessions, specially designed for new speakers. A shorter time slot may offer the opportunity you need to present your first session. Speakers filling a 45 minute slot qualify for free conference attendance on the day of their session.

The conference has always benefited from the strength of it's programme, making it the highlight of the year for many ACCU members and other attendees. Please help us make 2006 another successful event.

Ewan Milne

Membership Report

David Hodge `<membership@accu.org>`

We currently have just over 1000 members in 42 countries. Those outside the UK with more than 10 members are USA, Germany, The Netherlands, Denmark, Sweden, Australia and Switzerland.Renewal time is upon us for the majority of members, Please get your renewal completed before the end of August, it saves me a lot of time.

Again a reminder to keep me informed of changes in mail address (for the journals) and email address (so we can contact you). Please note that the number of times we use your email address to contact you is very low, certainly not more than 10 times in a year.

David Hodge

Secretary's Report

Alan Bellingham `<alanb@episys.com>`

The first meeting of a new committee year usually has some additional work to do, over and above the normal business. Quite how much work there is does, though, depend on the outcome of the AGM, especially in the matter of actual committee members.

The first committee meeting of the new year took place on the 21st May 2005 in Royston, Hertfordshire (at the home of your secretary). It started at 13:45, a little later than usual, after a good pre-meeting chat at the local pub that went on a little longer than planned. Six people were there, a little fewer than desired though still quorate.

As per normal, the first action was to go through the minutes from the previous meeting and see which items were completed, and which were not. (When fewer members are present, it is a little more difficult to actually determine which items have been completed - and a number of items in this case were left as unsure.)

Following the minutes, the next stage is the reports. For the most part, I won't be repeating the content of the reports, since they will be very similar to the C Vu reports closest to the meeting date. (But if you want to hear the latest reports before they reach C Vu, then come to a committee meeting.)

After the reports, the first piece of 'real' business was the co-option of certain members. In this case, it should be no surprise to our long term membership that Lois Goldthwaite, Silas Brown and Reg Charney were co-opted again, into the same roles as last time. And those that have been paying close attention at the AGM should also have expected the co-option of Allan Kelly as a general member of the committee without a specific portfolio.

(The result of this is to leave us with a slightly smaller body than last year, though not as small as some people's ideal of a committee of one.)

After this, the major business of the meeting started, with the report on the progress on the new website. A full report is promised for elsewhere in this issue, and that report will be somewhat more up-to-date than the one we received. But the tenor of the report was that the work is well in hand, though not yet ready to go live. Since we have a sub-committee dealing with the work, and since the AGM had elected a new committee (albeit with much the same members), the sub-committee also needed and received reconfirmation.

Another matter that has been providing much work for some of us over the past months and years has been the webification of back issues of these journals. The ideal has been to try to get all issues into an XML format from which web pages may be generated, indices drawn up, and so forth. With recent issues, this hasn't been too hard, but early issues have been more of a problem, many of them requiring scanning, OCRing and correcting. In the end, the decision was been made for the earliest issues to be provided in PDF format, rather than trying to impose structure this long after the fact.

After that had been decided, we then turned to the issue of the post of Treasurer. As those who attended the AGM will be aware, Stewart Brodie has felt himself unable to fully fulfill his duties as Treasurer, most especially in the role of preparing the accounts for auditing, and he tendered a conditional resignation shortly before the AGM. Since we did not find a replacement at the AGM, we declined the resignation pro-tem.

The meeting considered the problems, and actually came to the interesting conclusion that the major hassle that Stewart has been encountering - that of preparing for the audit - is actually unnecessary in an organisation like ours. Yes, there are requirements, but no, they do not actually need to devolve on the person of the Treasurer and, so long as receipts and the like are properly recorded, a professional bookkeeper may be hired to prepare the accounts. Once an independent bookkeeper is involved, professional auditing is no longer required.

At the end of the meeting, Jez Higgins noted that the sub-committees that do certain jobs - for example, the conference organisation, and the web-site work - were not feeding back to the committee as much as they could. This isn't because they're trying to be secretive, but more the programmer's tendency to 'just get on with the work' without getting bogged down with management meetings. We felt that yes, just as this report is now trying to tell the membership in general more, so should the sub-committees provide better feedback to the committee.

Finally, the meeting broke up at 16:40.

Provisional future meeting dates:

- Sep 17th
- Nov 19th
- Feb 18th
- May 20th

Alan Bellingham

Officer Without Portfolio

Allan Kelly <allan@allankelly.net>

The last time Paul called for officers reports I tried to sneak out the back door, he caught me and forced me to write something. I'd like to sneak out the back door now but I know what would happen...

The reason I feel like sneaking out is because I've spent ACCU money, I've claimed we have a new website and yet few members can see it. Well, you don't have to take my word for it any longer... Tony Barrett-Powell will also tell you so.

Tony has stepped forward to take on the position of website Editor he is in the process of reviewing and updating the website content and building a team to help him keep the website up to date. If you would like to get involved please e-mail Tony, I'm sure he'd be grateful of your help.

Still, at the moment the old website is the ACCU's website. I'm sorry its taking so long to get the new one up and running, things just take time when your squeezing them in between real jobs.

Fact is, the content was quite out of date and needed a lot of work. There was a debate in the ACCU a couple of years ago, some people said "Changing the technology without changing the content is pointless" and others said "Changing the content without updating the technology is impossible." Well the debate went around and around, finally we broke the circle. Now we have new technology it is clear we need new content.

There is more development behind the scenes too. I'm getting a specification ready for phase 2 of the website, this will see Mentored Developers, the Journal Archives and ACCU-USA move across.

In addition the New Web committee has been in discussions about a new book review database. The regular ACCU committee meeting approved in principle a proposal that would see the development of a new book review database this will provide links to booksellers online systems. The new database would be paid for from the proceeds of the partner programme. Final details and technicalities have yet to be sorted out but I hope this system will be live before too long.

Finally, I've been a member of the ACCU for over seven years now. Its great, I love it, I'll never leave. In the past my contribution has been through the journals but I joined the committee in 2004 to see the website revamped. I can see the end of this process now, hopefully before Christmas. Therefore I do not intend to stay on the ACCU committee when the AGM comes along in April 2006. My work will be done.

Who knows, I may even find time to write for Overload again.

Allan Kelly

Standards Report

Lois Goldthwaite <standards@accu.org>

The international C and C++ committees will be meeting a bit earlier than usual this fall, starting September 25 in Mont Tremblant, Canada. This is because the Standards Council of Canada is combining the two meetings with another one, the annual SC22 plenary, which is traditionally held in August or September. SC22 is the ISO/IEC committee which is the parent of the two working groups and several other WGs.

Because of some schedule-juggling, the C committee is scheduled to meet only four days instead of five, but the C++ committee has acquired an extra full day for meetings. That's good because there is plenty of work to occupy us - this meeting is the cutoff deadline for any new proposals to be considered for inclusion in C++0x. If you want to keep a weather eye on what's on the horizon for C++, submitted papers can be downloaded from the WG21 committee web site at `www.open-std.org/jtc1/sc22/wg21`.

The agenda for the C committee includes further discussion of its Technical Report on Some Additional Library Functions Which May Help Careful Programmers Avoid Buffer Overruns and Assorted Nasty Code Glitches. Well, that's not exactly its real title, but it will do until the committee can decide on a new one. First it was called Specification for Secure C Library Functions, but a number of countries, including the UK, objected that wording implied a promise to deliver more 'security' than was actually possible from the rather modest contents of the library. (In a nutshell, the new functions are variations on existing library functions, with some extra parameters added to support bounds-checking against buffer overruns.)

So the name was changed to Specification for Safer C Library Functions and everyone was satisfied. Until it was pointed out that 'Safer C' is a trademark in the UK belonging to Les Hatton, a member of the BSI C panel and well-known speaker at ACCU conferences. (And if you're not familiar with his book, Safer C, it's worth a read.) A new title is now being sought. We can boringly refer to it as PDTR 24731 in the meanwhile.

If you'd like to suggest a more descriptive name, to comment on some proposal before the working groups, or to join the C or C++ panel and enjoy some stimulating technical discussions, please write to `standards@accu.org` for more information.

Lois Goldthwaite

Dialogue

Student Code Critique Competition 35

Set and collated by Roger Orr
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, as are possible samples.

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 the UK and Europe.

Student Code Critique 34 Entries

I send an unsigned char called tipus which is meant to be some hex number from 0x00 to 0xff and which should decide the string to be returned...
If the unsigned int called `valor` *I send is above 0xA000 I use the struct* `decidetest` *to send a string back, else, I send a string selected from the struct* `decide`*.*
If no substitution is made, then the string returned is always a space in html... `&nbsp;`
For some reason I always get the `static char escape` *string returned... any idea? (the function compiles all right...)*

```
char* Detect_type(unsigned char tipus,
unsigned int valor)
{
int i;
static char escape[16] = "&nbsp; ";
static struct decide {
unsigned num;
char *string;
} cadena [] = {
         {0x00, "Sobre V PK "},
         {0x02, "Sobre V RMS "},
         {0x0E, "——Power OFF--"},
         {0x10, "——Power ON——"},
         {0xff, " "},
};

static struct decidetest {
unsigned numero;
unsigned char num;
char *string;
} cadenatest [] = {
         {0x00, "Sobre V PK Test"},
         {0x02, "Sobre V RMS Test"},
         {0x0E, "——Power OFF--"},
         {0x10, "——Power ON——"},
         {0xff, " "},
};

if (valor >= 0xA000)
  {
    for(i = 0; i < sizeof(cadenatest) /
        sizeof(cadenatest[0]); i++)
    if(cadenatest[i].num == tipus)
      return cadenatest[i].string;
  }
else
  {
    for(i = 0; i < sizeof(cadena) /
        sizeof(cadena[0]); i++)
    if(cadena[i].num == tipus)
      return cadena[i].string;
  }

return (escape);
}
```

From Roger Leigh <rleigh@whinlatter.ukfsn.org>

The Initial Problem

We are told the function "compiles all right". This is not true:

```
$ gcc -c orig.c
orig.c: In function 'Detect_type':
orig.c:22: warning: initialization makes
integer from pointer without a cast
orig.c:22: error: initializer element is not
computable at load time
orig.c:22: error: (near initialization for
'cadenatest[0].num')
orig.c:22: error: initializer element is not
constant
orig.c:22: error: (near initialization for
'cadenatest[0]')
```

The following changes were made to correct various problems with the code. The code is compliant with the ISO C99 standard.

Looking at the source code, we see that `struct decidetest` has three members, of which only two are used. Comparing with `struct decide`, `numero` looks out of place. If this is removed, the function compiles, and appears to work as intended (I added a simple test program to test each case, including failure). It's always a good idea to do this when writing code: when making the following changes I could check for breakage after each change.

Cosmetic Problems

`Detect_type` is an unusual name. Think about why you are using a particular naming scheme. Common types are `CamelCaseNames` and `lower_case_underscored`. As an example, I usually use camel case for structure and class names, and lower case for all function names, methods and variables. The aim of the capitalisation and underscoring is to separate individual words to give readable and meaningful identifiers. Often there will be a common suffix within a particular program or library, a "namespace". I renamed the function to `detect_type`.

Your indentation was mostly acceptable, but indentation following curly brackets varied depending on whether they were enclosing the function body or any other use. I re-indented it using Emacs.

The identifiers used, `tipus` and `valor`, do not appear to have any meaningful bearing on their use within the function. It is important to use identifiers that convey meaning. I renamed `tipus` to `value`.

`valor` is an unsigned integer, yet is used in a Boolean manner. Quite what the special significance of 0xA000 is is not at all clear. This sort of undocumented magic inside functions will only cause maintenance problems later: keep this localised to where it actually means something. Presumably this has meaning elsewhere in your code, but here a simple true/false value is sufficient. I replaced `unsigned int valor` with `bool test`. The caller can simply use `(valor >= 0xA000)` to achieve the same behaviour.

`escape` is only used once. Why not eliminate it entirely?

The return value is in parentheses. `return` is a keyword, not a function, making this unnecessary.

struct decide

After removing `numero` from `struct decidetest`, `struct decide` and `struct decidetest` are now identical. There is no need to define two identical structures, so I removed `struct decidetest`.

The **`num`** member of **`struct decide`** is an **`unsigned int`**, but **`valor`** (now **`value`**) is an **`unsigned char`**. There's no need for the extra size, so I changed it to **`unsigned char`**. Because it's used as a value rather than a character, I then changed all the instances of **`unsigned char`** to **`uint8_t`** from **`<stdint.h>`**, to better reflect its use as a number rather than a character.

The "string" member of **`struct decide`** is not **`const`**, but it only ever contains string literals. The function also returns a non-**`const`** pointer to it. I made all character data **`const`**, and made the return type **`const`** as well. **`const`** correctness prevents accidental modifications, and therefore makes your code more robust.

The initialisers are quite simple, because there are only two members. You might want to consider using C99 named initialisers for bigger structures, which significantly improves readability and protects against accidentally missing out a member (as we saw in the case of **`struct decidetest`**).

The last member of both **`cadena`** and **`cadenatest`** is followed by a trailing comma. Some compilers don't like this.

Quite what the meaning of the "magic" constants 0x00, 0x02, 0x0E, 0x10 and 0xFF are is not apparent. If their only purpose is to be unique and in a certain order, using an **`enum`** would make the code cleaner and more maintainable. I haven't changed this because I can't be certain of breaking other code.

Strings

The string `escape` and most of the other string literals are of length 16. Is this a requirement? If so, the type of "string" in `struct decide` should also be `char` [16], and all of the strings starting with "—Power" are too short. If it's not a requirement, just make them all of type `const char *` (field widths may be specified with a `printf` format string, illustrated in the test code). I made this change, because hard-coded string length limitations are rarely used in modern code.

Eliminating Redundancy

The for loop that iterates through the various values is repeated twice for both `cadena` and `cadenatest`. This can be eliminated by using a simple pointer. This also required adding an extra `NULL` element to the end of each array. This also allowed `i` to be removed.

Amended Code

This is the result of making the above changes. I hope you can see that a few simple changes have made the code much more readable and easy to understand. It's not that the code is doing anything very different, but that the intentions of the programmer are made clear because the structure of the code makes this obvious. The code is also simpler, which makes it easier to understand.

Also note the comments preceding the function. It's important to document the public interfaces of our code, and there are many tools available which can extract the comments from the code and produce full API references.

```
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>

/**
* detect_type:
* @value: the value to detect the type of
* @test: true if this is a test,
* otherwise false
*
* Detect a type.
*
* Returns a string describing the type of
* @value, or "&nbsp;" if @value was not
* found. The string must not be freed.
*/
const char *
detect_type (uint8_t value,
       bool test)
{
  struct decide
  {
    uint8_t num;
    const char *string;
  };

  static const struct decide cadena[] =
    {
      {0x00, "Sobre V PK"},
      {0x02, "Sobre V RMS"},
      {0x0E, "—Power OFF—"},
      {0x10, "—Power ON—"},
      {0xff, “”},
      {0x00, NULL}
    };

  static const struct decide cadenatest[] =
    {
      {0x00, "Sobre V PK Test"},
      {0x02, "Sobre V RMS Test"},
      {0x0E, "—Power OFF—"},
      {0x10, "—Power ON—"},
      {0xFF, ""},
      {0x00, NULL}
    };

  for (const struct decide *iter =
        (test == true) ? &cadenatest[0] :
                         &cadena[0];
       iter->string != NULL;
       ++iter)
    {
      if (iter->num == value)
        return iter->string;
    }

  return "&nbsp;";
}

// A simple test case
#include <stdio.h>

int
main (void)
{
  printf("Value Test String\n");
  printf("——- ——— ———————————\n");
  printf("0x00, false: '%-20s'\n",
detect_type(0x00, false));
  printf("0x02, false: '%-20s'\n",
detect_type(0x02, false));
  printf("0x0A, false: '%-20s'\n",
detect_type(0x0A, false));
  printf("0x0E, false: '%-20s'\n",
detect_type(0x0E, false));
  printf("0x10, false: '%-20s'\n",
detect_type(0x10, false));
  printf("0xFF, false: '%-20s'\n",
detect_type(0xFF, false));
  printf("0x00, true: '%-20s'\n",
detect_type(0x00, true));
  printf("0x02, true: '%-20s'\n",
detect_type(0x02, true));
  printf("0x0A, true: '%-20s'\n",
detect_type(0x0A, true));
  printf("0x0E, true: '%-20s'\n",
detect_type(0x0E, true));
  printf("0x10, true: '%-20s'\n",
detect_type(0x10, true));
  printf("0xFF, true: '%-20s'\n",
detect_type(0xFF, true));
  return 0;
}
```

From Nevin Liber <nevin@eviloverlord.com>

First observation: I'm not sure what compiler the student was using. I couldn't get it to compile under either CodeWarrior 9.5 or gcc 2.96, even turning off all the warnings I could.

Bug #1: The compiler error message (this one from Metrowerks; gcc is similar) points out the bug:

```
Error : illegal implicit conversion from
'char *' to 'unsigned char'
Detect_type.c line 34     {0x00, "Sobre V PK Test"},
```

Tracing back, `struct decidetest` has three elements while each of the initializers for the `cadenatest` array only have two elements. Removing element `numero` from `decidetest` fixes the bug. Writing a little test harness:

```
#include "Detect_type.h" /* Detect_type(...)
                            prototype */
#include <stdio.h>

int main()
{
  unsigned int valor;
  for (valor = 0xa000 - 1;
       valor <= 0xa000 + 1; ++valor)
  {
    static const unsigned char tipus[] =
      { 0x00, 0x02, 0x0e, 0x10, 0xff, 0xbe };
    unsigned t;
    for (t = 0;
         t != sizeof(tipus)/sizeof(tipus[0]);
         ++t)
    {
      printf("Detect_type(0x%.2x, 0x%.4x)"
        " ==\"%s\"\n", tipus[t], valor,
Detect_type(tipus[t], valor));
}
}
return 0;
}
```

I get the following output:

```
Detect_type(0x00, 0x9fff) == "Sobre V PK "
Detect_type(0x02, 0x9fff) == "Sobre V RMS "
Detect_type(0x0e, 0x9fff) == "—Power OFF—"
Detect_type(0x10, 0x9fff) == "—Power ON—"
Detect_type(0xff, 0x9fff) == " "
Detect_type(0xbe, 0x9fff) == "&nbsp; "
Detect_type(0x00, 0xa000) == "Sobre V PK Test"
Detect_type(0x02, 0xa000) == "Sobre V RMS Test"
Detect_type(0x0e, 0xa000) == "—Power OFF—"
Detect_type(0x10, 0xa000) == "—Power ON—"
Detect_type(0xff, 0xa000) == " "
Detect_type(0xbe, 0xa000) == "&nbsp; "
Detect_type(0x00, 0xa001) == "Sobre V PK Test"
Detect_type(0x02, 0xa001) == "Sobre V RMS Test"
Detect_type(0x0e, 0xa001) == "—Power OFF—"
Detect_type(0x10, 0xa001) == "—Power ON—"
Detect_type(0xff, 0xa001) == " "
Detect_type(0xbe, 0xa001) == "&nbsp; "
```

Now, the student said if valor is not above 0xa0000, then 'decide' is used. While my intuition makes me suspect he wasn't being precise, since I can't ask him in person, I have to go with what he said.

Bug #2: The edge case of 0xa000 is incorrect.

Fixing those two bugs:

```
char* Detect_type(unsigned char tipus,
       unsigned int valor)
{
int i;
static char escape[16] = "&nbsp;          ";
static struct decide {
unsigned num;
char *string;
} cadena [] = {
          {0x00, "Sobre V PK "},
          {0x02, "Sobre V RMS "},
          {0x0E, "—Power OFF—"},
          {0x10, "—Power ON—"},
          {0xff, " "},
};

static struct decidetest {
/* Bug #1: unsigned numero; */
unsigned char num;
char *string;
} cadenatest [] = {
          {0x00, "Sobre V PK Test"},
          {0x02, "Sobre V RMS Test"},
          {0x0E, "—Power OFF—"},
          {0x10, "—Power ON—"},
          {0xff, " "},
};

if (0xA000 < valor) /* Bug #2: if (valor >=
0xA000) */
   {
     for(i = 0; i < sizeof(cadenatest) /
         sizeof(cadenatest[0]); i++)
         if(cadenatest[i].num == tipus)
           return cadenatest[i].string;
   }
else
   {
     for(i = 0; i < sizeof(cadena) /
         sizeof(cadena[0]); i++)
     if(cadena[i].num == tipus)
         return cadena[i].string;
   }

return (escape);
}
```

While this is now producing correct output, it can be made a lot better.

1. Formatting. At a minimum, function bodies are indented and `struct` members are indented inside their `structs`. This just makes it easier to see the underlying structure of the code.
2. Types. The type of `tipus`, `decide.num` and `decidetest.num` should all match. I would add the following:
   ```
   typedef unsigned char tipus_t;
   ```
   and change the types for all three of those variables to `tipus_t`.
3. String literals.

`escape` is declared to have a size of 16 bytes, yet the literal that initializes it has a size of only 15 bytes (including the trailing '\0'). Plus, most of the other string literals have a size of 17 bytes. A reasonable guess is that the caller is expecting any string returned to have a size of 17 bytes. Additionally, there is no reason to store `escape` in a function static variable, since we can return the literal directly. Finally, since we are returning string literals, the caller shouldn't modify the elements in those string literals; therefore, the function should return a `const char*`.

And while we are on the subject of `const`, both `cadena` and `cadenatest` should be `static const structs`, to insure future maintainers of the code don't modify them.

Issue #3: Refactoring the `structs`

Looking at `decide` and `decidetest` now, I noticed that they have exactly the same members. So I made them instances of the same `struct`, which I call `decide_t`.

Issue #4: Declarations should be closer to use

The `cadenatest` array is only needed when `valor` > 0xA000, and the `cadena` array is only needed when `valor` <= 0xA000, so I moved their declarations into their respective `if` clauses.

Putting all of that stuff together, the code now looks like:

```
typedef unsigned char tipus_t;

const char* Detect_type(tipus_t tipus,
```

```
unsigned int valor)
{
  /* static char escape[16] = "&nbsp; "; */
  struct decide_t {
    tipus_t num;
    const char* string;
};
if (0xA000 < valor) {
  static const struct decide_t
    cadenatest[] = {
        {0x00, "Sobre V PK Test"},
        {0x02, "Sobre V RMS Test"},
        {0x0E, "—Power OFF— "},
        {0x10, "—Power ON— "},
        {0xff, " "},
  };
  int i;
  for (i = 0; sizeof(cadenatest) /
sizeof(cadenatest[0]) > i; ++i)
    if (cadenatest[i].num == tipus) {
      return cadenatest[i].string;
    }
  } else {
    static const struct decide_t cadena[] = {
        {0x00, "Sobre V PK "},
        {0x02, "Sobre V RMS "},
        {0x0E, "—Power OFF— "},
        {0x10, "—Power ON— "},
        {0xff, " "},
    };
    int i;
    for (i = 0; sizeof(cadena) /
sizeof(cadena[0]) > i; ++i)
    if (cadena[i].num == tipus) {
      return cadena[i].string;
    }
  }
  return "&nbsp; ";
}
```

Issue #5: Eliminate the loops

Hand coded loops are both error-prone and algorithmically slow (O(n)). We can do better by using a switch statement. While some folks might squawk at having multiple returns, I found it easier as I didn't need any local or static variables (other than the return). Here is the complete redesign:

```
const char* Detect_type(unsigned char tipus,
unsigned int valor) {
  if (0xA000 < valor) {
    switch (tipus) {
    case 0x00:
      return "Sobre V PK Test";
    case 0x02:
      return "Sobre V RMS Test";
    case 0x0e:
      return "—Power OFF— ";
    case 0x10:
      return "—Power ON— ";
    case 0xff:
      return " ";
  };
} else {
  switch (tipus) {
  case 0x00:
    return "Sobre V PK ";
    case 0x02:
      return "Sobre V RMS ";
    case 0x0e:
      return "—Power OFF— ";
    case 0x10:
      return "—Power ON— ";
    case 0xff:
      return " ";
    };
     }
  return "&nbsp; ";
}
```

Commentary

The student claimed *the function compiles all right* – but they were unlucky with their compiler! As both entrants found with gcc, many compilers will not actually compile the code – and some will only compile it as 'C' code but not as 'C++' code. Actually, I suspect that some warnings were generated for the student but they probably ignored them. Compiler warnings for computer programmers are much like pain is for athletes - something is wrong and simply pressing on may cause more serious damage.

The basic problem was initialising a structure (`decidetest`) with the wrong number of data values. However the code hid this problem since the two data structures were accessed in such a similar fashion that the reader expected they were defined identically. I would suggest the writer of the code should follow the so-called 'principle of least surprise' and use the same structure for both arrays.

As Roger's solution shows making the two structures share a common structure also makes it easier to avoid code duplication – the same loop can be used for both the data structures (although in this case it also changed the loop termination condition from a count to a test of a 'special value' - `NULL`).

An approach that could be considered as an alternative to a `switch` statement is a 'library' solution – since the arrays are sequential the standard 'C' library function `bsearch` can be used to find the match. This would avoid writing an explicit loop at all but retains the explicit data structure.

The Winner of SCC 34

The editor's choice is: **Roger Leigh**
Please email `francis@robinton.demon.co.uk` to arrange for your prize.

Student Code Critique 35
(Submissions to `scc@accu.org` by September 1st)

Here is a C++ header file with a number of potential problems. Please critique the code to help the student identify the problems and to help them to provide some better solutions.

Note: the class Report is not shown. It contains a large amount of data, which can be explicitly deleted by a call to ClearAll.

```
// Reports : vector of reports
class Reports : public map<Data*, Report>
{
public:
  Reports() : nIndex(0) {}
  void ClearAll()
  {
    for (iterator iter=begin();
         iter != end(); iter++)
      (*iter).second.ClearAll();
  }

  Report& GetReport(int nReport)
  {
    int nSize = size();
    assert(nReport < nSize);
    if (nIndex == 0 || nIndex > nReport ||
                    nIndex >=(nSize-1))
  {
    iter = begin();
    nIndex = 0;
  }
  for (; iter != end(); iter++, nIndex++)
  {
    if (nIndex == nReport)
      return iter->second;
  }
  // keep compiler happy
  return *((Report*)0);
}

protected:
  int nIndex;
  iterator iter;
};
```

Francis' Scribbles

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

SESE or SEME

Several very competent programmers have been debating the relative merits of the requirement for a 'Single Entry, Single Exit' coding style. This debate is on comp.lang.c++.moderated. Andrei Alexandrescu's passionate defence of SEME (single entry, multiple exits) has led to the debate being more than just the 'lazy' versus the 'experts'.

You should note that SESE is not just about functions having only one return statement; it is about there being only one exit from any structure.

Part of Andrei's case is that the existence of exceptions in C++ (and C#, Java etc.) means that code often has potential for an alternative exit and the programmer needs to be aware of that and handle it.

Some of you may remember that I have never been an advocate of making SESE an absolute requirement for good code. I also place relatively little value on the arguments from SESE proponents that their code is easier to read. Ease of reading is partly governed by familiarity with the writing style.

Now from my perspective, the primary question is how to get more people writing good code. Here are some thoughts on the topic.

The best is the enemy of good. My observation is that when we present novices with SESE as an absolute, they write very poor code. In a strict sense, their code is SESE compliant but they achieve this with contortions. This bares a similarity to what happens when you tell novice writers that they must not use the passive voice.

There is a wide gap between the excellent programmer and the good one. Adopting methods that make more programmers into good ones without trying to turn them immediately into excellent ones, IMO, is more beneficial in the long term. (Note that I avoided the split infinitive, but did that avoidance help you understand the sentence?)

Now, I freely admit to being less than a devotee of SESE. However, I do have certain personal guidelines:

1. Loops have one exit point. This maybe internal, in which case the loop is written as `while(true){    }`
2. A loop-iteration may terminate early (for example, with `continue`) but only once in the body of a loop. Actually I very rarely if ever use `continue`.
3. All exits from a `switch` must be to the same place (either all `breaks` or all `returns`)
4. If a function contains more than a single `return` statement, re-view its structure to see if it can be written more cleanly with only a single `return`.
5. Be wary of negative tests, human beings do not handle these well.
6. Nested structures can usually be replaced by function calls. I make a great deal of use of the unnamed namespace for such functions.

I find application of these guidelines leads to most of my code being single exit, but I think that is a consequence not a target.

The fact that good code usually has single exits points from structures does not, in my opinion, mean that we should teach programmers that SESE is an end in itself. What we should be doing is teaching them coding methods that naturally lead to code that usually has only one exit point.

What I would value, by way of feedback, is more guidelines that help the less experienced programmer write simpler code. So what do you have to offer?

Recommended Books

Some time ago, I suggested that we should attempt to produce single topic lists of recommended books. For example, a list of useful books for an experienced C++ (C, Java, C# etc.) or a list for those writing for embedded systems. So far, I have had a single volunteer (to contribute a list for an aspect of Python).

I think we can do much better than that. I think we should do much better than that. Software developers do not have time to waste on reading bad or mediocre books. However, they do not have time to invent everything from scratch. They need good references and tutorials and they need help selecting relevant books.

Like the parable of the Stone Soup, if each contributes a little bit we eventually have something worthwhile. I have neither the expertise nor that time to sit and create lists. Nor do most of you, But each of us has time to draw up a list of books for what we are experts in and then get others to review and refine it.

Problem 21

Consider:

```
class x;
class xyz {
public:
  xyz();
  ~xyz();
  static int const elements(100);
// rest of interface
private:
  x * pointers[elements];
};

xyz::xyz(){
  for(int i(0); i != elements; ++i){
    pointers[i] = 0;
  }
}

xyz::xyz(){
  for(int i(0); i != elements; ++i){
    delete pointers[i];
  }
}
```

Is there a better way to implement the constructor? Of course there is, and I know some of you know it but do you?

Commentary on Problem 20

Here is a miniscule program. Now I think it is clear what the programmer intended but what do you think a conforming compiler will do with it and why? How should the author have written the definition of `p`?

```
struct data {
   int i;
   int j;
};
int main() {
   data** p = new (data*)[5];
}
```

It seems that the writer wants an array of five pointers to `data`. That looks a little unusual so the first question I would ask is 'Why do you want an array of pointers? Perhaps he is intending to create a two dimensional array of `data`.

It is easy to answer the immediate question; remove the parentheses surrounding `data*`.

The reason I asked the question was that I suspected (correctly as it happens) that most (nine out of the eleven) respondents would simply answer the syntax question. That is the easy way to respond to requests for help and many people asking questions like such simple answers, but it does not help them in the end.

By the way, in my opinion, the programmer should either be declaring an array (if the correct size will be fixed and known at compile time) or be using a `std::vector<data *>`. I am always deeply suspicious of uses of `new[]` outside low-level classes.

Cryptic Clues for Numbers

Twice lucky? Thrice lucky? About when China ruled the seas.

Here is Mick Brooks' clue for 1421:

The ultimate answer (read between the lines) is the last Lancastrian's birthday.

By the way, if you have not already come across it 1421 (ISBN 0-553-81522-9) is a fascinating read. I wish I had time to try to untangle wild speculation from the underlying reality. This book should probably be treated as a historical novel. However, like all such novels, it is hard to separate truth from fiction.

Here is another little clue to exercise your grey cells (A knowledge of classic SF might help).

The reverse of this issue adds a score to burning books. (3 digits)

Francis Glassborow

Francis Glassborow (francis@robinton.demon.co.uk) is a freelance computer consultant and long-term member of BSI language panels for C, C++ and more recently Java and C#. He is a regular member of the UK's delegations to WG14 and WG21. He is also the author of 'You Can Do It!' and introduction to programming for novices.

Comments

Not that huge a postbag this edition, but something in Pete Goodliffe's Professionalism article #32 did catch the eye of Chris Smith.

From Chris to Pete

I have just read your article 'Professionalism in Programming #32' (C Vu, June 2005). You invited us to voice our disagreements, so here it is: I disagree. :-)

My (single) complaint is with your improved version of the '`greatest_common_divisor` function. To quote your article:

> *Try feeding it a negative argument. This is a more robust (and more efficient) version written in C++:*

```
      int greatest_common_divisor(int a, int b)
      {
            a = std::abs(a);
            b = std::abs(b);
            ...
```

My complaint is that you are enforcing restrictions on the input which are not visible to the 'clients' of your function - you are effectively saying '*my function can't handle negative numbers, so I will silently convert them to positive numbers*'. (Granted, you could detail this restriction in comment, but that's not the point.)

I would have preferred the following:

```
  unsigned int greatest_common_divisor(
       unsigned int a, unsigned int b)
```

That clearly states that you should not call this function with negative values. (And, if you do the compiler will warn you.)

Now I invite you to disagree with **me**. :-)

Chris Smith

Pete Replies

Firstly, thanks so much for taking the initiative (and the time) to reply to the challenge I laid down in this article. Any writer really appreciates feedback! For years I've been trying to provoke some kind of response in my articles (either in letter form, or on accu-general) so it's great that #32 has provoked people to think and reply.

Also, well done for using your brain and disagreeing with what I wrote. Too many people fall into the "mindlessness trap" when reading books, magazines, even stuff online. You must always filter what you read through your own knowledge and understanding, and only accept what's definitely useful. Just because something is in print, that doesn't mean it's gospel (I know that some of the stuff I've written over the years has been drivel, and no one's challenged me about it!).

So, to your specific point: I understand what you're getting at, but (within my granted freedom to do so) I disagree with you! At no point did I state that a negative argument is invalid input. Indeed, it is most definitely a valid form of input. However, negative input would have caused an infinite loop in the original (typographically challenged) version of the GCD function.

The corrected version accounts for this by converting all input values to positive integers before working on them. The side effect of this is to only ever return positive GCD values but, more often than not, that is what's required anyway.

Of course, if negative values were invalid input then it's preferable to make this explicit in the code by careful choice of data type.Thanks again for your response. I hope my future columns elicit the same level of response!

Pete Goodliffe

Do you have any points you wish to be raised about anything to do with C Vu? If so, please just drop me a line - `cvu@accu.org`.

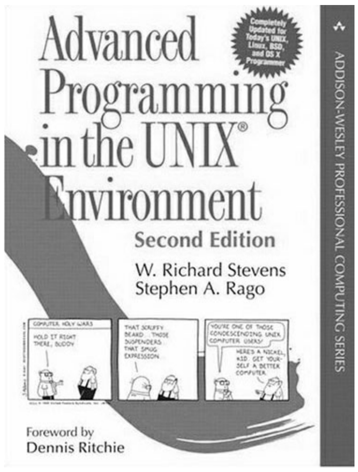
Caption Competition

Paul F. Johnson `<cvu@accu.org>`

It's been a long day. A very long day. Nothing wanted to compile, the boss decided that your project stunk and wants you to do something mind-bendingly tedious.

The Competition

Come up with a caption for this picture. Best one by the next edition wins a pristine copy of Advanced Programming in the UNIX environment, Second Ed. worth £53.99 (hardback).


Advanced Programming in the UNIX® Environment
Second Edition
W. Richard Stevens
Stephen A. Rago
Foreword by Dennis Ritchie
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

And before you ask – the handsome chap is me!

Paul Johnson

Features

ACCU Conference 2005

David Nash `<David.Nash@WallStreetSystems.com>`

I was inspired by Pete Goodliffe to write up some of my experiences from the ACCU conference in Oxford last April, as he did in the June issue of C Vu. Unfortunately the deadline for that issue whooshed past before I was able to get it finished, but perhaps that's not so bad as it means you get two consecutive issues containing conference reports.

These reports are based on the notes I made at the time, so if I have reported anything incorrectly you can blame me for not paying proper attention during the conference sessions. I hope they give a taste of the kind of things you can learn about in the formal sessions (you can of course learn many other things in the informal "sessions" in the hotel bar and the pubs of Oxford!) I have only omitted a couple of sessions, and that's because those were so interesting I didn't make good notes!

I would like to point out that all the sessions I attended were worthwhile and interesting, a testament to the ability of the presenters and relevance of their subject matter. I didn't fall asleep once, not even in any after lunch talks!

Wednesday Keynote: Ross Anderson

The first keynote speech of the conference was on the subject of security, a subject of course that has heightened importance in these days of script kiddies, phishing, DDOS, and other such threats. It was also a keynote on the subject of Open Source vs. Proprietary software, a nice controversial subject guaranteed to get the conference delegates talking on the first day of the conference proper.

After struggling with failing audio amplification, Ross posed the question: *Which is better for security, Open or Closed systems?*

In order to try to answer the question he turned to statistics and mathematics, and made an analogy between thermodynamics and software development, where the number of bugs in a software system is the analogue of the temperature of a physical system. At first I feared the mathematics would make this a rather dry keynote, however this wasn't to be and Ross moved on, making the speech more interesting as it progressed.

According to Ross it turns out, rather controversially, that open and closed source systems are equally good when it comes to finding and removing vulnerabilities (bugs) in them, because the statistics shows that the benefit from the undoubted increased initial rate of bug finding in the open source world is cancelled out in the closed source arena.

Having dropped this bombshell on the open-source advocates in the audience, he qualified this with the phrase "in an ideal world", and pointed out that nothing is black and white — various factors affect this equivalence, then becoming the first to mention one of the phrases of the conference, "Symmetry breaking". Then Ross moved on to report real-world experience backing up his findings, and empirical studies designed to identify the best approach to bug finding and fixing.

Concluding, Ross compared the process of software development with that of medicine: It started out as rather a black art, known only to initiates, but is now big enough to enable statistical methods to be used to help all of us.

Pete Goodliffe: Life in the Software Factory

In a packed Cartoon room, Pete started off by nearly alienating his audience when he said, "There are a lot of dross programmers out there!" Of course everyone in the room knew he was not talking about them, was he?

To get his audience involved, Pete then plucked several volunteers from the audience, not all of whom were called Alan (or Allan), and proceeded to use them to create a simulation of a software program. This was great fun and involved ultimately several threads passing data, in the form of a ball, around.

This generated much discussion and quite a bit of heckling from the audience, not for the first time in this talk. However the fun was now over and Pete moved on to his slides, beginning with a diagram showing a target with "the individual programmer" in the middle. I don't think this is what was intended however, as the idea was to show levels of teamwork — the main theme of this session.

The message from this talk was that software development management structure affects code, at least the "code shape". To clarify this, Pete listed seven team disasters or diseases, followed by a discussion of the opposite - team health.

Finally we were urged to use one of Pete's action sheets to actually go and achieve something positive as a result of this talk. Hopefully some people actually did this!

Jutta Eckstein: Planning, Estimating, & Correction in an Agile World

In this session, Jutta talked about how planning fits in to the Agile software development methodologies' short release cycles. The assumption seemed to be that most of the audience were at least fairly familiar with the concept of the Agile methodologies, but to be sure, Jutta began by reiterating their main points. The one that stood out the most was that "working software is a measure of success".

Moving on, the session proper began by discussing the short "time boxes" used in Agile development, and in particular, what was the best day of the week to finish each time cycle. Aided by two energetic members of the audience, both of whom were called Allan (or Alan) Kelly, it was quickly established that whatever day you settled on, it shouldn't be Friday.

Jutta continued on the discussion of various considerations when planning Agile projects. Many of these seemed like common sense, but like much good advice you don't always think of it until after the event, so having it presented in black and white is helpful. An example of a common sense point is, when requirements change — and they will —- don't try to control all four of the following factors: **Time**, **Project scope**, **Resources**, and **Quality**. It seems obvious but many people working in traditional methodologies would try, and fail. Leading from this is the idea that fixed-price projects are BAD, but realistically they are common, so you must plan to vary the scope or the time of such a project, but not both!

The key point underlying most of the principles is that your plan should be result-oriented, in other words you aim to deliver working software to the client. After discussing the planning of the Agile project's iterations, Jutta moved on to talk about measurement of results and reflection on past iterations or releases. Such feedback is vital in planning the next iterations.

Summary: Remember, requirements WILL change, this is not a bad thing because it means the customer knows more clearly what he or she wants. However this means we can't begin with "a plan", but rather engage in the activity of "planning" throughout the life of the project.

Paul Grenyer: Aeryn

Aeryn is a C++ test framework Paul developed to help him test his software. It can be used for unit-testing but is not specific to this function. Apparently it's named after a character in a television science fiction series.

Aeryn makes heavy use of macros, which Paul decided to state up-front, in case any of the audience had "a problem" with that. No-one did (or were brave enough to admit that they did) which was a relief as otherwise the session could have digressed straight away into a holy war over when and where to use macros.

A multi-layered approach is used, and Paul began at the bottom level, describing the Test Conditions he has implemented (with macros!) that allow you to put tests in your code using such expressions as `IS_EQUAL(LifeTheUniverseAndEverything, 42)`.

The next layer up is that of Test Fixtures, which are functions and classes containing Test Conditions. Next, Test Cases are wrappers for Text Fixtures. Finally Test Runner is an object to which Test Cases are added

Once a Test Runner has been created and appropriate Test Cases added to it, it can be invoked, whereupon it will run all the test cases and generate a report of each one's success, or otherwise. Finally it will return an overall status which can help when scripting automated runs of test cases.

Paul then showed a simple example using Visual Studio .Net. and engaged the audience in discussion about the reporting features. He also showed how Aeryn could be extended to generate custom test reports.

Finally, because this talk was a relatively short one, Paul decided to fill the time with a discussion of a testing technique called "Mock Objects". This is not related to Aeryn but Paul included it as it's a useful testing technique.

Thursday Keynote, Bjarne Stroustrup: Generic Programming

This was always going to be a popular one, and it paid to turn up early. As one might have supposed, the hall was packed with expectant delegates eager to hear what news the founder of C++ had to bring.

Bjarne started off with a threat to mobile phones. Suffice to say that people hurriedly turned theirs off in fear of being singled out.

On to the real subject of the day: C++, or to be more precise, the current state of the language and how it can (will?) be improved to better support generic programming.

The ability to do generic programming with C++ through templates has had a major impact on the development and use of the language. Templates, both through the STL, and through metaprogramming, are a big success and make many programming tasks easier. But there are problems, which Bjarne described as "the language is straining" under the effort. Certain error messages are an abomination, and some uses of templates simply require too much brain power!

Bjarne stated that the aims of the language additions he was describing were to support generic programming and templates. He described the following three areas for improvement:

1. Minor improvements including `auto` and `decltype` which are used to enable the compiler to work out the actual type of a variable itself, and template aliases which should allow "template `typedefs`".
2. Concepts - these are a kind of restriction on what "kind" of class a typename passed as a template parameter can be. For example currently when you say "typename T" the user is free in principle to pass any type they want as T. Using concepts you could tell the compiler that only types usable as forward iterators, say, are allowed.
3. Initialization - the final improvement Bjarne described was to allow easier initialization of container classes with a list of items. This can be done with a kind of "sequence constructor". Other constructor improvements were to be expected, including forwarding constructors.

Bjarne Stroustrup: Direction for C++0x

This two-presenter talk followed on from the Bjarne's keynote, and dealt with similar items, ie. changes we can expect to see in the next C++ standard, expected within 10 years (not that long in ISO standard-time!).

Bjarne began by defining the problems faced by the standards committee when trying to "fix" C++. Namely, that people want improvements but you can't please everyone and we also want stability.

So why change? The language must adapt but carefully! He described a list of "rules of thumb" to be followed by the committee with the intention of minimizing problems with new features.

After the talk, Bjarne opened the floor to questions, and some discussion on the merits of deprecating features followed, before he gave way to the co-presenter of this session, Herb Sutter.

Herb Sutter: Something cool in C++0x

As if Bjarne wasn't sufficient draw, the first session was combined with a talk by one of ACCU conferences most popular speakers. Herb, as he often does, had given this talk a subtitle - "The Concurrency Revolution". His point was that concurrency was "here", and is a revolution on the scale of "the OO revolution". By which Herb means, and I quote directly, "*It will change the way you write software*".

One last quote from the introduction to make the point: "*The state of the industry is terrible!*"

Herb really, really wanted to get the point across that concurrency is a big issue and getting bigger. And if we're not careful it's going to be a big problem. He began a list of truths by by describing the fact that Moore's famous law just didn't apply any more. A "wall" had been hit and we can't have faster single-threaded programs any more. Instead, through innovations like dual-core processors and Hyperthreading, we get faster multi-threaded programs, which means we all have to learn to be multi-thread programmers.

The next list after Truths, was Consequences. This was all about issues with memory models (we need guarantees), locking (broken!), and something about Santa Claus and elves that I haven't come across before!

Finally, appropriately, was the list of Futures. Herb pointed out that non-mainstream languages are better but this doesn't help the majority of programmers.

A thought-provoking talk.

Francis Glasborrow: Proposals for Change.

Continuing the day's theme on the future of C++, Francis began by explaining that the C++ standard committee was known as SC22-WG21 and if we want to have a say, we should get involved, urging us to Google for that string of characters.

Now began the list of proposals under consideration for the next version of C++, beginning with the Forwarding Constructor. This brings the much-desired ability for one class constructor to forward to another at the initialisation stage, rather than having to call an ordinary member function from the constructor body. However it also raises the question of when is the class considered to be complete? When one constructor has finished? When ALL constructors, including any forwarded to, have finished? This subject is still under discussion.

Another proposal Francis described was the idea of Explicit Classes. These are classes that have no automatically-generated classes (copy constructors and so forth). The idea is that there will be switches to explicitly re-enable the generation of these functions.

Francis then described a couple of features that are on hold because of overlap with other proposals, before finally mentioning the idea of extended "switch". Currently the C and C++ switch statement can only use constant integers for the case values. I have known people familiar with other languages, where this is not the case, express bewilderment at why this should be so. This reaction is commonest with newcomers who want to switch on a string, and can't understand why they can't put a complete string value at each case. Francis proposed relaxing this rule and allowing more flexibility. Just how much is yet to be decided - should we allow variables as well as constants? In that case why not allow any expression at all. Then the switch would be transformed into simply "syntactic sugar" for a multi-level if-else statement.

Allan Kelly: Software Development As Learning

Allan presented a highly-interactive session in which he presented his idea of how software development and learning are related.

He started by noting that the solution domain (eg. of C++), the application domain, and the process domain ("how we build software") overlap and interact.

Next Allan spoke about learning. Software is the embodiment of knowledge, and learning continues after development has finished. Why is learning important? - because it can enable us to get better software.

The next point led to an audience discussion. Allan stated that learning creates change, and change creates learning.

There was much discussion of points covering information, knowledge, action, problem-solving, and different types of learning (single-loop, double-loop, with triple-loop introduced into the arena by Jim Coplien in the audience).

Now, Allan introduced a phrase to describe bodies like the ACCU: Communities of practice. This is a better term than something like "society", to describe a group which is not an official professional body or guild or suchlike, but which develops standard practice in the industry and encourages learning.

Finally there was more discussion, this time with a whiteboard, in which Allan solicited ideas from the audience for how to improve learning.

It was a very thought-provoking talk, and different in some ways from the majority of conference sessions. Many people left the room saying what a success it had been.

Friday Keynote: Jim Coplien - Beyond the Curse of Symmetry

In another packed keynote, Cope (as he tends to be known) warmed up the audience with a display of amusing slides, including some pictures of Bjarne Stroustrup and Kevlin Henny looking very bizarre indeed. The reason for the bizarreness was that the pictures had been altered so they were completely symmetrical, that is the left side was a mirror of the right side. This neatly and amusingly illustrated that although we tend to think that we are, humans are not really symmetrical.

Cope continued by talking some more (actually rather a lot) about symmetry. He demonstrated that a starfish displays another type of symmetry (it's symmetrical when you rotate it by 72 degrees).

That example was used to point out that there are quite a few different kinds of symmetry. In computing we tend to think of structures of "blocks", rather than the UML diagrams we are encouraged to use. These blocks are symmetrical and regular. In computing symmetry is the holy grail, but he also pointed out that broken symmetry leads to beauty. This was backed up with some quotes from Christopher Alexander, the originator of the patterns movement in architecture, which has been taken up to enthusiastically by some of the computer science community.

Returning to the idea of symmetry, via a rather funny joke he had been told about Iraq and the United States Constitution, Coplien described the physical process of k-meson decay, which broke an underlying symmetry and was apparently responsible for the overal balance of matter against antimatter in the universe. Consequently, he suggested, God is left-handed. A wag in the audience asked whether He drove on the left too!

Finally returning to computing, there are many design methods that break symmetry. C++ reflects reality by allowing us to express broken symmetry. In this respect reality is messy — or, since the symmetry is broken, is it beautiful?

Frank Buschmann: Model-Driven Software Development

According to Frank Buschmann, Model-Driven (software) Development, or MD(S)D, is motivated by “the software crisis”. Software development is expensive, and despite multi-structured and component services design, complexity hasn’t gone away.

MDD uses a domain-specific language to specify a model of the system to be implemented, then uses a model compiler (that is, a compiler of models, rather than a model of a compiler) to generate code that implements the system.

Models can be built using common modelling tools such as Visio, XML, or others.

Frank described a two-step process of building the concrete design: step one takes the previously-mentioned model, and various performance, scalability and architecture requirements (and so on). This generates the architecture. Then step two generates a concrete design, code and configuration.

Naturally, what you get out of it depends on what you want and on the domain. As you might expect, and Frank emphasised, the model must be extremely precise. It takes a lot of up-front effort. Although it apparently removes the need for coding, MDSD should not be seen as a silver bullet, but as just another software development method.

Frank described some apparent disadvantages of the method, such as:

- Off-the-shelf software tools are usually not appropriate, and custom ones must be developed
- Skilled software developers are required to implement the model
- The domain must be well-understood and have well-defined boundaries (it seems to me that this applies to any software development method!)

Following Frank’s description was some discussion with the audience who were, perhaps understandably, rather sceptical. Contributions were made by the usual suspects — Henny, Jossutis, Stroustrup, one of the Allan Kellys.

After this, Frank conceded the audience’s point that MDSD was probably not worth it for single unique software projects. Where it stands up is when the architecture and model can be re-used for different applications within the domain.

Finally Frank noted that there were several implementations including the OMGs MDA, OpenArchitectureware, and Microsoft Software Factories.

Hubert Matthews: Concurrency Requirements

Concurrency was one of the topics that cropped up several times in this conference. Herb Sutter named it as the next big thing in computer software development, and so this talk was rather popular.

Hubert started by noting that not much progress has been made in concurrency development. We still use the same old Critical Sections, Mutexes and Semaphores to implement the locking and checking that’s required for safe concurrent programming. And it’s still hard.

Next he asked the question, “What do users want?” There are often unexpressed requirements such as “it must not do Y” rather than “it must do X”.

The next question is, “What do clients want?” Using the example of a price lookup for a shopping or catalogue system, what is needed is a guarantee over time that the price won’t change unexpectedly (i.e. in the middle of a transaction). Note that the concurrency being described here is that of a client and server both accessing the same data.

Four types of locking were described:

- Exclusive lock
- Time-based
- Optimistic
- None

Questions are raised - in the case of two clients making changes to a database record, which one “wins”? The one that read the record first? The one that makes the write first? The one that read the record last? Or the one who writes the record last? Arguments can be made for each. Again, concurrency is hard.

Finally (at least according to my notes) described the so-called Heisenberg Triple. This is a familiar type of statement that gives three criteria and tells you, you can have two out of the three, but not all of them. In this case the three options are Client consistent with server; Client has data available; and Client is independent of the server.

To finish with a quote, to achieve a compromise between all of these, there is usually some “acceptable window of unsynchronisation”.

Saturday Keynote: Kevlin Henny - Five Considerations

As it was for the other keynotes, the hall was packed on Saturday morning, despite following the Speakers Dinner the night before. Somewhat appropriately, Kevlin began by describing how this talk was the result of a pub conversation, the details of which are available in Kevlin’s blog at `http://www.artima.com/weblogs/viewpost.jst?thread=5432`

The five considerations of the title are five points that Kevlin came up with in the aforementioned pub conversation, when asked for advice for beginners in software design.

The first of them is a restatement of the “Less is more” sentiment familiar to readers of Kevlin’s articles - “Less Code, More Software”. Those readers will be aware that Kevlin sometimes likes to speak in sound bites, at least in his talks and articles, and for this point he excelled with “remove to improve”, but, “Don’t encode your code”. In other words concise code is good so long as it’s readable.

The second of the five considerations is undoubtedly the recurring theme of the conference, “Symmetry”. Kevlin pointed out that normally, symmetry is seen as good. However this is not always the case and sometimes it must be broken, a point made by Jim Coplien in his keynote.

The next subject for our consideration is “Spacing”. In other words, separation of concerns, frequently touted as A Good Thing in object-oriented software design. This time it came with a warning to avoid too much separation, resulting in Fragmentation (A Bad Thing!) It’s also concerned with making your code easy to read through the sensible use of white space and suchlike.

“Visibility” is the next consideration. Software is aphysical. As Kevlin pointed out, we can’t use our physical intuition about it.

Lastly, we must consider “Emergence”. This refers to so-called emergent behaviour in which a few simple rules produce apparently complex results. One of the classic examples, and the one described by Kevlin, is birds flocking. A few simple rules like “follow your neighbour” results in the ability of huge flocks to sweep gracefully across the sky all apparently knowing where they are going. In software terms we are being told to use simple rules and mechanisms such as polymorphism rather than complex sequences of “if” statements.

Finally, it is important to remember that these considerations are not rules or recommendations, but just as the word suggests, things to consider.

Lois Goldthwaite: XSLT

In this presentation, Lois used her experience of being thrown “in at the deep end” when having to implement an XSLT (XML stylesheet language and templates) project, to communicate the absolute minimum you have to know to get up and running, and productive, with XSLT.

A sampling of the points Lois made:

- It might be tempting to avoid XML namespaces, but it is absolutely essential to understand them (and use them) when using XSLT
- The minimal template just outputs the XML element content
- Tools can be useful, such as XML Cooktop and Altova XSLT
- Test-Driven-Development is recommended when developing XSLT applications
- xsl:comment is a useful aid to development, it outputs an XML comment

Lois spent a very useful couple of hours showing how XSLT works, with several sample stylesheets. All in all it was a useful talk that I believe everyone who attended found useful.

David Nash

Becoming and Being Agile

Phran Ryder <phran@agilenorth.org.uk>

Pete and His Pilates

Pete looked at his watch which told him that the time was 17: 25. He cursed and waited for the editor and source file to fill his screen. White characters on a light blue background duly appeared and the hunt began. The urgency and stress of the situation precluded any introspective pity - that would be counter productive. He had read how stress can lead to thematic vagabonding or encystment so he relaxed aiming to keep a cool clinical head.

At a young age Pete had learnt to read quietly without speaking the words, most people do. But like a lot of people who find themselves in a state of irritation he muttered the words he read as he typed. He trawled the code inserting trace lines as fast as he could, " insert, head, delete, next...".

Pete was different from the average programmer because he could do more than one thing at once. Reading while simultaneously speaking is well within his abilities so his mind was able to wonder and ponder a little. 'Why am I doing this'? he thought, 'why am I debugging somebody else's code again'? 'And what crap code it is.'

Since joining the company Pete had tried and tried to introduce even the most primitive principles of software engineering. But all efforts seemed to be in vain. At one time he had explained to his colleague Rod the concepts and advantages of software re-use. Rod had smiled and explained that they re-used code all the time. This re-use was then illustrated by liberal examples of similar looking blocks of statements with minor differences. Choosing his most diplomatic hat, Pete had suggested that there were 'even better' ways of re-using code than cut and paste.

Rod was an experienced, intelligent, amiable developer. Above all he wanted to get the job done. Rod's expression was one that Pete took to be one of interest - he wondered if it was that of someone whose mind is elsewhere but who is too polite to end the conversation. Even so he went on "These blocks of code are all very similar, it would be very simple to extract them into a function and supply a parameter to account for the differences."

Rod's expression of apparent interest changed to one of mild disbelief. "When you do *you* can test it."

If there had been a soap box near by Pete would have mounted faster than an escaping bandit. "Indeed I will but first I will write some automated tests so that the code can be tested and retested in seconds."

Rod's expression be it of interest or disbelief faded in an instance. Now he was listening. And in the manner of many listening programmes his face was completely blank.

"With the automated tests I will be able to re-factor the code to remove duplication, make it easier to read, and I will be able to demonstrate exactly what I have tested. Every day!" Rod's expression didn't move, Pete still had his audience. "With the automated tests I will be able to implement a few requirements each day. In each I iteration will be able to show a working system to the business so they can give feedback on whether it is what they want."

Rod's colleague's expression changed to one of realisation. "Oh that's that Extreme Programming rubbish. I've tried that it's no good." All Pete's attempts are resurrecting the conversation were met by a stony wall of 'been there, done that, burnt the t-shirt'.

After several similar abortive attempts he was finally called before the Managing Director and told in no uncertain terms that "this company does NOT do Extreme Programming. I've read the book - it's for hackers". His attempts to explain the relationship between short iterations, feedback and value were dismissed - but he wasn't - dismissals come after three warnings. Pete left the MD's office boiling inside, steam spurting from his ears, and a mist in front of his eyes. How could someone from *this* company dismiss Extreme Programming as hacking.

Now here he was attempting to debug a classic example of re-use. His thoughts were interrupted as a colleague sped towards the door. It was Norman Moore, normally known as 'that Norman Moore'. "Good night Pete", whispered Norman as he closed the door behind him. Pete was on his feet in a flash and caught Norman as he pressed the lift button.

"Norman!", Pete started, catching his breath. "Weren't you told? Those changes you made for Mega Big Bucks Ltd. still aren't working".

"They worked when I tested them ", Norman retorted. Norman's words tended to slither towards the listener. "Besides I'm going bowling now. Bye." Norman disappeared into the lift.

The delivery to MBB had already been delayed four months. If they didn't get the software to them by tomorrow MBB would cancel the order. That would be the fifth cancellation in as many months. Pete flew down the stairs praying that he would be able to catch Norman and persuade him to stay. As he ran the stairs started to sway beneath his feet. He could feel a warm soft hand stroking his hair.

"Wake Up! Wake Up! You're dreaming again." Pete awoke sweat dripping from his brow. The bed clothes were wrapped tightly around his legs which were still attempting to walk in mid air. He had been dreaming what a nightmare! Pete's nightmare occurred regularly, but he did not mind too much as it helps him to enjoy his new job all the more. Pete had long since given up software engineering and now teaches pilates to rock climbers and professional footballers - at least he was introducing Agility in his own way.

The Message

Pete's tail, while (mostly) fictional is a tail that recounts what has happened or could happen to many. It is a tail that relates a situation in which many people, in many companies, in many industries find themselves. A large number of us have seen or been Pete, Rod or Norman. While the tail makes Rod and Norman appear to be villains, this is unlikely to be the case. They are probably intelligent, capable individuals who, given a better environment, would do a whole lot more.

Across the industry there are countless Pete's who want to change their company, improve their development process, become more agile, and make work a whole magnitude more enjoyable. Most of the Pete's struggle in vain, make little or no change, and often devolve into Rod or Norman. So how can you, or I, or Pete make changes? What changes should be made? Faced with a plethora of methodologies, terminologies, and ruthless consultants where should we turn? What should we do?

The answer is not easy, software development is not easy. This article briefly introduces Agile development. In those that follow I hope to provide my thoughts on what it is to be agile, and develop software in an agile way, and what can be done to become (more) agile. I don't claim that I will give you answers. I hope I will say things that you will want to challenge. I will certainly say things that I will disagree with in a year's time – or sooner. My aim is to inform and in doing so I hope I will give you information or incite that will make it easier for you to find answers yourself.

The Agile Alliance and The Agile Manifesto

In February 2001 a group of seventeen software pundits got together to discuss the growing field of what used to be called lightweight methods. These people were visionaries who had, between them, created development methodologies such as eXtreme Programming (XP), DSDM, Scrum, Crystal, Lean Thinking, and Adaptive Software Development. They decided to use the term agile to describe this new breed of agile methodologies.

They found that they had a lot in common and agreed on many important aspects of software development. So they decided to go further than just talk. They liked the idea of writing a document that would both capture the common ground and act as a rallying cry to the software industry. Later they formed the Agile Alliance <http://www.agilealliance.com> as a non-profit organization to act as a centre for furthering agile methodologies.

The document they created is the Manifesto for Agile Software Development. It sets out the values and principles of these agile processes. The values really capture the core of the ideas. The manifesto says what the seventeen stand for and also what they are opposed to or at least value less. Several items were worded to clearly make a distinction between their views and those views of many others in the software industry. Here is the manifesto.

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

When I first read this my thoughts went from: "Yes, yes, that's it, so right!" to "is that it?" to "what DOES it mean?" This simple statement says masses without saying much – but what does it say? Why is it good? How does it help? In the next article I will give you my (current) thoughts on what the manifesto means and why it is good.

Phran Ryder

*Phran Ryder is Chairman of AgileNorth.org.uk - a non profit organisation for technical and business staff who wish to learn and share their experiences of **becoming** and **being** agile.*

Find out more about becoming and being agile by attend the AgileNorth.org.uk conference, details at: www.agilenorth.org.uk.

Professionalism in Programming #33

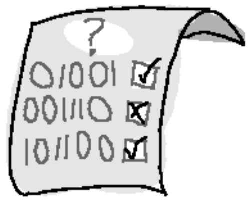
A Review to a Kill

by Pete Goodliffe <pete@cthree.org>

Reviewing has one advantage over suicide: in suicide you take it out on yourself; in reviewing you take it out on other people.
George Bernard Shaw

Is it me, or does this look familiar? Astute readers (who are old-time ACCU members) might feel a sense of déjà vu here. Back in the mists of time, the fourth ever professionalism column discussed code reviews. I recently needed to evangelise reviews once again; it's always the same old battles we fight. Different job, same problems.

I dug back, revisited, and refreshed my look at the code review process. I reviewed the material, if you like. It seemed appropriate to present the fruit of my labours for a fresh audience. We'll start with some craftsman philosophy…



En Route to Good Code

How do you learn to be a good carpenter? You become an apprentice to someone. You watch them work, help them daily, gradually take on more responsibility, and learn from their advice. You don't jump in feet first without any practical ability and expect to churn out quality woodwork straight away.

We don't have a real analogue of that in the coding world, even though programming is as much a craft as it is an engineering discipline (possibly more so). Code reviews, however, do give us a little taste of that in the discipline of an engineering process.

Code reviews (or inspections, or walk-throughs) have similar effects to the open source model of software release – providing a structured opportunity for others to eyeball your precious code. Reviews encourage you to take responsibility for your handiwork. When you know that it's not just for you to look at, but it will be viewed, used, maintained and criticised by others your approach tends to change. You're less likely to make the quick-and-dirty fix that you'll never have time to revise. The accountability brought on by code review brings a greater quality to coding.

Code reviews are employed in traditional software engineering processes. They are arguably less important when pair programming, or when more than one person is responsible for parts of the code. In these situations they are still useful nonetheless.

What is a Code Review?

The code review seeks to analyse a section of source code at several levels:

- the overall design (e.g. the choice of algorithms and external interfaces),
- the expression of that design in the code (e.g. its breakdown into classes/functions),
- the code in each semantic block (e.g. class, function, loop), and even
- each individual code statement.

We look in great detail to ensure that the code is correct and of a suitably high quality. This process generates a huge list of 'must-fix' issues. Sometimes you will spot improvements that are not worth making now; we chalk these up for future experience.

A code review doesn't replace the code's functional specification review. The code is validated to conform with its specification, but the content of this specification is taken to be correct. If it wasn't then the task would be herculean! Sometimes code review comments might feed up to the specification (for example, where clarification is needed), but this is not our ultimate goal.

Code reviews can be:

- **Personal**
  The author carefully and methodically reviews their own work to make sure that it's good. Don't get this confused with casually reading your code after typing it; a 'personal' code review is a more detailed and involved task.
- **Open**
  Involving other programmers brings new expertise, more experience, and more eyeballs to the task. It's consequently harder to coordinate and requires greater overall effort, but is more likely to find problems. It's not easy to delve this deeply in a personal review; often the author is too close to the code and it's easy to overlook problems.

Why Review Code?

Bugs are our enemy, the nemesis of good software development. We need to be confident about the quality of our work, and need to find faults as early as possible in the development process. The earlier we try to find problems, the more we are likely to find and fix.

Code reviews are an excellent tool to achieve this goal. According to Humphrey: "Students and engineers typically inject 1 to 3 defects per hour during design and 5 to 8 defects when writing code. They only remove about 2 to 4 defects per hour in testing but find 6 to 12 per hour during code review" [Humphrey 97].

But code reviews do more than identify bugs; they weed out all sorts of problems. We perform code reviews to improve code quality. This includes:

- removing coding errors,
- identifying design problems,
- removing redundant code, and
- ensuring efficient[1] algorithms are used

We also check the code against a number of yardsticks, which include:

- its specification,
- any project coding standards and best practices, and
- the appropriate language idioms (e.g. ratify the use of design patterns)

Apart from the obvious benefits of correct code, reviews have other useful side-effects. The cross fertilisation that comes from looking at each other's code ensures that coding style is more uniform across a whole project. A review also spreads knowledge about the inner workings of core bits of code, so there is less risk of loosing information when people leave a project.

Reviewing the Alternatives

There are a number techniques that could potentially make formal code review meetings redundant. These are:

- **Pair programming**. When you pair program your code is effectively reviewed on the fly. Two pairs of eyes are better than one, and will find many, many more faults as they are entered. However code reviews do still catch more problems, by employing reviewers who are physically and emotionally removed from the implementation work.
- **Open source**. Opening and freely releasing the source code allows anyone to see it, to judge the code's quality and to fix problems. Some call this the 'ultimate code review'. However, it doesn't actually guarantee that anyone will inspect the source. Only really popular open projects have actively maintained codebases. Making some code 'open source' will not instantly bring code review-like benefits.
- **Unit tests**. These are an automatic means to show that a modification hasn't degraded the correctness of your code's output, but they don't help to increase the quality of the written code statements.
- **Not reviewing**. You can alternatively trust the programmer to get it right – that's their job after all. If this is a winning strategy then you don't need to test the code either. Good luck!

None of these, on their own, can honestly replace the code review. Perhaps a combination of them and a particularly effective development team culture would render reviews less necessary.

When do you Review?

In an ideal world every bit of code is carefully reviewed prior to release. According to the Software Engineering Institute at the Carneige Mellon University, a thorough code review should take at least 50% or more of coding time (although they do include 'personal' code review in this statistic) [Humphrey 98]. That would take longer than a Real World project is prepared to invest[2].

So as we write a system, we need to ask whether to review the code and, if so, exactly what code to review.

Whether to Review

Bugs are inevitable, and you can be guaranteed your code contains some classic mistakes. There will be obvious flaws that you'll find quickly, and

1. For an efficient definition of 'efficient' in the code's context.
2. The fact that they're rarely prepared to invest any time in code review is a more damning problem.

many more subtle problems that would only be spotted by a fresh pair of eyes approaching the code with no preconceptions. It's hard for the original author to see most inherent faults in their own work – they're too close to it, suffering the psychological cognitive dissonance described in [Weinberg71]. If your code is at all important (clue: it is, or you wouldn't have written it) and you care about its quality (clue: you do, or you're a disgrace) then you must code review.

Not reviewing code drastically increases the chance of faults slipping into your production software. That could spell your embarrassment, a lot of expensive rework and in-the-field upgrades – even your financial ruin. The effort of a code review pales in comparison with the consequences.

Often people make excuses to justify avoiding reviews. They say "the code's too large to review fully" or "it's too complex, no one person could ever understand it – there's no point even trying to review it". If a company can employ enough people to write a large program, they can employ enough people to review it. If the code is too complex then it desperately needs reviewing! In fact, it probably needs something a little more drastic. Well-written code is decomposed into sections that can undergo separate reviews.

What to Review

Even the most modest project quickly produces a tonne of source code. For all but the most stringent development processes there simply isn't enough time to review every last scrap of code. So how do you decide which bits to review? That isn't easy.

You want to select the code that will benefit most from review. This is the code that is most likely to be bad, or that is most important to the correct functioning of your system. You could try these strategies:

- select core bits of code in the 'central' components,
- run a profiler to see where most CPU time is spent – review those parts of code,
- run complexity analysis tools, and review the worst offending code,
- target areas that have already exhibited a high bug-count, or
- pick on code written by programmers you don't trust (a code review vendetta!).

The most practical approach is probably a hybrid of all of the above.

Performing Code Reviews

There are several ways to perform a code review:

- in a code review meeting,
- a 'virtual' review (run online, with no physical meeting),
- as a 'gate' for code modifications to be included in the main source tree,
- using code review tools, and
- in a personal code review.

Any form of review places the source code under the microscope – really aiming to criticise and verify it. This is not to pillory or 'get at' the author, but to improve the quality of the software the team produces. Simply having a code review is not enough. In itself it's not going to solve all the problems. You also need to make sure that you review properly.

The most common setting is the formal code review meeting. There is a fixed agenda (to ensure that no action is forgotten) and a defined ending (not necessarily a time limit, but a definition of exactly what code you are reviewing, and what you're not – it's very easy to be woolly about this).

An example code review meeting procedure is described below:

Where?

The best place to hold a code review is in a quiet meeting room. The reviewers should not be disturbed. There should be coffee.

A suite of networked laptops with code editors may be useful, as may a computer hooked up to a projector. Old-school programmers swear by printouts and pen-and-paper note taking – detaching from the computer screen can help to find new faults. This really depends on how much respect you have for trees and electricity consumption.

When?

Obviously, at a mutually convenient time. Common sense tells us that Friday at 4 p.m. is not a good time. You need to devote serious time to this, so make sure that you won't be disturbed or distracted.

If the code is too large, split the review into a number of separate sessions. You can't sit people in an enclosed space for hours on end and expect the quality of their review to remain high.

Roles and Responsibilities

One of the most important contributing factors to the success of a code review meeting is who attends. There are a number of distinct roles which people should be specifically assigned. One person can be both a reviewer and another role.

- **Author**. Obviously the person who wrote the code should attend the review, to describe what they have done, argue against unfair or incorrect criticism, and to listen to (and subsequently act on) valid constructive feedback.
- **Reviewers**. The reviewers should be carefully picked, the people with available time and skill to review. It helps if the code is in their area of expertise, or they are involved with it in some way. For instance: the writer of a library should be invited to review a program that uses the library, to diagnose incorrect API usage.
- **Test department**. There should be an appropriate number of experienced software engineers present. There should possibly be a representative from the QA/testing department, so QA can be assured of the software's quality, and the quality of the development process.
- **Chairman**. Any kind of meeting needs a chairman, or chaos will ensue. This person leads the review, and guides the discussion. They ensure that the conversation keeps to the point and that the meeting doesn't get side-tracked. Any minor issues that don't need to be discussed in the meeting should be quickly taken off-line by the chairman. Given half a chance, programmers will discuss a minute technical detail for hours at the expense of the rest of the code review.
- **Secretary**. The secretary takes minutes. This means writing down all points that arise, to make sure that nothing is forgotten after the review. If there is a review checklist then they fill it in. The secretary role should not be fulfilled by the same person who acts as chair.

Before arrival, everyone is expected to have familiarised themselves with the code. Everyone must have read the supporting documentation (any relevant specifications etc)[3] and be aware of any project coding standards. Whoever organises the meeting should highlight these documents in the meeting announcement to prevent misunderstanding.

Agenda

To organise the code review meeting:

- The author signals that their code is ready for review.
- The chairman arranges the meeting (booking an appropriate location, setting the time, and assembling the correct set of reviewers).
- All required resources (computers, a projector, printouts, etc) are arranged.
- The meeting must be called sufficiently ahead of time to allow the reviewers to prepare.
- After the meeting announcement, the author cannot change their code – this is not fair on the reviewers.

The code review meeting is run as follows:

- The chairman arranges for the room to be prepared beforehand, so the review can start on time.
- The author takes a couple of minutes (no longer!) to explain the purpose of the code, and a little bit about its structure. This should be prior knowledge, but it's surprising what misunderstandings can be caught at this first stage.
- Structural design comments are invited. These are comments relating to the structure of the implementation – not the actual code at statement-level. This could include the breakdown of functionality into classes, the split of code into files, and the style of function writing (are there invariants, and good test harnesses?)
- General code comments are invited. These may relate to a consistent incorrect coding style, bad application of design patterns, or incorrect language idioms.
- The code is carefully stepped through in detail, a line or block at a time, to prove that it is correct. The things to look out for are described later.
- A number of example scenarios of code usage are considered, and the flow of control is investigated. This helps the reviewers to understand the code and cover all execution paths.
- The secretary notes all changes required (recording the filename and line number).
- Any issue that might percolate out to the wider codebase is recorded for further investigation.

3. Naturally, all supporting documentation will have been thoroughly reviewed beforehand.

- When the review has finished a follow-up step should be agreed. The possible scenarios are:
  - **OK** – the code is fine, no further work is necessary.
  - **Rework and re-review** –the code needs a lot of rework, and another code review is deemed necessary.
  - **Rework and verify** – the code needs some rework, but another code review meeting is unnecessary. The chairman nominates someone to act as verifier. When the rework is complete, the verifier checks it against the recorded minutes of the code review meeting.

A reasonable deadline should be imposed for any rework, so that the detail of and reasons for actions stay fresh in people's minds.

Remember: the aim here is to identify problems, not to fix them in the meeting. Some problems require considerable thought to fix, and this is a job for the author (or modifier) after the review has finished.

Virtually Different

Code review meetings are a high-ceremony review method. They're hard work, but they undoubtedly find many problems that would otherwise go undetected.

Other less intense review procedures exist, providing most of the benefits of code review meetings but packaged in an easier to swallow pill. Perhaps the most effective is the integration review, performed whenever new code is integrated onto a mainline code branch. This could be when:

- a new piece of code is about to be checked into source control,
- a new piece of code has been checked into source control, or
- a code package is merged from a feature development branch onto the main release branch.

At such a point, the code in question is marked for review, and a suitable reviewer is picked: either someone responsible for that module (the code integrator or maintainer[4]) or a shadow (or code buddy) who is assigned to verify that author's work.

These 'gated' code check-ins are often implemented with a software tool that is integrated with the source control system. They're quite hard to arrange manually, and are usually left as a check-in discipline: you are not supposed to check any code in unless it has already been peer reviewed. This approach is quite hard to police; errors slip past in hurried last minute check-ins.

The actual review step here is usually a lot less formal than the meetings described earlier. The reviewer scans the code to check that it's not obviously broken, tests it (perhaps reviewing the available unit tests to ensure they're valid), and then authorises it for inclusion in the mainline. Only then will the code integrator migrate the verified code into the release tree. For more serious projects, or at more sensitive times (just before a major release milestone, for example) this review step may become much more stringent – requiring more eyeballs and more effort.

Since the reviewer and author needn't actually meet face to face (although it is preferable to do so) this can be considered a form of 'virtual' review process.

Review your Attitudes

Code reviews require a constructive attitude – you need to approach a review with the correct mindset or it will be unsuccessful. This works two ways, for the author, and the reviewer:

Author

Many people shy away from a code review for fear it will expose their inadequacies. Don't do this. Having your code reviewed is a good way to learn new techniques. You must be humble enough to admit that you're not perfect, and willing to accept criticism from others. Your coding style will improve as you learn from the changes made to your work.

As an author, do not be defensive about your code. There is a natural tendency to take all criticism personally and assume it's an assault on your abilities. To cope with a code review, you need to reduce ego and personal pride. Understand that no one writes perfect code: even the most awesome programmer's code will be criticised for tedious little problems in a code review.

When you're in the hotseat, try not to waste other people's time. Before you present your code for review, run a dummy review by yourself first. Imagine you're presenting your work to the others. You'll be surprised how many little flaws you'll filter out, and it will help you to be more confident in the real review. Don't rush out half-baked code and expect others to review the flaws out for you.

4. Compare this with an open source project's maintainer, who collates patches submitted by other hackers and integrates them into the main source tree, performing periodic software update releases.

Method in our Madness

Code reviews are a universally acknowledged technique, and have been around since people punched their programs into stacks of cards. We've looked at two review procedures in detail, but there are many subtle variants. Programming teams pick their review mechanism to suit the members and the nature of their work. (Poor teams perform no code review at all.)

These are some other common review methods and terms:

- **Fagan inspections**. A well respected process for formal reviews, much as described here, defined by Michael Fagan in his Defect Free Process [Fagan 76]. Fagan emphasises the importance of an ability to review, and shows how to improve review skills. Fagan inspections identify problems both with the work product, and with the process that created it.
- **Egoless programming**. Described by Weinberg in his 1971 book The Pyschology of Computer Programming [Weinberg 71], this is a timeless description of the critical attitude that makes reviews work. Programmers who aren't afraid of bugs in their code, and of others finding them, will generate better, safer, more correct software. A willingness for others to help find faults in your work is an essential attribute of the master programmer.
- **Shadowing**. This is a a halfway house between pair programming and code reviews. Each code module has a lead developer who works on the code. A shadow developer is also assigned; periodically they review the module with the lead. As design solidifies, the shadow developer verifies the decisions made. As the code fills out, the shadow reviews progress and makes constructive advice.

In more formal settings, the shadow is given authority to approve the code for release. No module can be integrated until the shadow developer agrees that it's ready for inclusion in the release build.

Reviewer

When reviewing code and making criticism, you must be sensitive. Comments must always be constructive, and not intended to lay blame. Do not launch personal attacks (you always do this…) on the author. Diplomacy is important here.

Code review is a peer process: every reviewer is considered equal. Seniority doesn't matter, and all views are considered. It is interesting that even the least experienced programmer will have something worth mentioning in a code review. And just as the author learns from the review, so may a reviewer.

Over time you will perform many, many reviews (especially if you perform integration reviews). Be careful that your review process doesn't become a mundane chore; it'll soon be an ineffective waste of everyone's time. Maintain a positive approach to your code reviewing. As a reviewer, always try to have something useful to say at each review. Sometimes this is easy, sometimes its very difficult to say anything interesting. But by forcing yourself to make comments you won't fall into the easy review rut, becoming a check-in 'yes man' who adds nothing to the process.

Code Perfection

We haven't yet considered what type of code will 'pass' review, and what code will 'fail'. It's beyond the scope of this article to describe what 'good code' looks like. As we look for bad code design and hunt software bugs, there are a few specific themes we can draw out. The reviewed code needs to be:

Correct

The code must meet all relevant standards and its requirements. Ensure that all variables are of the correct type (e.g. there is no chance of numeric overflow). Comments must be completely accurate. The code must meet any memory size or performance requirements (especially important for embedded platforms). Check that there is appropriate use of libraries, and that all function parameters are correct.

Complete

The code must implement the entire functional specification. It must have been integrated and debugged satisfactorily, and pass all test suites.

Well-Structured

Check that the implementation's design is sound, that the code is easy to understand, and that there is no duplication or redundant code. Look for any obvious cut-and-paste programming, for example.

[concluded at foot of next page]

Patterns in C – Part 4: OBSERVER

By Adam Petersen <adampetersen75@yahoo.se>

Managing dependencies between entities in a software system is crucial to a solid design. In the previous part we had a look at the open-closed principle. This part of the series will highlight another principle for dependency management and illustrate how both of these principles may be realized in C using the OBSERVER pattern.

Dependencies Arise

Returning to the examples used for the STATE pattern [2], they described techniques for implementing the behaviour of a simple digital stop-watch. Implementing such a watch typically involves the task of fetching the time from some kind of time-source. As the time-source probably will be tied to interactions with the operating system, it is a good idea to encapsulate it in an abstraction hiding the system specific parts in order to ease unit testing and portability. Similarly, the internals of the digital stop-watch should be encapsulated in a module of its own. As the digital stop-watch is responsible for fetching the time from the time-source (in order to present it on its digital display), it stays clear that there will be a dependency between the watch and the time-source. There are two obvious choices for the direction of this dependency.

Consider the Watch as a Client

It may seem rather natural to consider the watch as a client of the time-source. That is, letting the watch depend upon the time-source. Unfortunately, implementing the dependency in this direction introduces one obvious problem: how does the watch know if the time changes? The quick answer is: it doesn't. At least it doesn't unless it introduces some kind of polling mechanism towards the time-source. Just as important as it is to avoid premature optimization, one should also strive to avoid premature pessimization; even if the direction of the dependency seems correct, this solution is very likely to be extremely CPU consuming. In case the application needs to do more than updating a display, the problem calls for another solution.

Let the Time-Source Update the Watch

The potential capacity problem described above may easily be avoided by reversing the dependency. The time-source may simply notify the watch as soon as its time changes. This approach introduces a dependency from the time-source upon the watch.

```
#include "DigitalStopWatch.h"
#include "SystemTime.h"

static DigitalStopWatchPtr digitalWatch;
static SystemTime currentTime;

/* Invoked once by the application at start-
up. */
void startTimeSource()
{
digitalWatch = createWatch();

/*Code for setting up handlers for inter-
rupts, or the like, in order to get notified
each millisecond from the operating system.
*/
}

/* This function is invoked each millisecond
through an interaction with the operating
system. */
static void msTick()
{
/* Invoke a function encapsulating the knowl-
edge about time representation. */
currentTime = calculateNewTime();

/* Inform the watch that another millisecond
passed. */
notifyChangedTime(digitalWatch, &current-
Time);
}
```

Listing 1: Code for the time-source

The attractiveness of this approach lies in its simplicity. However, if scalability and flexibility are desired properties of the solution, the trade-offs are unacceptable. The potential problems introduced are best described in terms of the principles that this design violates.

[continued from previous page]

Predictable

There must be no unnecessary complexity, and no unexpected surprises. The code should not be self-modifying, must not rely on 'magic' default values, and not contain the subtle chance of infinite loops or recursion.

Robust

The code is defensive. Wherever possible the code should protect against detectable run time errors (divide by zero, number out of range errors, etc). Input should be checked (both function parameters and program input). The code handles all error conditions, and is exception safe. All appropriate signals are caught.

Data Checking

Bounds checking is performed on C-style array access. Other similarly insidious data access errors are avoided. Multithreaded code has correct use of mutexes. The return values of all system/library calls are checked.

Maintainable

The programmer has been wise in their use of comments. The code is kept under correct revision control. There is appropriate configuration information. The code formatting meets 'house standard'. It compiles quietly, without spurious warnings.

Beyond the Code Review

A review process is key to the production of any high quality item, and so is not solely useful for source code development. A similar review process is used for specification documents, lists of requirements, etc.

Conclusion

Code reviews are an essential part of the software development process. Just as an apprentice learns their trade from knowledge passed on, code reviews spread knowledge and teach coding capability. As more of a peer-to-peer than master-apprentice activity, they provide a learning opportunity for author and reviewer alike, and result in higher quality code.

Write your code to be reviewed; bear in mind that it's never just for you to read. Other people must be able to maintain it as well. The author is always accountable for the quality of their code.

Pete Goodliffe

References

[Humphry 97] Introduction to the Personal Software Process. Watts S Humphrey. Addison-Wesley, 1997. ISBN: 0201548097

[Humphry 98] The Software Quality Profile. Watts S Humphrey. In: Software Quality Professional, December 1998. Available from: http://www.sei.cmu.edu/publications/articles/quality-profile/

[Fagan 76] Design and code inspections to reduce errors in program development. Michael Fagan. In: IBM Systems Journal, Vol. 15, No. 3. 1976

[Weinberg 71] The Psychology Of Computer Programming. Gerald Weinberg. Van Nostrand Reinhold, 1971. ISBN: 0932633420

The Open-Closed Principle

Having a quick recap on the open-closed principle, it is summarized as "*Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification*" [4]. The code for the time-source above clearly violates this principle. In its current shape it supports only a single watch of one type. Imagine supporting other types of watches, for example one with an analogue display. The code for the time-source would, due to its hard-coded notification, explode with dependencies in all directions on all types of watches.

The Stable Dependencies Principle

During software maintenance or incremental development changes to existing code are normally unavoidable; even when applying the open-closed principle the design is just closed against certain modifications based upon assumptions by the original designer (it is virtually impossible for a software entity to be completely closed against all kinds of changes). The stable dependencies principle tries to isolate the impact of changes to existing code by making the recommendation that software entities should "*depend in the direction of stability*" [4].

In the initial approach, the watch itself fetched the time from the time-source. With respect to the stable dependencies principle, this was a better choice. A time-source is typically a highly cohesive unit and the more stable entity; simply encapsulating it in a proper abstraction, which hides the system specific details, makes it a good candidate to be packaged in a re-usable lower-level domain layer.

By having the time-source depend upon the higher-level digital watch, we violate the stable dependencies principle. This violation manifests itself by making the code for the watch difficult to update. Changes may actually have impact upon the code of the time-source itself!

Combining the dependency direction of the first approach with the notification mechanism of the second would make up an ideal design and the design pattern OBSERVER provides the extra level of indirection necessary to achieve the benefits of both solutions without suffering from any of their drawbacks.

OBSERVER

The OBSERVER pattern may serve as a tool for making a design follow the open-closed principle. *Design Patterns* [3] captures the intent of OBSERVER as "*Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically*". Applying the OBSERVER pattern to our example, extended with a second type of watch, the "*dependents*" are the different watches. *Design Patterns* [3] defines the role played by `TimeSource` as a "*concrete subject*", the object whose state changes should trigger a notification and update of the dependents, i.e. observers.

Implemenation Mechanism

In order to decouple the subject from its concrete observers and still enable the subject to notify them, each observer must correspond to a unique instance. The FIRST-CLASS ADT pattern [1] provides a way to realize this. However, as seen from the subject, all observers must be abstracted as one, general type sharing a common interface. In the C language, without language support for inheritance, generality is usually spelled `void*`; any pointer to an object may be converted to `void*` and back to its original type again without any loss of information. This rule makes a common interface possible.
The interface shown in Listing 2 declares a pointer to a function, which proves to be an efficient technique for implementing dynamic behaviour; a concrete observer attaches itself, together with a pointer to a function, at the subject. As the subject changes, it notifies the observer through the attached function.

```
typedef void (*ChangeTimeNotification)
   (void* instance,
   const SystemTime* newTime);

typedef struct
{
void* instance;
ChangeTimeNotification notification;
} TimeObserver;
```

Listing 2: Interface of the observers, `TimeObserver.h`

By using `void*` as abstraction, type-safety is basically traded for flexibility; the responsibility for the conversion of the observer instance back to its original type lies on the programmer. A conversion back to another type than the original may have disastrous consequences. Franz Kafka, although not very experienced in C programming, provided a good example of such behaviour: "*When Gregor Samsa woke up one morning from unsettling dreams, he found himself changed in his bed into a monstrous vermin*" [5]. Obviously, Gregor Samsa wasn't type-safe. In order to guard against such erroneous conversions, the `TimeObserver` structure has been introduced to maintain the binding between the observer and the function pointer. The implicit type conversion is encapsulated within the observer itself, as illustrated in the code in listing 3.

```
/* Include files omitted. */
struct DigitalStopWatch
{
Display watchDisplay;
/* Other attributes of the watch, e.g.
digital display. */
};

/* Implementation of the function required by
the TimeObserver interface. */
static void changedTime(void* instance,
   const SystemTime* newTime)
{
DigitalStopWatchPtr digitalWatch = instance;
assert(NULL != digitalWatch);

updateDisplay(digitalWatch, newTime);
}
```

Listing 3: A concrete observer implemented as a FIRST-CLASS ADT [1]

Before an observer can get any notifications, it has to register for them. Listing 4 declares the interface required by the observers.
These functions are implemented by the concrete subject, the `TimeSource`, which has to keep track of all attached observers. The example below uses a linked-list for that task. Attaching an observer corresponds to adding a copy of the given `TimeObserver` representation to the list. Similarly, upon a call to `detach`, the node in the list corresponding to the given `TimeObserver` shall be removed and that observer will no longer receive any notifications from the subject. Listing 5 illustrates this mechanism.

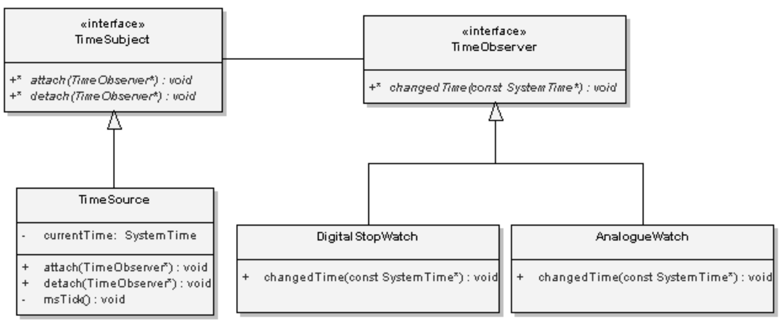

«interface»
TimeSubject
+* attach(TimeObserver*) : void
+* detach(TimeObserver*) : void
«interface»
TimeObserver
+* changedTime(const SystemTime*) : void
TimeSource
- currentTime: SystemTime
+ attach(TimeObserver*) : void
+ detach(TimeObserver*) : void
- msTick() : void
DigitalStopWatch
+ changedTime(const SystemTime*) : void
AnalogueWatch
+ changedTime(const SystemTime*) : void

Figure 1: OBSERVER pattern structure

```
#include "TimeObserver.h"

 void attach(const TimeObserver* observer);
 void detach(const TimeObserver* observer);
```

Listing 4: Interface to the subject, `TimeSubject.h`

```
#include "TimeSubject.h"

struct ListNode
{
   TimeObserver item;
   struct ListNode* next;
};

static struct ListNode observers;
static SystemTime currentTime;

/* Local helper functions for managing the
linked-list (implementation omitted). */

static void appendToList
   (const TimeObserver* observer)
{
   /* Append a copy of the observer to the
linked-list. */
}

static void removeFromList
   (const TimeObserver* observer)
{
   /* Identify the observer in the linked-list
and remove that node. */
}

/* Implementation of the TimeSubject interface.
*/

void attach(const TimeObserver* observer)
{
   assert(NULL != observer);
   appendToList(observer);
}

void detach(const TimeObserver* observer)
{
   assert(NULL != observer);
   removeFromList(observer);
}

/* Implementation of the original responsibility of
the TimeSource (code for initialization, etc
omitted). */
static void msTick()
{
   struct ListNode* node = observers.next;
   /* Invoke a function encapsulating the
knowledge about time representation. */
   currentTime = calculateNewTime();

/* Walk through the linked-list and notify every
observer that another millisecond passed. */
   while(NULL != node) {
      TimeObserver* observer = &node->item;
      observer->notification(observer-
>instance, &currentTime);
      node = node->next;
   }
}
```

Listing 5: Implementation of the subject, `TimeSource.c`

The code in Listing 5 solves our initial problems; new types of watches may be added without any modification to the TimeSource, yet these watches, our observers, are updated in an efficient way without the need for expensive polling.

Observer Registration

An additional benefit of implementing the OBSERVER pattern as a FIRST-CLASS ADT [1] is the combination of loose dependencies with information hiding. The client neither knows nor depends upon a subject. In fact, the client doesn't even know that the `DigitalStopWatch` acts as an observer because the functions for creating and destructing the ADT encapsulate the registration handling. The code in Listing 6, which extends Listing 3, illustrates this technique.

Subject – Observer Dependencies

The introduction of the OBSERVER pattern results in loose dependencies between the subject and its observers. However, no matter how loose, the dependencies cannot be ignored and an often overseen aspect of the

```
static void changedTime(void* instance, const
SystemTime* newTime)
{
/* Implementation as before (Listing 3). */
}

DigitalStopWatchPtr createDigitalWatch(void)
{
DigitalStopWatchPtr watch = malloc(sizeof
*watch);
if(NULL != watch){
/* Successfully created -> attach to the
subject. */
TimeObserver observer = {0};
observer.instance = watch;
observer.notification = changedTime;

attach(&observer);
}
return watch;
}

void destroyDigitalWatch(DigitalStopWatchPtr
watch)
{
/* Before deleting the instance we have to
detach from the subject. */
TimeObserver observer = {0};
observer.instance = watch;
observer.notification = changedTime;

detach(&observer);
free(watch);
}
```

Listing 6: Encapsulation of the registration handling

OBSERVER pattern is to ensure correct behaviour in case of changed registrations during a notification of the observers. The problem should be addressed in all OBSERVER implementations and is illustrated by investigating the notification loop coded earlier, as shown in Listing 7.

In case an observer decides to detach itself during a notification, the list containing the nodes may become corrupted. The solutions span between forbidding subject changes during notification and, at the other extreme, allow the subject to change and ensure it works.

The solution with forbidding subject changes during notification calls for a well-documented Subject interface. Further, the constraint may be checked at run-time using assertions, as shown in Listing 8.

The solution at the other side of the spectrum depends upon the actual data structure used to store the observers. The idea is to keep a pointer to the next observer to notify at file-scope. Attaching or detaching an observer now involves the possible adjustment of that pointer. By exclusively using this pointer in the notification-loop, the problem with unregistrations is

```
/* Walk through the linked-list and notify
every observer that another millisecond
passed. */
while(NULL != node) {
TimeObserver* observer = &node->item;
observer->notification(observer->instance,
&currentTime);
node = node->next;
}
```

Listing 7: Code extracted from Listing 5

solved. By choosing a strategy for new registrations, defining if the new observers are to be notified during the loop where they attach or not, the solution is complete. The extra complexity is rewarded with the flexibility of allowing observers to be added or removed during notification.

One Pattern, Two Models

In the example above, the part of the subject that changed (in our example the system time) was given to the observers as an argument to the notification-function. This technique is known as the push-model. The advantage of this model is its simplicity and efficiency; the data that changed is immediately available to the observers in the notification. A potential problem is the logical coupling between a subject and its observers; in order to deliver the correct data, the subject has to know about the needs of its observers.

This potential problem is eliminated by the other model used in OBSERVER implementations. This model, known as the pull-model, does not send any detailed information about what changed at the subject; the observers have to fetch that data themselves from the subject. In case the subject contains several large data structures, the efficiency of the pull-model may be improved by introducing an update protocol. Such a protocol specifies what changed while still putting the responsibility on the observers to fetch the actual data. A simple `enum` may serve well as an update protocol.

Consequences

The main consequences of applying the OBSERVER pattern are:

1. Introduces loose dependencies. As the subject only knows its observers through the Observer interface, the code conforms to the open-closed principle; by avoiding hard-coded notifications, any number and any types of observers may be introduced as long as they support the Observer interface. New behaviour, in the form of new types of observers, is added without modifying existing code. Further, the loose dependencies provide a way to communicate between layers in a sub-system. Design Patterns [3] recognizes this potential: "Because Subject and Observer aren't tightly coupled, they can belong to different layers of abstraction in a system. A lower-level subject can communicate and inform a higher-level observer, thereby keeping the system's layering intact." This property may serve as a tool for minimizing the impact of modifications by following the stable dependencies principle and yet enable a bidirectional communication between layers.
2. *Potentially complex management of object lifetimes*. As illustrated above, the FIRST-CLASS ADT pattern [1] simplifies the management by encapsulating the interaction with the subject. However, in case the subject is also implemented as a first-class object, the dependencies have to be resolved on a higher level and the client has to ensure that the subject exists as long as there are observers attached to it.
3. *May complicate a design with cascades of notifications*. In case an observer plays a second role as subject for other observers, a notification may result in further updates of these observers leading to overly complex interactions. Another problem may arise if an observer, during its update, modifies the subject resulting in a new cascade of notifications.
4. *Lowers the cohesion of the subject*. Besides serving its central purpose (in our example being a time-source) a subject also takes on the responsibility of managing and notifying observers. By merging two responsibilities in one module, the complexity of the subject is increased. This extra complexity is acceptable in case the loose dependencies, gained by introducing the OBSERVER pattern, provide significant benefits. Further, the subject may be simplified in cases where, during the life of the subject, there isn't any need to detach observers. In such a case, the detach function is simply omitted.
5. *Trades type-safety for flexibility*. The gist of the OBSERVER pattern is that the subject should be able to notify its dependents without making any assumptions about who they are. I.e. it must be possible to have observers of different types. The solution in this article uses void* as the common abstraction of an observer. The potential problem arises as the subject notifies its observers and passes them as void* to the notification functions. When converting a void-pointer back to a pointer of a concrete observer type, the compiler doesn't have any way to detect an erroneous conversion. This problem may be prevented by defining a unique notification function for each different type of observer in combination with using a binding such as the TimeObserver structure introduced above.

```
static int isNotifying = 0;

void attach(const TimeObserver* observer)
{
assert(0 == isNotifying);
/* Code as before. */
}

void detach(const TimeObserver* observer)
{
assert(0 == isNotifying);
/* Code as before. */
}

static void msTick()
{
struct ListNode* node = observers.next;

/* Ensure that no changes are done to the
subject during notification. */
isNotifying = 1;

while(NULL != node) {
/* Loop thorugh the observers as before. */
}
/* All observers notified, allow changes
again. */
isNotifying = 0;
}
```

Listing 8: Checking Subject constraints with assertions

Summary

This article illustrated how the loose coupling introduced by the OBSERVER pattern may serve as a way to implement modules following the open-closed principle. We can add new observers without modifying the subject. In fact, observers may even be replaced at runtime.

Further, as OBSERVER reverses the dependencies it allows lower-level modules to notify modules at a higher abstraction level without compromising layering. This property makes it possible to implement modules following the stable dependencies principle in cases where the lower layers in a system are the more stable and yet need to communicate with the higher layers.

However, when hard-coded notifications are enough, by all means stick to them; in case the loose coupling and extra level of indirection isn't needed, the OBSERVER pattern may just overcomplicate a design.

Next Time

We'll climb one step in the pattern categories and investigate the architectural pattern REACTOR. A REACTOR is useful in event-driven applications to demultiplex and dispatch events from potentially many clients.

Adam Petersen

References

1. Adam Petersen, "Patterns in C, part 1", C Vu 17.1
2. Adam Petersen, "Patterns in C, part 2: STATE", C Vu 17.2
3. Gamma, E., Helm, R., Johnson, R., and Vlissides, J, *Design Patterns*, Addison-Wesley
4. Robert C. Martin, *Agile Software Development*, Prentice Hall
5. Franz Kafka, *The Metamorphosis*

Acknowledgements

Many thanks to Tord Andersson, Drago Krznaric and André Saitzkoff for their feedback.

Qt 4.0 is Out!

Jasmin Blanchette <jasmin@trolltech.com>

Qt 4.0, the latest major version of Qt, is now available for download. Like previous Qt releases, Qt 4.0's primary goal is to allow C++ developers to create readable, maintainable, cross-platform GUI applications that look native on all platforms. Beyond that, the focus has been to make Qt even easier to learn and use, to increase Qt's performance, and to make multithreaded programming easier.

In the previous instalment of this series on GUI programming with Qt, we presented the new set of collection classes introduced with version 4.0. In this article, we will concentrate on some of Qt 4.0's architectural changes and on its powerful 2D drawing capabilities. But first, a note on licensing.

Qt's Dual-Licensing

Starting with Qt 4, the Windows version of Qt is available both under a commercial license and under the GNU General Public License (GPL), extending Qt's Open Source offer to cover all platforms supported by Qt. This means that if you want to build commercial applications using Qt, you must buy a commercial licence from Trolltech; if you want to build Open Source programs for Unix/Linux, Windows or Mac OS X, you can download the Qt Open Source Edition from `http://www.trolltech.com/`. Time-limited evaluation versions are available for commercial evaluators.

The availability of a Qt/Windows Open Source Edition this year is the last brick in the construction of our successful dual-licensing business model. In the early days, Qt/X11 was available as a binary-only package to Open Source developers—notably the developers of the K Desktop Environment (KDE) for Linux/Unix. The table below shows how we have made our licensing more flexible and extended it to more platforms through the years.

Year	Event
1998	Qt/X11 is released under the Q Public License (QPL), an Open Source license, to satisfy the needs of the KDE project
2000	Qt/ X11 is released under the GPL, in addition to the QPL, making it possible to write GPL software using Qt
2000	Qt/Embedded is released under the GPL
2001	Qt/Windows version 2.3 is made available as a binary-only package under a non-commercial license
2003	Qt/Mac is released under the GPL
2005	Qt/Windows is released under the GPL

Other Trolltech products, notably Qt Script for Applications (QSA) and Qtopia PDA Edition, are available under both commercial and Open Source licences.

Architectural Changes

Unlike previous Qt releases, Qt 4 is a collection of smaller libraries:

Library	Description
`QtCore`	Core non-GUI functionality
`QtGui`	Core GUI functionality
`QtNetwork`	Network module
`QtOpenGL`	OpenGL module
`QtSql`	SQL module
`QtXml`	XML module
`Qt3Support`	Qt 3 compatibility classes

`QtCore` contains tool classes like `QString`, `QList`, and `QFile`, as well as kernel classes like `QObject` and `QTimer`. The `QApplication` class has been refactored so that it can be used in non-GUI applications. It is split into `QCoreApplication` (in `QtCore`) and `QApplication` (in `QtGui`). This split makes it possible to develop server applications using Qt without linking in any unnecessary GUI-related code and without requiring GUI-related system libraries to be present on the target machine (e.g. Xlib on X11, Carbon on Mac OS X).

In addition, Qt 4 provides an extension library that applications based on Qt 3 called `Qt3Support`, that Qt applications can link against. This allows for more compatibility than ever before, without bloating Qt.

- Classes that have been replaced by a different class with the same name, such as `QListView`, and classes that no longer exist in Qt 4 are available with a 3 in their name (e.g., `Q3ListView`, `Q3Accel`).
- Other classes provide compatibility functions. Most of these are implemented inline, so that they don't bloat the Qt libraries.

New Technologies

Qt 4 introduces the following core technologies:

- Tulip, a new set of template container classes.
- Arthur, the Qt 4 painting framework.
- Interview, a model/view architecture for item views.
- Scribe, the Unicode text renderer with a public API for performing low-level text layout.
- Mainwindow, a modern action-based mainwindow, toolbar, menu, and docking architecture.

Qt 4 also includes the new *Qt Designer* user interface design tool, which can be integrated with popular IDEs. In addition, the following modules have been significantly improved since Qt 3:

- fully cross-platform accessibility module, with support for the emerging SP-API Unix standard in addition to Microsoft and Mac Accessibility.
- The SQL module, which is now based on the Interview model/view framework.
- The network module, with better support for UDP and synchronous sockets.
- The style API, which is now decoupled from the widgets, meaning that you can draw any user interface element on any device (widget, pixmap, etc.).
- Enhanced thread support, with signal-slot connections across threads and per-thread event loops.
- A new resource system for embedding images and other resource files into the application executable.

Arthur: The New Paint System

Qt 4 features a brand new paint subsystem, codenamed Arthur, that takes advantage of advances in 2D graphics support on modern window systems and the increased speed of desktop computers. Arthur introduces support for antialiasing, alpha blending, gradient filling, vector paths, and more. In this section, will show how to use these new features to make Qt applications look nicer.

Antialiasing

Aliasing when painting on a fixed-resolution device (such as a screen) is a visual distortion that occurs when the edges of a shape are converted into pixels. *Antialiasing* is a technique that reduces aliasing by using different colour intensities on the edges.

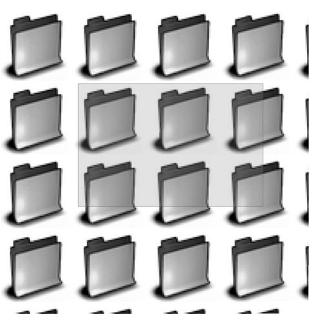
Antialiased shapes are usually more pleasant to look at. They can give the impression of a higher resolution than the screen actually uses, and make adjacent edges and intersection points clearer.

Qt 3 already supported antialiasing for screen fonts on systems that support it. Qt 4 adds supports for the other drawing shapes (lines, polygons, ellipses, etc).

Alpha Blending

Alpha blending is the process of compositing pixels with semi-transparency. This is done by extending the RGB triplet with an extra component, the "alpha channel", that specifies the level of transparency.

The image below shows a semi-transparent selection rectangle.



In Qt 3, only `QImage` and `QPixmap` supported alpha blending. `QImage` could support an 8-bit alpha channel and `QPainter` recognised this; `QPixmap` preserved the alpha channel for pixmaps that were created from `QImages`.

In Qt 4, `QPainter` supports alpha blending for all drawing operations, for filling, stroking, and text rendering. To make this possible, an alpha channel was added to `QColor`. For example, here's how we would draw a semi-transparent selection rectangle à la Windows XP:

```
painter.setPen(QColor(0, 0, 255, 191));
painter.setBrush(QColor(0, 0, 255, 63));
painter.drawRect(rect);
```

The fourth arguments to the `QColor` constructors specify the alpha channel. The alpha values extend from 0 (fully transparent) to 255 (opaque). In our example, the outline is 75% opaque and the fill is 25% opaque.

Gradient Filling

Gradient fills are defined by the interpolation between two or more colours, as opposed to a solid fill, which consists of a uniform colour. In Qt 4, they are implemented as a new `QBrush` fill pattern. For example:

```
QLinearGradient gradient(0, 0, 200, 100);
gradient.setColorAt(0.0, Qt::red);
gradient.setColorAt(1.0, Qt::blue);
painter.setBrush(gradient);
painter.drawEllipse(0, 0, 200, 100);
```

The code snippet defines a linear gradient fill that extends from red at (0, 0) to blue at (200, 100). The colours at points in between are interpolated linearly; points beyond the extremities are filled with the nearest extremity's colour. The points are expressed in logical painter coordinates. In addition to linear gradients, Qt 4 also supports radial and conical gradients.

This feature makes it easy to create visually pleasing effects such as shading on buttons. It can also be used in combination with alpha blending to fade colours in and out. For example, the following code snippet draws a gradient-filled rectangle on top of a pixmap to fade it out:

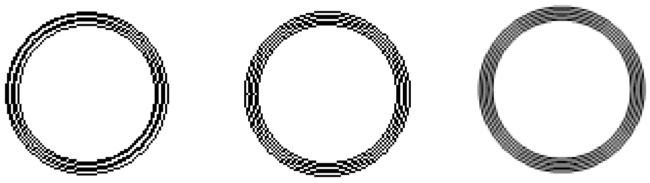
```
QRect rect = pixmap.rect();
painter.drawPixmap(rect, pixmap);

QColor transparent(255, 255, 255, 0);
QLinearGradient gradient(rect.topLeft(),
rect.bottomLeft());
gradient.setColorAt(0.0, transparent);
gradient.setColorAt(1.0, Qt::white);
painter.fillRect(rect, gradient);
```

Floating Point Based Painting

`QPainter` in Qt 4 introduces a floating point based API in addition to the existing integer-based API. Classes like `QPoint`, `QSize`, and `QRect` are now complemented by float-based classes such as `QPointF`, `QSizeF`, and `QRectF`.

This feature is often used in combination with antialiasing to increase the perceived resolution of what can be seen on screen. To illustrate this, we will study the example of concentric circles with alternating odd-even diameters.



The image on the left is drawn using `QPainter`'s `int`-based API. The image in the middle is drawn using the new float-based API. The image on the right uses the float-based API together with antialiasing.

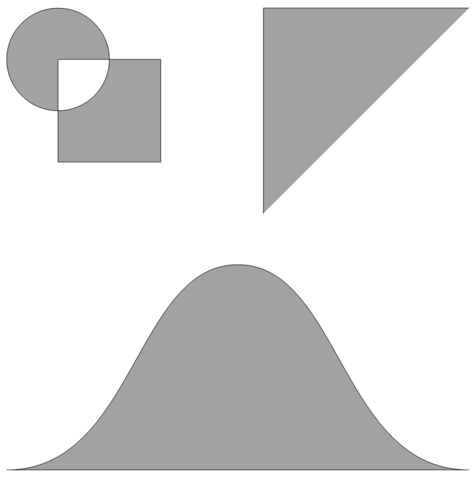
The circles with an even diameter on the left image don't have exactly the same centre point as the circles with an odd diameter; there is a half pixel offset between the odd circles' center and the even circles' centre. With the float-based API, we can specify "half pixel" coordinates, and with antialiasing, we get circles that look more accurate.

Another benefit of using floating point coordinates is that you can work in arbitrary resolutions. Previously, if you wanted to plot floating point data, you typically had to scale the coordinates and convert them to integers.

Painter Paths

Qt 4 introduces support for painter paths through the `QPainterPath` class. A *painter path* (also called "vector path") is a vectorial specification of a shape. Painter paths are the ultimate drawing primitive, in the sense that any shape (rectangle, ellipse, spline, etc.) or combination of shapes can be expressed as a path. QPainter uses painter paths internally when talking to the underlying paint engine.

A path specifies both an outline and an area. `QPainter::drawPath()` draws the outline using the current pen and fills the area with the current brush. Paths can also be used for clipping using `QPainter::setClipPath()`. For many applications, the basic drawing operations provided by `QPainter`



(`drawRect()`, `drawEllipse()`, etc.) are sufficient. Applications with more advanced 2D graphics, such as CAD applications, might use `QPainterPath` to represent their graphical data. `QPainterPath` can also be used to create scalable icons or more complex clip areas than are possible with `QRegion`.

A painter path is composed of two primitive elements: straight lines and cubic Bézier curves. You can compose shapes by connecting primitive elements together. A single path can contain multiple shapes, or *subpaths*.

You can create paths using the basic functions `moveTo()`, `lineTo()`, and `curveTo()`. For example:

```
QPainterPath path;
path.moveTo(10, 10);
path.lineTo(20, 10);
path.curveTo(30, 20, 15, 15, 10, 20);
```

You can call `moveTo()` at any time to start a new subpath, and `closeSubpath()` to connect the first and last points of the current subpath. `QPainterPath` also provides convenience functions for commonly used shapes: `addRect()`, `addEllipse()`, `addPolygon()`, `addText()`, and `addRegion()`.

[concluded at foot of next page]

Sharp as C

George Shagov <georgeshagov@yahoo.com>

1:18 For in much wisdom is much grief:
and he that increaseth knowledge increaseth sorrow.
KJV - Ecclesiastes

In the beginning was … a word.

And the word was … an algorithm!? Or should I say al-khwarizm? What does wikipedia (`http://en.wikipedia.org/wiki/Main_Page`) say about the term algorithm?

> *"An algorithm (the word is derived from the name of the Persian mathematician Al-Khwarizmi), is a finite set of well-defined instructions for accomplishing some task which, given an initial state, will terminate in a corresponding recognizable end-state"*

Al-Khwarizmi? Citing: "*Abu Abdullah Muhammad bin Musa al-Khwarizmi, was a Persian scientist, mathematician, astronomer/astrologer, and author. He was probably born in 780, or around 800; and probably died in 845, or around 840.*" 1200 years!

What This Article is About and How to Read This

This article represents my own point of view at the general approach to software development and its architecture. In respect of your time, I'm going to hide all my thoughts I had and way I did, whether they were short or long and going to offer you just a final conclusions. Sometimes these conclusions might seem strange, but that is what I'm thinking, my personal opinion. In this article I'm doing nothing, but expressing my own point of view, does it make any sense to you or not, do you see any useful ideas here or think all is absolutely useless it is for you to decide.

The plan of the article is pretty simple: In 'How it should be' section I'm describing the general idea. If you find it interesting then going further in the section 'It is possible.' You will find the details of the realization. Everything else is no more than pros & cons of that approach, proposed in the 'How it should be.' section. If the idea, described in the 'How it should be' section seems pointless to you, it might spare your time to read no further.

How Should It Be

"...since brevity is the soul of wit,
And tediousness the limbs and outward flourishes,
I will be brief: your noble son is mad:"
POLONIUS, Hamlet, Prince of Denmark, W. Shakespeare

Whether it possible to draw an architecture of application in general? Somebody might say it depends on business. In my opinion it should look like Figure 1.

Business logic is to be written in script to make it as plain as possible. The business entities are whatsoever your business needs, like collections (vectors, maps, sets, etc.), logging systems, DB vendors (MSSQL, Oracle, SyBase, etc), IPC (DCOM, RPC, Sockets, pipes), threading systems (such as POSIX) and so on. All these entities should expose some kind of an plain interface which basically are getters and setters. These entities should be as simple in logic as possible and in general they should export either data or simple functionality.

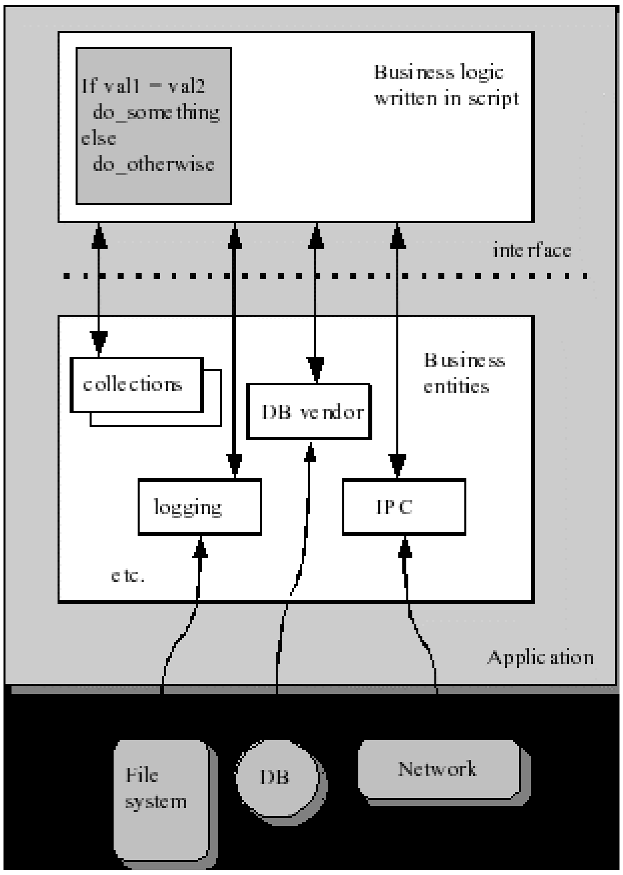


Figure 1

The business logic is written in some scripting language and portable also. If some entity is going to be changed (you are switching from SQL to Oracle for instance) the logic should not be changed, in the perfect case.

What I'm trying to say here is that any business logic and entities are to be separate. Let us see a classical sample:

```
int main()
{
  printf ("Hello world.\n");
  return 0;
}
```

[continued from previous page]

Advanced Text Rendering

In this last section, we will combine antialiasing, gradient filling, and painter paths to render the word "Arthur" in a cool way.

We start by setting up the painter to use antialiasing and by erasing the background:

```
QPainter painter(this);
painter.setRenderHint(QPainter::Antialiasing);
painter.fillRect(rect(), Qt::white);
```

We create a `QPainterPath` object that corresponds to the outline and area of the text "Arthur" in 200-point Times:

```
QFont timesFont("Times", 200);
timesFont.setStyleStrategy(QFont::ForceOutline);

QPainterPath path;
path.addText(0, 200, timesFont, "Arthur");
```

We fill the painter path's area with a gradient fill:

```
QLinearGradient pathGradient(0, 0, 0, 210);
pathGradient.setColorAt(0.0, QColor(219, 238, 188));
pathGraident.setColorAt(1.0, QColor(59, 156, 69));
painter.fillPath(path, pathBrush);
```

Finally, we use the `QPainterPathStroker` class to create a 2-pixel thick painter path for the outline and fill it with a somewhat darker gradient fill:

```
QPainterPathStroker stroker;
stroker.setWidth(2);
stroker.setJoinStyle(Qt::RoundJoin);
QPainterPath stroke = stroker.createStroke(path);

QLinearGradient strokeGradient(0, 0, 0, 210);
strokeGradient.setColorAt(0.0, QColor(150, 170,
140));
strokeGradient.setColorAt(1.0, QColor(0, 100, 20));
painter.fillPath(stroke, strokeBrush);
```

This is just a quick overview of the possibilities offered by Arthur. Check out the Arthur demos in Qt's demos directory for some mind-blowing effects.

Jasmin Blanchette

In this sample business logic is represented by means of C script (in general this is a script, since we have no idea how we are going to start it up) and business entities are only the one, this is the C-library (`libc`, `msvcrt` for instance), exposing plain 'exported' C-functions (`printf` in our case), this is the interface (see figure 2).

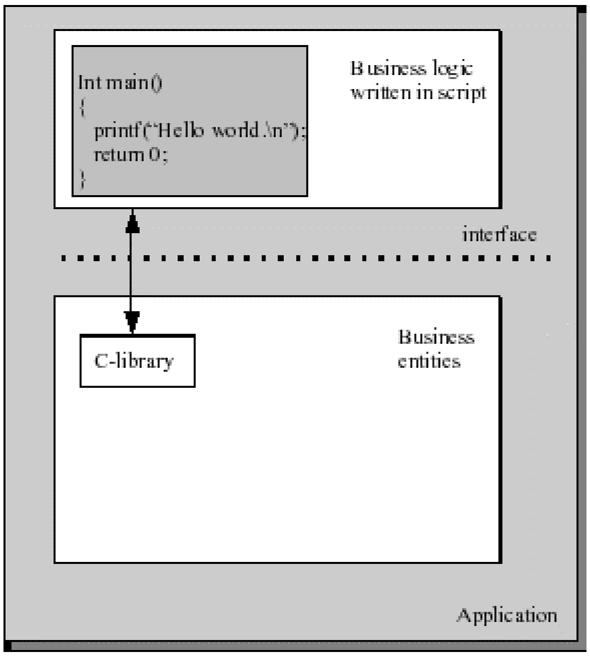


Figure 2

This approach goes contrary with the traditional OOP approach of development. Since OOP puts together an object and its functionality, this approach does otherwise. It is even thus. Keeping things clean and trying to save some time I would dare to say that, OOP worked out its resource and it is … dead.

Now I'm saying that developing business entities is not as painful as the development of business logic algorithms. Construction of business algorithms is a much more peculiar, painful, nervous process and takes much more time and resources than anything else.

Therefore **business logic is to be written in plain script, and this script is to be changeable at run-time, without any recompilation**. My basic objective is that business logic should not be as a 'sacred ground', once-working-never-changed. Its to be 'playable' whenever it is required, especially on development/QA stages, **this logic/script can be changed in run-time, with no any commits/check-ins to be done, no rebuilding, no restarting, or any other annoying procedures, just simple changing the script should immediately impact the running system**. Let me guess, you say impossible, or if it be possible – too complicated.

It Is Possible

And it is not so complicated. This section will show how it works. (The sample code is written for the Microsoft Windows platform)

This is an application tree:

```
+—-c_dispatcher
+—-Debug
+—-frontend_app
+—-include
+—-my_script
+—-my_script_c_proxy
+—-my_script_d_proxy
```

Inside the folder `fronend_app` is the main (console) application. There is only one file there: `frontend_app.cpp`

```
// frontend_app.cpp
// (c) George Shagov, 2005
#include <windows.h>
#include "..\\include\my_structs.h"

typedef int (__cdecl *MYFARPROC)
            (int nArg,
             char* pString,
             SMyStructure* pMyStruct);
```

```
int main(int argc, char* argv[])
{

  HMODULE hMyScript = LoadLibrary
        ("my_script_d_proxy.dll ");
  MYFARPROC pProcSource = (MYFARPROC)
        GetProcAddress
        (hMyScript, "c__my_entry_point");

  SMyStructure myStruct;

  myStruct.m_nVal = 0;
  strcpy(myStruct.m_sString, "");
  /*
  * calling for entry point.
  * directly
  */
  char sMyString[32];
  strcpy(sMyString, "My string here.");
  pProcSource(argc, sMyString, &myStruct);

  return 0;
}
```

As you can see here, it gets an address of entry point of the script and executes it.

The script itself might be found inside `my_scipt` folder, the file name: `my_script.c_`. There are some additional files there: `my_script.gnrtd.c` `my_script.gnrtd.h`, these ones are to be generated from `my_script.c` (below).

```
// my_script.c_
// (c) George Shagov, 2005

/*********************************************
*
* this file is automatically generated from
* my_script.c_    do not modify it
*
*********************************************/
#include <stdio.h>
#include <string.h>
#include "..\\include\\my_structs.h"
#include "my_scri"pt.gnrtd.h"

int c__get_value_1_impl(char* pString)
{
  return 1;
}

int c__get_value_2_impl(int nArg)
{
  return 2;
}

int c__call_in_case_varables_are_equal_impl
    (SMyStructure* pMyStruct)
{
  pMyStruct->m_nVal = 0;
  strcpy(pMyStruct->m_sString, "equal");
  return 0;
}

int c__call_in_case_varables_are_not_equal_impl
      (SMyStructure* pMyStruct)
{
  pMyStruct->m_nVal = 0;
  strcpy(pMyStruct->m_sString, "not equal");
  return 0;
}

int c__re_entry_impl(int nArg, char* pString,
                     SMyStructure* pMyStruct)
```

```
{
  int nVar1 = c__get_value_1(pString);
  int nVar2 = c__get_value_2(nArg);

  if (nVar1 == nVar2)
  {
    c__call_in_case_varables_are_equal
          (pMyStruct);
  }
  else
  {
    c__call_in_case_varables_are_not_equal
         (pMyStruct);
  }

  return 11;
}

int c__my_entry_point_impl(int nArg,
      char* pString, SMyStructure* pMyStruct)
{
  int nRet;

  printf("——————-\nbefore:\n");
  printf("nArg: %d, string: %s\n", nArg,
        pString);
  printf("pMyStruct->m_nVal: %d,
        pMyStruct->m_sString: %s\n",
        pMyStruct->m_nVal,
        pMyStruct->m_sString);

  nRet = c__re_entry(nArg, pString,
        pMyStruct);

  printf("++++++after:\n");
  printf("nArg: %d, string: %s\n", nArg,
        pString);
  printf("pMyStruct->m_nVal: %d,
          pMyStruct->m_sString: %s\n",
          pMyStruct->m_nVal,
          pMyStruct->m_sString);
  printf("ret: %d\n———————-\n", nRet);

  return nRet;
}
```

`c__my_entry_point_impl` is an entry point to be called from `frontend_app`. `my_script.gnrtd.c` is merely a copy of the original script. `my_script.gnrtd.h` represents the declarations.

As you can see `fronend_app` uses `my_script_d_proxy` library in order to make a call to `c__my_entry_point_impl`.

There are two files under `my_script_d_proxy` folder `my_script_d_proxy.gnrtd.c` & `my_script_d_proxy.gnrtd.h`, both these files are to be generated from original script (`my_script.c_`) also. `my_script_d_proxy.gnrtd.c` contains plugs for all the functions, written in the script, like this:

```
int c__re_entry_stub(int nESP, int nArg,
    char* pString, SMyStructure* pMyStruct)
{
  void* pArgs = 0;
  int nSize = 0;
  _asm
  {
    push eax; /* saving eax */
    mov eax, ebp; /* ebp points out at the
          parameters (as known) */
    add eax, 8; /* now eax points out at the first
         argument, which is nESP*/
    mov pArgs, eax;
    add pArgs, 4; /* since first argument is esp,
          but we need real argument here */
    mov eax, nESP;
    sub eax, pArgs; /* eax now has a phisical size
          of the stack */
    shr eax, 2; /* eax/4 - eax now has an amount
          of arguments put in the stack */
    mov nSize, eax; /* saving that size */
    pop eax; /* restoring eax */
  }
  return g_pDispatcherEntry("c__re_entry",
        pArgs, nSize);
}

int c__re_entry(int nArg, char* pString,
    SMyStructure* pMyStruct)
{
  int nESP;
  _asm
  {
    mov nESP, esp;
  }
  return c__re_entry_stub(nESP, nArg, pString,
        pMyStruct);
}
```

The assembly code remembers the pointer to the first argument, which was put in the stack, the count of argument in stack, and delivers a call to `c_dispatcher` library, which then exports the `g__c_dispatcher_entry_point` function.

The Code of `c_dispatcher.cpp`

```
// c_dispatcher.cpp
// (c) George Shagov, 2005
#include <stdio.h>
#include <windows.h>
#include "c_dispatcher.h"

static HINSTANCE s_hCSource = NULL;
static HINSTANCE s_hProxy = NULL;
typedef int (__cdecl *MYFARPROC)();

MYFARPROC GetMyProcAddress(
    const char* pFunctionName)
{
  char pFile[128];
  char pFnName[128];
  sprintf(pFile, "my_script.%s_impl.c_",
        pFunctionName);
  sprintf(pFnName, "%s_impl", pFunctionName);
  FILE* f = fopen(pFile, "r");
  if (f)
  {
    fclose(f);
    return (MYFARPROC)GetProcAddress(s_hProxy,
          pFnName);
  }
  else
    return (MYFARPROC)GetProcAddress
          (s_hCSource, pFnName);
}

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD ul_reason_for_call,
                       LPVOID lpReserved)
{
  switch (ul_reason_for_call)
  {
    case DLL_PROCESS_ATTACH:
      s_hCSource = LoadLibrary
            ("my_script.dll");
      s_hProxy = LoadLibrary
            ("my_script_c_proxy.dll");
    break;
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    break;
    case DLL_PROCESS_DETACH:
      FreeLibrary(s_hCSource);
      FreeLibrary(s_hProxy);
    break;
}
return TRUE;
}

// This is an example of an exported function.
C_DISPATCHER_API int
g__c_dispatcher_entry_point(
    const char* pFunctionName,
    const void* pArguments,
    int nArgumentsCount)
{
MYFARPROC pProc =
GetMyProcAddress(pFunctionName);
void* pStack = 0;
if (nArgumentsCount)
{
_asm
{
mov ecx, nArgumentsCount;
loop_start_01:
push 0;
loop loop_start_01;
mov pStack, esp;
}
memcpy(pStack, pArguments, nArgumentsCount*4);
int nRet = pProc();
_asm
{
mov ecx, nArgumentsCount;
loop_start_02:
pop eax;
loop loop_start_02;
}
return nRet;
}
else
return pProc();
}
```

As you can see here the case dispatcher has found a file `my_script.<function_name_impl>.c_` it delegates call `my_script_c_proxy` library, otherwise it defaults to `my_script.dll`, where the compiled script code is located. This actually is a substitution. Before the call it simulates the stack, knowing the pointer at the original position and its size, after the call – simple unwinding. Simple, right?

`my_script_c_proxy` library contains four files. (Here I should say, since we are going to change the code at run-time we need some kind of a C-interpreter. I took `cint`. `cint` is free C-interpreter, powerful enough and very suitable for this demo, yet there are couple of issues which means that some disadvantages in this demo implementation will be closely connected to this particular interpreter. `G__clink.c` `G__clink.h` – these files are generated from `my_script_d_proxy.gnrtd.h` (`my_script_d_proxy` folder) by `cint`, since `cint` during interpretation should not call to the script functions, but to stubs, implemented inside the `my_script_d_proxy` library, in order to be able to re-implement any function we need, not the whole script. The rest of the functions are to be called from `my_script.dll`. It's a little bit tricky. The file `my_script_c_proxy.gnrtd.c` contains stubs which look like this:

```
MY_SCRIPT_C_PROXY_API int
c__my_entry_point_impl(int nArg,
     char* pString, SMyStructure* pMyStruct)
{
  char tmp[128];
  int nRet;

  s__setup_cint();
  sprintf(tmp,"c__my_entry_point_impl((int)%d,
     (void*)0x%08lx, (SMyStructure*)0x%08lx);",
     nArg, (int)pString,pMyStruct);
     nRet = G__calc(tmp).obj.i; /* Call Cint
     parser */ return nRet;
}
```

`G__calc` is a `cint` function, which make a call to the script.

Well, actually, that's it.

Let us see how it works.

The context of `c_\Debug` folder (after getting the project built) looks like this:

```
C_dispatcher.dll
frontend_app.exe
my_script.dll
my_script_c_proxy.dll
my_script_d_proxy.dll
```

Starting the application we get:

```
before:
nArg: 1, string: My string here.
pMyStruct->m_nVal: 0, pMyStruct->m_sString:
++++++after:
nArg: 1, string: My string here.
pMyStruct->m_nVal: 0, pMyStruct->m_sString:
not equal
ret: 11
———-
```

This is what produced by the compiled script, and now located in the `m_script.dll` library.

In `Debug` folder we are creating an empty file: `my_script.c__get_value_1_impl.c_`. The existence of this file will be a sign to the dispatcher that there is a substitution for the `c__get_value_1_impl` function. We should create the `my_script.c_` file also, within the next content: (the presence of two files is that disadvantage I referred to earlier caused by `cint`).

```
// my_script.cpp : Defines the entry point for
the DLL application.
//

#include <stdio.h>
#include "..\\include\\my_structs.h"

int c__get_value_1_impl(char* pString)
{
  pString[1] = 'X';
  printf("c__get_value_1 ==>> str: %s\n",
        pString);
  return 2;
}
```

The contents of the Debug folder looks like this:

```
C_dispatcher.dll
frontend_app.exe
my_script.c_
my_script.c__re_entry_impl.c_
my_script.dll
my_script_c_proxy.dll
my_script_d_proxy.dll
```

Restarting application gives the result:

```
———-
before:
nArg: 1, string: My string here.
pMyStruct->m_nVal: 0, pMyStruct->m_sString:
c__get_value_1 ==>> str: MX string here.
++++++after:
nArg: 1, string: MX string here.
pMyStruct->m_nVal: 0, pMyStruct->m_sString:
equal
ret: 11
———-
```

Now let us try to re-implement two functions. For this purpose we are creating the second file: `my_script.c__re_entry_impl.c_`, in order to signalize the dispatcher, and modifying the script.

```
// my_script.cpp : Defines the entry point for
// the DLL application.
#include <stdio.h>
#include "..\\include\\my_structs.h"

int c__get_value_1_impl(char* pString)
{
  pString[1] = 'X';
  printf("c__get_value_1 ==>> str: %s\n",]
        pString);
  return 2;
}

int c__re_entry_impl(int nArg, char* pString,
    SMyStructure* pMyStruct)
{
  printf("\"I'll not be juggled with.\nTo
      hell, allegiance! Vows, to the
      blackest devil!\nConscience and grace,
      to the profoundest pit!\nI dare
      damnation. To this point I stand,\"\n");
  printf("...for this is script\n");

  int nVar1 = c__get_value_1(pString);
  int nVar2 = c__get_value_2(nArg);

  if (nVar1 == nVar2)
  {
    c__call_in_case_varables_are_equal
          (pMyStruct);
  }
  else
  {
    c__call_in_case_varables_are_not_equal
          (pMyStruct);
  }

  return 11;
}
```

The result:

```
——————-
before:
nArg: 1, string: My string here.
pMyStruct->m_nVal: 0, pMyStruct->m_sString:
"I'll not be juggled with.
To hell, allegiance! Vows, to the blackest
devil!
Conscience and grace, to the profoundest pit!
I dare damnation. To this point I stand,"
...for this is script
c__get_value_1 ==>> str: MX string here.
++++++after:
nArg: 1, string: MX string here.
pMyStruct->m_nVal: 0, pMyStruct->m_sString:
equal
ret: 11
———————-
```

Now a little bit about parameters or arguments of the functions. I might see already that the string 'My string here' has been changed to 'MX string here', It has been done by means of `c__get_value_1_impl` and re-implemented in the script. We are able to do the same with structures. Creating a new file: `my_script.c__call_in_case_varables_are_equal_impl.c_` and adding next function to the script:

```
int c__call_in_case_varables_are_equal_impl(
    SMyStructure* pMyStruct)
{
  pMyStruct->m_nVal = 0;
  strcpy(pMyStruct->m_sString, "– EQUAL –");
  return 0;
}
```

The result:

```
——————-
before:
nArg: 1, string: My string here.
pMyStruct->m_nVal: 0, pMyStruct->m_sString:
"I'll not be juggled with.
To hell, allegiance! Vows, to the blackest
devil!
Conscience and grace, to the profoundest pit!
I dare damnation. To this point I stand,"
...for this is script
c__get_value_1 ==>> str: MX string here.
++++++after:
nArg: 1, string: MX string here.
pMyStruct->m_nVal: 0, pMyStruct->m_sString: –
EQUAL –
ret: 11
———————-
```

By now the contents of the `Debug` folder looks like this:

```
c_dispatcher.dll
frontend_app.exe
my_script.c_
my_script.c__call_in_case_varables_are_equal_impl.c_
my_script.c__get_value_1_impl.c_
my_script.c__re_entry_impl.c_
my_script.dll
my_script_c_proxy.dll
my_script_d_proxy.dll
```

It works. As you can see:

1. It is possible to change (or rather to say substitute) the code (script) on run-time, no recompilation required.
2. It is not a hard task.

Performance

Yes of course using script instead of native code does mean significant loss of performance, yet there are two things to say:

- In the systems where performance is a key point (such as real-time systems), no substitution is to be allowed. It means there should not be any dispatcher library and all the calls to be compiled as direct ones and linked during the compilation. In this approach there will not be any losing of performance. Yet in development for QA where possibility for substitution is highly required but performance does not play a significant role, this approach will be applicable.
- In general, performance is not a key point. In this case If we need substitution right in production. It is possible to do that without significant lose of performance. In order to do that we should:
  1. Create and compile a separate library, let it have a name `my_script_subst.dll`. This library would contain the re-implementation of these functions which we need to substitute.
  2. Create and compile an additional proxy library, let it have a name `my_script_s_prioxy.dll` which should look like `my_script_c_prioxy.dll`, save that all the calls will be delegated not to `cint`, but to `my_scipt_subst.dll` (see step 1)
  3. Modify the dispatcher so that it should know what `my_script_s_prioxy.dll` is.

I didn't do that just in order not to overload the code. If the basic idea is understandable, the rest is but technique.

Cons and Pros

Had I patience and time I would write a book here, or two. Yet in brief.

Disadvantages.

- The build procedure becomes more complicated, additional parsing is required.
- There should be an interpreter supplied
- Read the performance section
- Using C as a script might cause some problems, since C, by default, has a direct access to memory and has no mechanism of automatic unwinding, which may potentially cause leaks. Yet, that should be a C-script, not C, it means that all functions which uses access to memory should be exposed as entities.

[concluded at foot of next page]

Transactions with Delegates in C#

Matthew Skelton <matthew.skelton@gmail.com>

Introduction

The Delegate type is a powerful feature of C#. This article introduces delegates, and shows how they can be used to solve a common problem: ensuring that a series of database updates occur as a single operation by use of a transaction. The solution presented is flexible and minimises duplication of code.

Transactions

Transactions are essential for maintaining data integrity within computer systems; they seek to guarantee that logically dependent data are amended together in (effectively) a single operation. If an error occurs during that operation, the entire operation fails, and no changes are persisted. Indeed, the `ACID ()` properties of a database rely heavily on correctly implemented transactions[1].

However, the application developer should not be forced to memorise the details of transaction-handling code in order to perform 'ACID-ic' data updates: transaction code should be transparent, and not interfere with core application logic.

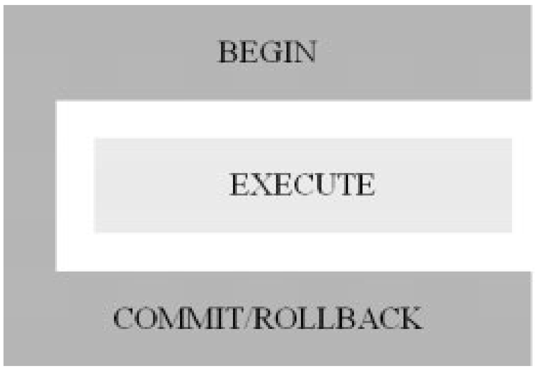


Figure 1 – Generalised transaction logic. The EXECUTE stage is logically separate from the BEGIN and COMMIT/ROLLBACK stages.

The 'boiler-plate' transaction code (i.e. that which is generic) would best be hidden behind a programming interface that feels 'natural'; ideally, we would have just a single point in the code for handling transactions. With the complex details of transactions hidden, programmers are free to concentrate on application logic.

We need a solution which:

1, Minimises proliferation of transaction code.
2. Hides the details of transaction code
3. Allows data update code to be defined separately from transaction code.

Transactions and DataSets in C#

Transactions in C# [4, 7] are simplified by use of the Framework Class Library (FCL) System.Data.IDbTransaction interface, which provides Commit() and Rollback() methods, and a Connection property, allowing the transaction to take place in the context of a particular database connection. This interface allows us to use the same database update logic with different database types and transaction models.

Potential Solutions

The programmatic nature of transactions can be summarised as:

1. BEGIN transaction
2. EXECUTE updates in context of transaction
3. COMMIT/ROLLBACK transaction depending on result of EXECUTE.

The BEGIN and COMMIT/ROLLBACK actions represent the 'boiler-plate' code, whereas EXECUTE is dependent on specific data update requirements. Figure 1 shows this relationship.

Here are some possible schemes for using transactions:

1. Wrap BEGIN and COMMIT/ROLLBACK around every EXECUTE operation (Figure 2). This provides flexibility in the definition of EXECUTE, but at the expense of gratuitous code duplication, with associated maintainability problems: if changes are needed to the code for BEGIN or COMMIT/ROLLBACK, those changes must be propagated throughout the application code.
2. Derive a group of data update classes from a common base class. Place the BEGIN and COMMIT/ROLLBACK code within a method on the base class, and have it call an overridden virtual method on derived classes to perform the EXECUTE logic. The limitation of this scheme is that a type hierarchy is imposed on data update code, which may not fit with existing class models, and which will likely lead to 'brittle' code and inflexibility for programmers. This is shown in Figure 3 – the EXECUTE methods are restricted in functionality by being derived from the same base class.
3. Use the Automatic Transactions feature supported by .NET [12]. This makes use of COM+/MTS technologies, and requires some specific code and application modifications that may not be applicable or suitable in many cases, and does not provide precise control over the transaction used for a given method.

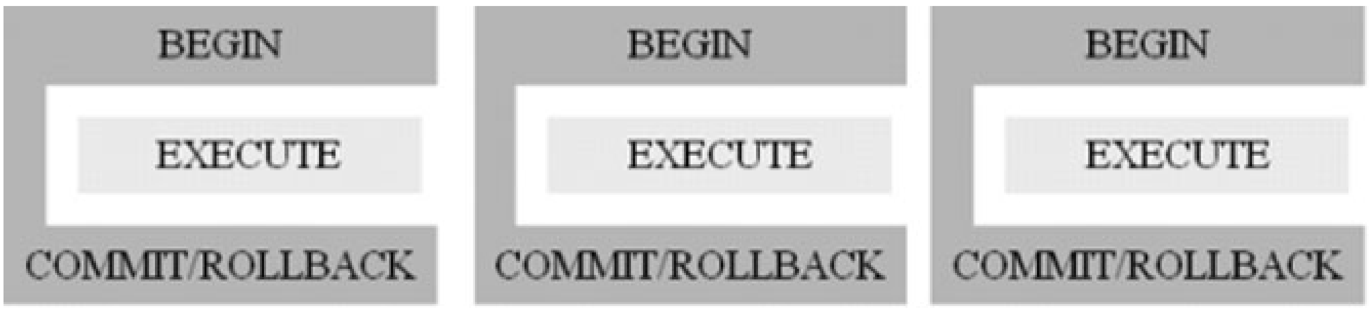


Figure 2 – Transaction code explicitly surrounding each data update.

1. Here, 'transaction' generally refres to a one-phase operation between a single database and a single application. Distributed Transactions (between multiple data sources and/or applications) are out of the scope of this discussion.

[continued from previous page]

Benefits

- Clarity. OOP code is much less readable than plain script and this clarity is the main goal.
- Ability to change the business logic run-time.
- Control. Just think what we able to do having all entry-points in our hands.

TODO

A lot.

- There should be a suitable C-interpreter
- Parsing procedure
- See the second clause described in the 'Performance' section
- Dispatcher. Yes of cause the way it is implemented is not applicable to real system. There should be a map of functions which is to be updated in separate thread, according to timestamp of the modification
- And so on…

George Shagov

A Solution: C# Delegates

The ECMA specification [4] for C# introduces the delegate type thus:

Delegates enable scenarios that some other languages have addressed with function pointers. However, unlike function pointers, delegates are object-oriented and type-safe.[2]

A delegate is declared as follows:

```
public delegate void SomeDelegate(int i);
```

The delegate SomeDelegate expects a single int parameter and returns no value. Note the keyword `delegate`. The delegate is invoked as follows:

```
public class SomeClass
{
  // ...
  // Signature matches that of delegate
  public void SomeMethod(int i) {
     /* do     something here */ }
  // ...
}
public class AnotherClass
{
  public void CallDelegate(int i)
  {
    SomeClass c = new SomeClass();
    // Wrap the method as a delegate
    SomeDelegate d = new
        SomeDelegate(c.SomeMethod);

      #// Invoke the delegate
    d(i);
  }
}
```

The delegate acts a typed 'wrapper' around another method. Following instantiation, the delegate object is 'free' of the class in which its wrapped method is defined; the delegate can be treated as (and in fact is) a first-class object in its own right. For example, the delegate can be stored in a container or list of some sort for later processing, if required. So, delegates can be considered as akin to type-safe function pointers (or somewhat analogous to function objects in C++ and Python).

By placing all data update code inside delegate methods, it is possible to trigger all data updates in the same generic way: namely, by invoking the appropriate delegates. The parameters expected by one update method are identical to (or compatible with) those expected by all other update methods. We can thus guarantee that every delegate-based update method has access to the required `IDbTransaction` reference to use during data updates. Referring back to Figure 1 and the three stages of a transactional data update, we now have a way to link any EXECUTE step (contained within a delegate) to the transaction associated with our BEGIN and COMMIT/ROLLBACK logic. See Figure 4.

However, C# delegates sport another feature which is particularly useful in the context of this articleiii: they have 'multi-cast' abilities. Delegates of the same type can be 'combined' or concatenated (using the overloaded operator +=), producing a new delegate object which will – when invoked – call each of the original delegates in turn.

This multi-cast feature can be used to chain together a sequence of data update calls, without the need explicitly to manage the group of delegates in the chain. For example,

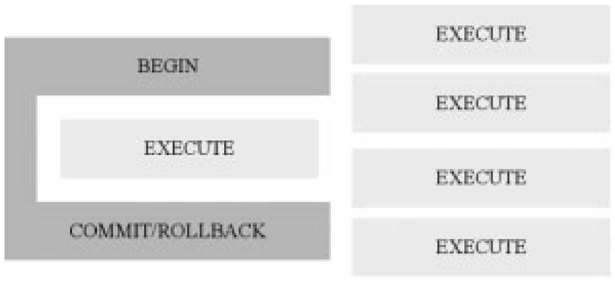


Figure 3 – A base class handles the BEGIN and COMMIT/ROLLBACK logic, with overridden EXECUTE methods containing data update logic. Data update methods are limited to a single class hierarchy

data updates might be queued on a background thread, and then submitted together as a batch. The code invoking the multi-cast delegate need not know how many individual delegates will be invoked; the invoked delegate handles this automatically.

Delegates: Recapitulation

Delegates will allow us to place our database update code in any class we choose, providing we arrange for the method signature of the update methods to match that of a particular delegate. We can pass into the delegate method the `IDbTransaction` reference for use during data update.

A Working Example

The following code demonstrates the ideas presented so far in working code.

First we declare our delegate which expects two parameters: `sender` and `args` (details follow later).

```
public delegate void
TransactionedOperationDelegate
    (TransactionWrapper sender,
     TransactionArgs args);
```

Next we declare a lightweight class to represent a database update operation:

```
public abstract class TransactionedOperation
{
  public TransactionedOperationDelegate Execute;
}
```

We must derive sub-classes from this class, as it is marked `abstract`, and therefore not instantiable. It holds a delegate reference in its `Execute` field. We will return to this shortly.

Referring to the basic transaction model (see Potential Solutions, above; Figure 1), we can map out the method required to implement the BEGIN and COMMIT/ROLLBACK logic.[4]

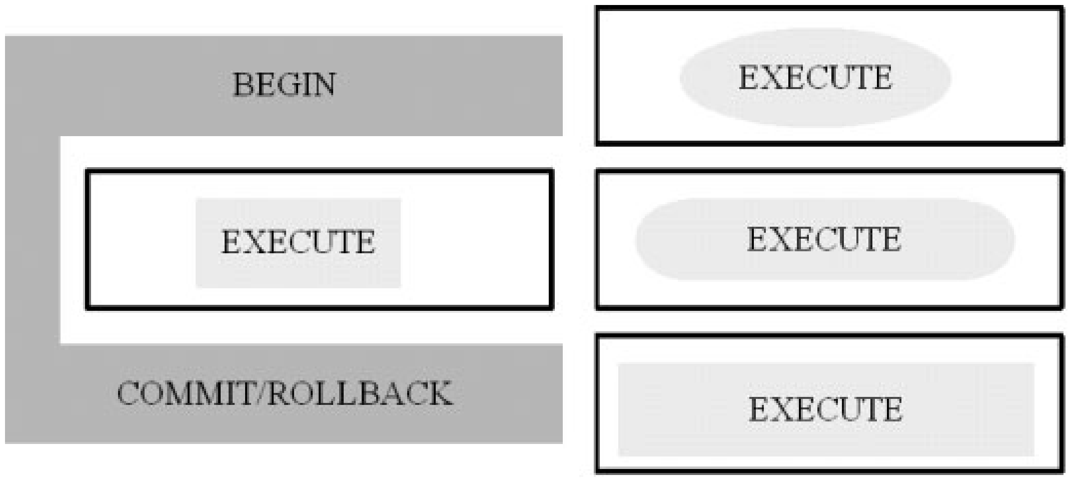


Figure 4 – Transactions using Delegates. The delegate EXECUTE methods can be invoked in a generic way (outer box) but may belong to disparate classes and contain very different logic.

2. The ECMA spec also notes: *The closest equivalent of a delegate in C or C++ is a function pointer, but whereas a function pointer can only reference static functions, a delegate can reference both static and instance methods. In the latter case, the delegate stores not only a refreence to the method's entry point, but also a reference to the object instance on which to invoke the method.*

3. Delegates are also used to implement anonymous methods in C# 2.0 [13], but that subject is beyond the scope of this article.

4. Error handling omitted for brevity.

The various data update operations passed to this method should be treated as a single, atomic operation:

```
// Make all operations into one atomic operation
public void MakeAtomic
    (TransactionedOperation[] operations)
{
  // Connection is set elsewhere
  this.Connection.Open();
  try
  {
    // BEGIN
    using (IDbTransaction transaction =
        this.Connection.BeginTransaction())
    {
      try
      {
      /// Prepare, then call each operation
      /// Any exceptions will abort the transaction

        foreach (TransactionedOperation
            operation in operations)
          {
            // EXECUTE
            // Invoke the delegate here!
            // i.e. operation.Execute(...)
            // ...
          }
          // COMMIT...
          transaction.Commit();
      }
      catch (Exception exception)
      {
        // ... or ROLLBACK
        transaction.Rollback();

        // TODO: Log the exception here

        // Re-throw exception
        throw;
      }
    }
  }
  finally
  {
    this.Connection.Close();
  }
}
```

The code as shown implements the BEGIN and COMMIT/ROLLBACK stages of the transaction logic. What is missing is the EXECUTE stage; that is, the invocation of the data update delegate contained in each `TransactionedOperation` in the `operations` array. What sort of parameters should we pass to the delegate? We clearly cannot make any sensible generalised assertions about the kind of parameters that might be needed by any given data update method. After all, one of the goals of using delegates is to allow data update code to reside in any arbitrary class; locking down the parameters would negate that flexibility.

The solution is to allow calling code to ‘hook on’ parameters to the `TransactionedOperation` parameter, by deriving a sub-class (as mentioned above), and defining new fields[5] to store the required parameters. For example:

```
public class CustomTransactionedOperation :
TransactionedOperation
{
  // hook arbitrary data...
  public CustomTransactionedOperation
      (CustomDataSet dataSet)
  {
    this.Example = dataSet.ExampleData;
  }
  public readonly string Example;
}
```

5. Read-only fields are used instead of the 'better practice' properties purely to minimise code bloat.

A sub-class `CustomTransactionedOperation` is derived from the base class `TransactionedOperation`. A delegate method can now access the `Example` field of the operation parameter.

Remember that the delegate method takes two parameters: a reference to the `TransactionWrapper` calling the delegate (the ubiquitous ‘sender’), and a parameter of type `TransactionArgs`. We use this second parameter to pass essential information to the delegate method:

```
public class TransactionArgs : EventArgs
    // for convenience
{
  public TransactionArgs(
      IDbConnection connection,
      IDbTransaction transaction,
      TransactionedOperation operation)
  {
    this.Connection = connection;
    this.Transaction = transaction;
    this.Operation = operation;
  }
  public readonly IDbConnection Connection;
  public readonly IDbTransaction Transaction;
  public readonly TransactionedOperation
      Operation;
}
```

The `CustomTransactionedOperation` (with its custom data fields) can be attached to the `TransactionArgs` parameter, thereby allowing arbitrary information to be passed to the delegate method.
Our code for invoking an operation delegate therefore looks like this:

```
// Connection is an IDbConnection reference
// transaction is an IDbTransaction reference
// operation is a TransactionedOperation
// reference
TransactionArgs args = new TransactionArgs(
    Connection, transaction, operation);
operation.Execute(this, args);
```

It might seem slightly strange that we use the `operation` parameter twice (once to allow us to call the delegate, and a second time to pass custom parameters to the delegate inside `args`): this is largely for convenience, although also allows for flexibility in the management of update operations.

All that is left now is to prepare the `TransactionedOperation` and use the `TransactionWrapper` class to make the data update operation(s) atomic:

```
// Some arbitrary update code, somewhere...
public void UpdateCustomDetails(CustomDataSet
dataSet)
{
  // Connect here to a SQL Server database -
  // could change
  IDbConnection connection = new
      System.Data.SqlClient.SqlConnection
      ( /* connection string */ );
  TransactionWrapper transactionWrapper = new
      TransactionWrapper(connection);

  // Hook the data onto the operation
  // parameters
  CustomDataUpdate customDataUpdate = new
      CustomDataUpdate();
  CustomTransactionedOperation operation = new
      CustomTransactionedOperation(dataSet);
  operation.Execute += new
      TransactionedOperationDelegate(
      customDataUpdate.UpdateData);
/* chain another delegate here if required:
operation.Execute += new
    TransactionedOperationDelegate(
    SomeOtherDelegateMethod);
*/
  // wrap in transaction
  transactionWrapper.MakeAtomic(operation);
}
```

The delegate used here is the `UpdateData()` method of a `CustomDataUpdate` instance. This is the method that will make use of the `IDbTransaction` transaction to perform atomic updates. Other update methods that expect the same dataset[6] can be chained on to the first delegate (shown in comments). Each delegate in the multi-cast chain will receive the same arguments. The `UpdateData()` method could look like this:

```
public class CustomDataUpdate
{
// The delegate method
public void UpdateData(TransactionWrapper
sender, TransactionArgs args)
{
// Expect special operation type
CustomTransactionedOperation operation =
args.Operation as
CustomTransactionedOperation;

SqlCommand sqlCommand = new SqlCommand();
sqlCommand.Transaction = args.Transaction as
System.Data.SqlClient.SqlTransaction;
sqlCommand.Connection = args.Connection as
System.Data.SqlClient.SqlConnection;

// Use custom params from operation object
sqlCommand.CommandText = operation.Example; //
etc.

sqlCommand.ExecuteNonQuery();
}
}
```

Clearly, a production system would employ somewhat more sophisticated logic, but the principle remains!

Roundup

The code presented above achieves the three goals set out at the beginning of this article: there is no proliferation of transaction code (only a single method, `TransactionWrapper.MakeAtomic()` holds such code[7]); data update methods need only use the `IDbTransaction` reference when calling into the database (no other details are needed); and data update methods can be defined in arbitrary locations, and can require arbitrary data, and all participate in transactions in the same way. See Figure 4.

The scheme provides additional benefits, such as the ability to log all details of a transaction at a single point (in the `TransactionWrapper`).

Finally, the scheme implements a modified version of the Command Pattern [1] with the `operation.Execute()` method [3] functionality, and the `TransactionWrapper` is arguably an implementation of the Façade Pattern [2].

Observations and Notes

In this scheme all database updates take place synchronously within the same `IDbConnection` context. Use of a single connection is normally essential for transactions (although MS SQL Server allows sharing of transaction space between connections via a special Stored Procedure `sp_bindsession` – see [10]). Delegates are processed (and data updated) in order of submission to the `TransactionWrapper.MakeAtomic()` method.

C# delegates also support asynchronous execution, but this scenario has not been tested with the scheme presented here. If the ordering of data updates is not important, then it is possible that asynchronous delegate execution could be made to work with this code, but mixing asynchronous calls with transactions could lead to problems: transactions typically lock (part of) a database during execution, and long-running transactional asynchronous method calls could therefore cause performance bottlenecks.

Any method that performs data update using the technique presented here must use the provided `IDbTransaction` interface. A deadlock condition is likely to ensue if an application mixes transactional database requests with non-transactional requests. Enforcing this requirement is probably largely down to good practice.

The scheme is not designed to support Distributed Transactions, although could in principle be modified for such a purpose, if a suitable implementation of the `IDbTransaction` interface were available.

It might be useful to consider providing event notifications at certain stages of the transaction, such as `TransactionStarted`, `TransactionCommitted`, `TransactionAborted`, etc. This would allow listeners to register for notification of these events, probably in the context of the submission to the `TransactionWrapper` of a group of transactional operations.

The code demonstrated here has been tested only with the Microsoft version of C#, running under the .NET Platform (versions 1.0 and 1.1) [9], but should work with little or no modification under alternative C# implementations, such as Mono [8].

Summary

This article introduced the C# Delegate type, and showed how it can be used to coordinate and simplify the management of transactional database updates. We have seen a coding scheme using Delegates which keeps tight control of the transaction logic, while also allowing a good deal of flexibility in the definition and location of database update code.

Matthew Skelton

References

1. *Design Patterns: Elements of Reusable Object-Oriented Software* - E. Gamma, R. Helm, R. Johnson, J. Vlissides: , Addison-Wesley, Reading, 1995. ISBN 0-201-63361-2
2. Facade Pattern:
   - http://en.wikipedia.org/wiki/Facade_pattern
   - http://www.dofactory.com/Patterns/PatternFacade.aspx
3. Command Pattern:
   - http://en.wikipedia.org/wiki/Command_pattern
   - http://www.dofactory.com/Patterns/PatternCommand.aspx
4. C# Language Specification: http://www.ecma-international.org/publications/standards/Ecma-334.htm
5. ACID definition: http://en.wikipedia.org/wiki/ACID
6. Aspect-Oriented Programming (AOP) - introduction: http://www.onjava.com/pub/a/onjava/2004/01/14/aop.html
7. *Introduction To C#*, Mike Bergin, CVu 16.3, June 2004
8. Mono: http://www.mono-project.com/
9. .NET Framework: http://msdn.microsoft.com/netframework/
10. Sharing transactions between connections on SQL Server using `sp_bindsession`: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_sp_ba-bz_9ini.asp
    Creating strongly-typed DataSet classes using the `xsd.exe` tool: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpcongeneratingstronglytypeddataset.asp
11. Automatic transactions in .NET: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconautomatictransactionsnetframeworkclasses.asp
12. Anonymous methods in C# 2.0: http://msdn.microsoft.com/vcsharp/2005/overview/language/anonymousmethods/

About the Author

Matthew Skelton is a Senior Analyst Programmer for Porism Limited (http://www.porism.com/), an application development company that specialises in bespoke database applications for the web. The company also publishes standards for e-Government, which are available at: http://www.esd.org.uk/standards.

Matthew is also a member of ACCU, and a recent convert to Python.

6. Although not central to this discussion, it is worth mentioning the FCL System.Data.DataSet class. An object of this class is an in-memory cache of data retrieved from a database, and greatly simpilfies data retrieval andupdate operations. Strongly-typed datasets can be used, whose class definitions are created from XML Schema [11], allowing compile-type type-checking of dataset operations, and access to tables and columns by name, instead of collections and indices.
7. In some respects, the plumbing required for transactions of this kind is akin to some of the problems addressed by Aspect-Oriented Programming [6], although such issues are beyond the scope of this article.

Full Source Code

```
using System;
using System.Collections;
using System.Data;
using System.Data.SqlClient;
namespace Library
{
  #region Delegates
  /// <summary>
  /// A delegate for data updates which
  /// require a transaction i.e. need to be atomic
  /// </summary>
  public delegate void
      TransactionedOperationDelegate(
      TransactionWrapper sender,
      TransactionArgs args);
  #endregion
  #region Operation Parameters Class
  /// <summary>
  /// Base class for all classes implementing
  functionality of parameters for transactioned
  operations
  /// Derive sub-classes from this, and customise
  with fields etc.
  /// In the transactioned operation, check the type
  of the object.
  /// </summary>
  public abstract class TransactionedOperation
  {
    public TransactionedOperationDelegate Execute;
    // cannot use 'event' designation as we are
    calling this multicast from outside the defining
    class
  }
  #endregion
#region Transaction Args
/// <summary>
/// Lightweight class representing arguments for a
transactionable operation
/// </summary>
public class TransactionArgs : EventArgs
{
  public TransactionArgs(
      IDbConnection connection,
      IDbTransaction transaction,
      TransactionedOperation operation)
    {
      this.Connection = connection;
      this.Transaction = transaction;
      this.Operation = operation;
    }
    public readonly IDbConnection Connection;
    public readonly IDbTransaction Transaction;
    public readonly TransactionedOperation
          Operation;
  }
  #endregion
  #region TransactionWrapper Class
  /// <summary>
  /// Provides a fairly generic way to wrap any
  operation within a transaction.
  /// The functionality is implemented using
  delegates.
  /// </summary>
  public class TransactionWrapper
  {
    #region .ctor public TransactionWrapper(
          IDbConnection connection)
      {
        if (null == connection)
        {
          throw new ArgumentNullException(
                "connection");
        }
    this.Connection = connection;
  }
  #endregion

  #region Fields
  internal readonly IDbConnection Connection;
  #endregion

  /// <summary>
  /// Makes an operation atomic by allowing
  all data updates it contains easily to share the
  same transaction
  /// </summary>
  /// <param name="operations">An operation
  which should use a single transaction</param>
  public void MakeAtomic(
              TransactionedOperation operation)
  {
    #region Parameter checks
    if (null == operation)
    {
      throw new ArgumentNullException(
            "operation");
    }
    #endregion

    MakeAtomic(new TransactionedOperation[]
          {operation});
  }

  /// <summary>
  /// Makes one or more potentially disparate
  operations atomic by allowing them easily to
  share the same transaction
  /// </summary>
  /// <param name="operations">An array of
  operations which should share the same
  transaction</param>
  public void MakeAtomic(
        TransactionedOperation[] operations)
  {
    #region Parameter checks
    if (null == operations)
    {
      throw new ArgumentNullException(
            "operations");
    }

    for (int i=0; i < operations.Length;++i)
    {
      TransactionedOperation operation
            = operations[i];

      if (null == operation)
      {
        throw new ArgumentNullException(
              "operation"
      }
      if (null == operation.Execute)
      {
        throw new ArgumentNullException(
           String.Format(
           "operation.Execute at index {0}",i));
      }
    }
    #endregion

    #region Main functionality
    this.Connection.Open();
    try
    {
      using (IDbTransaction transaction =
         this.Connection.BeginTransaction())
        {
        try
          {
           /// Prepare, then call the operation
           /// Any exceptions will cause the
           transaction to be aborted
           /// All relevent objects have been
           checked for nullness prior to this
           ///
          foreach (TransactionedOperation
                operation in operations)
            {
              TransactionArgs args = new
                   TransactionArgs(Connection,
                   transaction, operation);
              operation.Execute(this, args);
  // TODO: Log the operation here if required.
            }
            transaction.Commit();
          }
          catch (Exception exception)
          {
            transaction.Rollback();
  // TODO: Log the error here
            throw;
          }
        }
      }
      finally
      {
// Strictly speaking, there is no need for this close
here because IDbConnection
// implements IDisposable - the connection will be
closed when the object is Garbage-Collected
        this.Connection.Close();
      }
        #endregion
      }
    }
    #endregion
  }
  namespace Application
  {
    /// <summary>
    /// Example logging interface
    /// </summary>
    public interface ILog
    {
      void Write(DateTime dateTime,
                IDbConnection connection,
                Library.TransactionArgs args);
    }
    /// <summary>
    /// Some kind of custom data set
    /// </summary>
    public class CustomDataSet : DataSet
    {
      public string ExampleData;
    }
    /// <summary>
    ///
    /// </summary>
    public class CustomTransactionedOperation
          : Library.TransactionedOperation
    {
      public CustomTransactionedOperation
            (CustomDataSet dataSet)
      {
        this.Example = dataSet.ExampleData;
      }
      public readonly string Example;
    }
    /// <summary>
    /// A base class for data update
operations. Provides logging capabilities.
/// </summary>
public abstract class BaseDataUpdate
{
  public BaseDataUpdate (ILog log)
    {
      this.log = log;
    }

  private ILog log;

  private bool loggingEnabled;
  public bool LoggingEnabled
  {
    get { return loggingEnabled; }
    set { loggingEnabled = value; }
  }

  public virtual void UpdateData(
        Library.TransactionWrapper sender,
        Library.TransactionArgs args)
  {
    log.Write(DateTime.Now,
             sender.Connection, args);
  }
}

/// <summary>
/// An example class which performs data
updates.
/// </summary>

public class
      CustomDataUpdate : BaseDataUpdate
{
  public CustomDataUpdate
        (ILog log) : base (log) { }

  public override void UpdateData
       (Library.TransactionWrapper sender,
        Library.TransactionArgs args)
  {
    CustomTransactionedOperation
          operation = args.Operation as
          CustomTransactionedOperation;
    SqlCommand sqlCommand =
          new SqlCommand();
    sqlCommand.Transaction =
          args.Transaction as
          System.Data.SqlClient
          .SqlTransaction;
    sqlCommand.Connection
          = args.Connection as
          System.Data.SqlClient
          .SqlConnection;

    sqlCommand.CommandText
          = operation.Example;
          /* Some suitable text here */

    sqlCommand.ExecuteNonQuery(); // TODO:
           note the number of rows affected and
           publish via an event

    base.UpdateData(sender, args);
  }
}

/// <summary>
/// Example class demonstrating a stand
alone method used in data updates.
/// </summary>
public class StandaloneDataUpdate
{
```

```
/// <summary>
/// This method does not need custom
TransactionArgs parameter.
/// </summary>
/// <param name="sender"></param>
/// <param name="args"></param>
  public void SomeMethod(
        Library.TransactionWrapper sender,
        Library.TransactionArgs args)
  {
    SqlCommand sqlCommand = new SqlCommand();
    sqlCommand.Transaction = args.Transaction
      as System.Data.SqlClient.SqlTransaction;
    sqlCommand.Connection = args.Connection as
      System.Data.SqlClient.SqlConnection;

    sqlCommand.CommandText = "";
    /* Some suitable text here */

    sqlCommand.ExecuteNonQuery();
  }
}

/// <summary>
/// Some other kid of data set
/// </summary>
public class SpecialDataSet : DataSet
{
  public int MoreExampleData;
}

/// <summary>
/// Example class where the data update method
is actually part of the class.
/// This avoid the need to redefine separate
classes for the update method and the
parameters but might reduce flexibility.
/// </summary>
public class AggregateTransactionedOperation :
      Library.TransactionedOperation
{
  public AggregateTransactionedOperation
        (SpecialDataSet dataSet)
  {
    this.Example = dataSet.MoreExampleData;
  }

  public readonly int Example;

  public static void SelfContainedUpdateData
        (Library.TransactionWrapper sender,
         Library.TransactionArgs args)
  {
      // TODO: Paramter checks

    AggregateTransactionedOperation operation
          = args.Operation as
          AggregateTransactionedOperation;

    SqlCommand sqlCommand = new SqlCommand();
    sqlCommand.Transaction = args.Transaction
          as System.Data.SqlClient.SqlTransac-
tion;
    sqlCommand.Connection = args.Connection as
          System.Data.SqlClient.SqlConnection;

    sqlCommand.CommandText = "";
    /* Some suitable text here */

    sqlCommand.ExecuteNonQuery();
  }
}

/// <summary>
 /// Manager class which co-ordinates access to
data update operations
/// </summary>
public class DataUpdateManager
{
  private ILog log = null; // Initialise this
        somewhere...

/// <summary>
///
/// </summary>
/// <param name="dataSet"></param>
public void UpdateCustomDetails
      (CustomDataSet dataSet)
{
  IDbConnection connection
        = new System.Data.SqlClient.SqlConnection
        ( /* Connection string goes here*/ );
  Library.TransactionWrapper transactionWrapper =
        new Library.TransactionWrapper(connec-
tion);
  // Hook the data onto the operation parameters
  CustomDataUpdate customDataUpdate =
        new CustomDataUpdate(this.log);
  CustomTransactionedOperation operation =
        new CustomTransactionedOperation(dataSet);
  operation.Execute +=
        new Library.TransactionedOperationDelegate
        (customDataUpdate.UpdateData);

  transactionWrapper.MakeAtomic(operation);
}

/// <summary>
/// Performs multiple updates in a single
transaction, with disparate (but presumably
related) data.
/// </summary>
/// <param name="dataSet1">first data set</param>
/// <param name="dataSet2">second data set</param>
public void UpdateMultipleDetails
      (CustomDataSet dataSet1,
       SpecialDataSet dataSet2)
{
  // Prepare the connection
  IDbConnection connection =
        new System.Data.SqlClient.SqlConnection
        ( /* Connection string goes here*/ );
  Library.TransactionWrapper transactionWrapper =
        new Library.TransactionWrapper(connec-
tion);
  // Hook data for first operation
  CustomDataUpdate customDataUpdate =
        new CustomDataUpdate(this.log);
  CustomTransactionedOperation operation1 =
        new
CustomTransactionedOperation(dataSet1);
  operation1.Execute +=
        new Library.TransactionedOperationDelegate
        (customDataUpdate.UpdateData);

  // Hook data for second operation
  AggregateTransactionedOperation operation2 =
        new AggregateTransactionedOperation
        (dataSet2);
  operation2.Execute +=
        new Library.TransactionedOperationDelegate
        (AggregateTransactionedOperation
        .SelfContainedUpdateData);
  // Collect together the operations and make them
  atomic
  Library.TransactionedOperation[] operations =
        new Library.TransactionedOperation[]
        {operation1, operation2};
```

Reviews

Bookcase

Collated by Christopher Hill
<accubooks@progsol.co.uk>

The Editor Writes

The book review section of C Vu is one of the most important parts of our output, but we are in need of more reviewers. While it is good that we have a strong cohort willing to do the reviews, fresh blood will bring another perspective to the book reviews.

For a mere £5 per book (this is for postage – not the book itself), you have a list of some of the most up to date books around (some of which aren't even released in the UK at the time of being offered to the ACCU). All we ask is that you review the books. You even get to keep it!

One thing we do need is a new collator of the book reviews. Again, it is not a massive task (the reviews are sent to you and towards the editorial deadline date, send them across for final formatting), just one which we need doing. If you don't feel up to the task of writing reviews, this may be your opportunity to help.

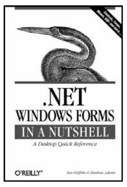
Interested? Email me and I'll fill you in with what is required.

Paul

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

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

.NET / C# / VB



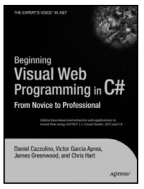
.NET Windows Forms in a Nutshell by Ian Griffiths & Matthew Adams (0-596-00338-2), O'Reilly £31-95 (Blackwells)
Reviewed by Paul F. Johnson

If you are already using Windows Forms in either VB or C# (both languages are covered), this is an invaluable book. In typical O?Reilly style, the book contains many well-explained code snippets on how to achieve certain tasks (such as overlays and positioning) in the now familiar no-nonsense way this type of book is expected to present.

This is not a teach yourself book by any stretch of the imagination; there are no directly compilable examples. If you want a teach yourself Windows Forms book, there are quite a number available - this book will compliment them perfectly.

Where the book fell down though is the CD. To use it, you must have Visual Studio.NET installed. Given that both C# and VB are available via the Mono project, it seems a pity that O'Reilly decided to make their CD unavailable to a growing market. Oh well, there is always the next edition.

Recommended



Beginning Visual Web Programming in C# by Daniel Cazzullno et al. (1-59059-361-8), Apress* £34-65 (Blackwells)
Reviewed by Paul F. Johnson

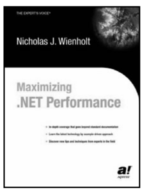
If you take this book as a novice book, it is not bad. It is well presented and well explained with the text being easy enough to follow with some very nicely presented examples.

The problem is that is all I can say about it. The code (available online) did not work on my Linux box, nor on an IIS server, and neither did the examples if I typed them in. I checked, rechecked, retyped, tried some examples I knew to work on both IIS and using the Mono framework to ensure the servers were happy with similar code and they were fine.

To me, this rendered the hard work in the book to be of very little use. Sure, given the time I could fix the problems (and indeed, I did on a few occasions), but for a novice, that is not what is required from the book.

Okay, take this for the professional who can fix code, does this book really offer very much? Given the caveat of what I have written above, not really. The examples are really for the novice and it does not really show any new techniques or better practice.

Not recommended



Maximizing .NET Performance by Nick Weinholt (1-59059-141-0), Apress £35-20 (Blackwells)
Reviewed by Richard Putman

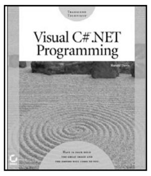
The experienced C/C++ coder usually has a reasonable idea of the cost of using certain program constructs and can visualise the generated assembler. However, with a .NET language the translation of instructions into Microsoft Intermediate Language (MSIL) calling into the Common Language Runtime (CLR) adds another layer between your code and the machine. Add in garbage collection and performance penalties are not obvious without investigation. This analysis is the subject of the book.

Assuming no previous experience of performance testing, it begins with basic definitions of white box/black box testing, profiling etc. itemising the factors to consider when benchmarking code. Most of this is common sense provided you have a healthy sceptical nature for performance figures. Described in the appendix, and available on the publisher's website, is the benchmarking testbed used throughout the book.

If you find yourself wondering how to balance execution speed against convenient code use, then this book will prove invaluable in helping you make an informed decision. If your completed programs are too slow, it is usually too late to refactor code, changing deep internal structures: hence, there is only one short chapter at the end of the book devoted to solving existing performance problems.

It is not only the choice of data structure that need investigating; the .NET framework provides alternative solutions for many problems e.g. when considering thread safety, there are several methods for synchronisation to choose from. Separate chapters are devoted to studying the comparative merits of string manipulations, collections, garbage collection, exceptions, IO and remoting amongst others. Throughout the book are references to numbered tests backing up the author's conclusions. Optimising is a notoriously controversial topic with more opinions than facts and I find it extremely refreshing to see frequent referenced tests.

Most of the example code is in C# or VB but C++ is included with sections to cover the effects of converting unmanaged to managed code as well as an interesting comparison of how all three languages take advantage of the CLR. The book is well written, perhaps occasionally a little verbose, but analytical without being dry, clearly typeset, includes plenty of valuable information and can be read cover-to-cover or used as a reference. In short, you should read this book before writing performance dependent .NET code. Recommended.



Visual C# .NET Programming by Harold Davis (0-7821-4046-7), Sybex £29-99 (Blackwells)
Reviewed by Ratul Bhadury

One of the fairly original things about VCNP hits you as soon as you begin with the first chapter. Rather than beginning with the clichéd console-based Hello World, and following the traditional sequence of chapters as most introductory textbooks, the first example is in fact the creation a simple web service! It was quite refreshing to see this approach as it immediately caught my interest. In fact, by the second chapter we had modified that for asynchronous use. Clearly, this book is focussed at the more experienced programmer, wishing to get acquainted with C#. The third and fourth chapters deal with windows programming, again, not what you would expect to be reading within the first third of an introductory textbook. The subsequent chapters follow a more conventional pattern, going over basic C# syntax and constructs, then onto the more advanced topics of Messaging, XML, and ADO.NET.

MVCN follows the more standard format (its first example is the console-based Hello World!) of starting at a beginners level with basic C# programming dealing with the data types, enumerations, operators, looping etc. and then gradually going onto the more advanced topics. While this is the more mundane approach, it is certainly more reassuring, more so given some of the holes in VCNP. Nowhere does VCNP discuss copy constructors, destructors, preprocessor directives that are fairly basic things, at least for someone coming from a C++ background. MVCN does a more comprehensive job of covering the fundamentals. But it also covers the advanced topics a lot more adequately.

There were some other worrying omissions from VCNP. There was no talk of multi-threading and synchronisation. For a book purported to be for experienced programmers I did find this very surprising. Also, as a book claiming to pay special attention to web services, I found its treatment of ADO.NET quite deficient. The authors of VCNP do direct the reader to the relevant topics in the Online Help, nevertheless I would have expected more than their extremely basic talk spanning 20-odd pages, as compared to MVCN?s significantly more exhaustive coverage over some 70 pages (in smaller print!). Also VCNP does not cover Remoting (again it does direct us to the Online Help), nor Security (but no mention of the Online Help either!).

From a readability perspective though, VCNP is far superior. It is written in a relaxed and casual, yet informative style. Reading it from cover to cover was no effort at all. On the other hand, MVCN, being the more conventional, was much more of a task. The two books have different objectives. MVCN comes across as the mundane, though reassuring pedagogue. The more exhaustive, it is undoubtedly the better reference book. VCNP, on the other hand, is the young enthusiastic teacher, obviously enamoured with his subject, and keen to spread his excitement to his students, but unfortunately overlooking important details. It is really up to the reader to decide what he/she prefers. I started with VCNP and could not help but to be drawn towards the brave new world of C#. It was only when I started with MVCN that I realized the rather gaping flaws of the former, with the inevitable feeling of being slightly let down. It is hard to say how I would have felt had I read them the other way round.

If it comes down to a choice, one or the other, I would have to, slightly reluctantly, recommend MVCN. But VCNP is nevertheless a very good book in its own right, much more readable, with very interesting example code. If money is no object, buy both!

C++ and Java

C++ Programming: Program Design Including Data Structures by D. S. Malik (0-619-16044-6), Thomson £25-20 (Amazon)

Reviewed by Mark Symonds

This is a very hefty tome weighing in at 1555 pages. There is some unnecessary filler material particularly in the early stages of the book, my particular favourite pointless paragraph being a brief history of computers which certainly lives up to its name by describing computer development for the 1950s to the present day in 9 lines of text!

The book gives a solid introduction to the basics of C++ programming and then delves into program design and data structures. All sample programs follow a similar format; a description of the problem followed by decomposition and code listing.

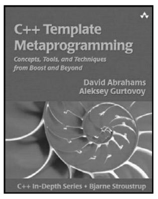
Good coverage is given of constructors and destructors but I disagree with the author’s advice of always having a default constructor; what if there is no reasonable default value for the class?

Templates are very briefly covered with only function and class templates described and template specialization not covered at all.

STL containers, iterators and algorithms have a chapter devoted to them although for serious usage another reference would also be needed. Examples do not use the STL widely because implementations of various container types are shown as design examples. Implementation and usage of various containers is given but these are not compatible with STL containers.

The chapter covering exceptions has examples catching exceptions by value and no information is given about the possibility of slicing when exception inheritance is used. No attention is paid to resource cleanup and auto_ptr is not covered.

UML diagrams of various classes and data structures are given in the program design discussion but no UML introduction is given and UML is not listed in the index.



C++ Template Metaprogramming by David Abrahams & Aleksey Gurtovoy (0-321-22725-5), Addison-Wesley* £31-99 (Blackwells)

Reviewed by Alan Griffiths

There have been other books that have presented aspects of template metaprogramming to the reader, but the ones I have read have been like travellers tales from a foreign land: interesting to hear, but not the stuff of everyday life. Or even to be emulated by any but the most adventurous and hardy tourist.

With C++ Template Metaprogramming Abrahams and Gurtovoy have changed all that! They have mapped out the territory; they have catalogued the fauna; they have provided an excellent glossary; and, they have organised a tour to remember.

This book is not an introduction to the use of C++ templates: if you are already not happy writing and reading code with lots of angle brackets I suggest you postpone reading this book for a while. Having familiarity with C++ templates will not only enable you to avoid discomfort, you will also have encountered some of the issues that motivate the Boost Metaprogramming Library [MPL] (which takes a key role in the book).

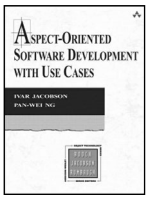
If you are familiar with templates but have never heard of ?template metaprogramming?, then don?t worry - you have almost certainly encountered it without a name before. At its most fundamental, it is getting the compiler to work something out at compile time. I am sure you will know how to get the compiler to produce the value type appropriate to an unknown iterator type. Obviously, template metaprograms vary from the trivial (like this example) to the complex (like the boost::spirit parser generator), and this book encompasses both extremes in its tour of the subject.

If, like me, you have experimented with using templates to perform computation at compile time you will appreciate the clarity brought by the conventions adopted by the MPL and the convenience of the many general-purpose metafunctions it provides. The MPL does for template metaprogramming what moving from assembler to C does for traditional programming - I remember the tedium of having to code my own stack frame handling and register passing.

The book demonstrates the wide range of applicability of template metaprogramming with powerful examples: a type system for SI units that performs dimensional analysis at compile time, implementing linear algebra efficiently and supporting an embedded language for expressing grammars are some of the highlights. To achieve this it requires the reader to keep up with the pace - while I read most of it while on public transport this is not ideal: I would recommend having your compiler nearby to explore the ideas it illuminates along the way.

Some of the techniques employed do push compilers hard, and the concerns raised by this are not ignored. There are practical discussions of both getting meaningful error messages out of the compiler and a study of the impact that the choice of coding idioms has on compile time with a range of current compilers.

Highly recommended.



Aspect-Oriented Software Development with Use Cases by Ivar Jacobson & Pan-Wei Ng (0-321-26888-1), Addison-Wesley* £39-99 (Blackwells)

Reviewed by Frank Antonsen

I was first very excited about this book, then somewhat disappointed, and then excited again.

Originally, I had expected more about Aspect Oriented Programming (AOP), a topic I had my first real introduction to last year at the ACCU spring conference in Oxford. The topic fascinated me, and I had done a little bit of reading on my own, before I saw this book on the book list. You turn to AOP when you need an elegant way of taking so-called cross-cutting concerns into account. These are concerns that cut across several classes, modifying the original behaviour. Typical examples are logging and persistence. The extra functionality this adds to the existing classes, and in particular their function members, has nothing to do with the original functionality.

I had expected more about AOP, but the book only gives very few concrete examples. This was the reason for my initial disappointment. However, I read on and soon found my enthusiasm for the book revived.

There is a heavy emphasis on proper software development processes. Many books have been written on that particular topic, but not as readable as here. Most of the discussion applies to non-AOP as well as to AOP. But this is actually just the beauty of the book. It shows how AOP can be seamlessly integrated in your daily practices as a software developer.

In fact, the book reads as if use-case driven design and AOP were made for each other. Unified Modelling Language, even in version 2, does not take AOP into account. But UML is flexible, and the authors show that very few extensions are needed to fully accommodate AOP.

Before AOP, or rather in non-AOP projects, use-cases didn't really have the same level of modularity as the rest of UML. The reason for this is precisely that a typical use-case cuts across different classes, in that a typical use-case involves different aspects of different classes. This can be described in a modular fashion using, well, aspects.

On the UML side, aspects correspond closely to what the authors coin use-case slices. This is defined as: A use-case slice contains the specifics of a model in a single package. A use-case slice of the analysis or design model contains classes and aspects of classes specific to a use-case. You describe aspects, and their corresponding pointcuts (where the modification is to be inserted in existing code), by adding new compartments to standard UML diagrams and using stereotypes. This is just the normal UML way of handling extensions to the basic diagrams, but it shows how elegantly AOP fits into standard object-oriented software development.

This being a book by Ivar Jacobson, the emphasis naturally is on how to identify aspects and use-case slices in every step of the process, from analysis and design to implementation. As I said earlier, even if you will not be using AOP tools in your next projects, it is still worth reading this book, just for the clarity in showing you how to use use-case driven methods in your project.

Beginning PHP, Apache, MySQL, Web Development by Michael Glass et al. (0-7645-5744-0), WROX £29-68 (Blackwells)
Reviewed by Richard Lee

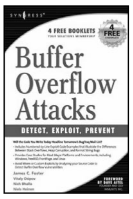
To get anything out of this book ignore the blurb on the back. The book does cater for those who have never met PHP before but previous HTML knowledge is essential. To this I would add experience of any programming language as the authors barely go beyond the syntax of PHP. A better attempt at explaining SQL to a beginner is made but pales in comparison to a dedicated book.

The majority of the book is split into two uneven sections, each building up typical web site. Chapters follow a cookbook template; quick introduction followed by the source code and an explanation of how it works. Further source code and explanations then elaborate or fix shortcomings in the original code. At the end of the chapter there is a short summary and the occasional exercise.

The six authors are linked through a scripting site, which shows in the practical examples used. If you are looking to achieve something on your web site there is a good chance that there is a chapter in the book dealing with a similar issue. Having chapters largely self standing allows the reader to develop solutions in a short time period.

This brings its own problems. The number of authors means the programming style chops and changes between chapters and useful information is not carried through to subsequent chapters. Having each chapter stand on its own does mean that near the end they get very long. Too much code (20 files in the CMS example) has to be entered in one go to realistically expect the reader to understand how everything fits together. My final gripe is that far too many mistakes have got passed proofreading.

This book is very practical and offers a shortcut to creating a web site. However it would benefit from a second revision.



Buffer Overflow Attacks by James C Foster et al. (1-932266-67-4), Syngress £19-99 (Blackwells)
Reviewed by Mark Symonds

This book covers the stack overflow, heap corruption and format string bugs that are some of the most common source of security vulnerabilities. The book has three main sections covering detection of vulnerabilities, exploitation and prevention.

Throughout the book case studies are given of various vulnerabilities on BSD, Linux and Windows platforms complete with exploit code. These are widely known and there are links to various sites with more information. The example code in the book is available for download.

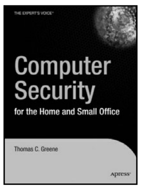
Chapters on stack overflow, heap corruption and format string bugs are detailed and informative and the code examples should pose no problems for ACCU members.

Although the book contains much information into how these types of vulnerabilities occur there are many problems with both the style and content of the book.

There are three co-authors contributing to the book and the chapters written by the authors do not sit well together. As an example the chapter introducing Win32 assembly, which mainly consists of an introduction to 80x86 assembly language programming, comes immediately after a chapter on writing shellcode, which uses 80x86. In addition information is repeated throughout the book.

The weakest part of the book is the section on finding buffer overflows in source, which consists of an overview of several code checkers with sample output.

This could be a good book with some heavy editing to glue the disparate parts together and to correct some of the mistakes in the text but cannot be recommended in its current form.



Computer Security for the Home and Small Office by Thomas C. Greene (1-59059-316-2), Apress £27-50 (Blackwells)
Reviewed by Alan Lenton

This book is a real curate's egg - a curious mixture of clear practical advice for Windows and Linux users, and hard line Open Source advocacy.

The first couple of chapters are absolutely excellent stuff on computer security explained in a non-geek fashion. Just the section on how to harden your Windows computer by turning off unnecessary services is probably worth the price of the book. Greene goes through the services one by one, explaining in what circumstances you might need to leave them on, and gives detailed instructions for turning them off. The instructions given are for XP, but I used them on my Win2K setup with no problem.

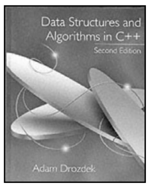
Other chapters give useful insights into the sort of social engineering used by hackers and an introduction to basic network monitoring tools.

After this things start to break down a bit. It wasn?t that anything was wrong, or even badly written, but the next chapter is a very paranoid piece on not leaving unsecured traces of your data around (?data hygiene? as Green terms it). The following chapter (chapter 6) is a hard line piece of Open Source advocacy. This links back to the introduction where Greene has a section on installing the Mozilla browser.

The final part of the book is called ?Trust Nothing, Fear Nothing? and is an expose of myths about the Internet. It is excellent journalism, but it feels odd stuck in as a chapter at the end of this book.

Some of the book?s shortcomings are possibly explained by the fact that Greene is an associate editor of The Register (one of the best, if not the best, online publications around, in my opinion). Greene is an excellent journalist - I regularly read his pieces - and in this book he is mixing hardheaded advice with investigative journalism. Unfortunately, he does not quite bring it off, and as a result the book feels uncoordinated and unsure of what its mission is.

Recommended, but with reservations.



Data Structures and Algorithms in C++ by Adam Drozdek (0-534-49182-0), Thomson £36-99 (Blackwells)
Reviewed by James Roberts

[This review covers both the C++ and Java versions of this book.]

These are textbooks for a course in Computer Science. They are almost identical, covering exactly the same basic topics, using different languages for illustrative purposes.

In general these books works well as academic sources. The algorithms (there is more on algorithms than data structures) are presented in related chapters (trees, lists, data compression etc). The author tends to explain the algorithm in text, pseudo-code and in example. In many cases a complete source listing is also provided. Thus I was able to gain an understanding by comparing the textual description of the algorithm with the pseudo-code and the illustrative example. This was occasionally very important, as I sometimes got lost in the dense textual descriptions.

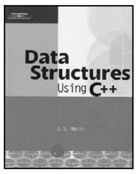
The efficiency of the algorithms (best case, worst case, order-of) is generally discussed and explained in terms of big-O notation. Accessible proofs of the big-O values are provided.

In both books there were references to the STL/Java implementations of the general principle - although I was not sure whether the book was over-dogmatic regarding implementations (does a map *have* to be implemented in a red-black tree, or is it just a common choice?).

There is a great deal of exercise material included as end-of-chapter exercises, which points to the books? roots as course material.

These are not books to be read to gain an understanding of how to mechanically use the STL (or Java equivalent methods), but may inform their usage in a more subtle manner.

Although I don?t think that I will use these as reference material, I enjoyed reading them as a change£for more target-focussed material. If you have a need to understand how various algorithms work, and how one might implement them, these are a pretty good summary.



Data Structures Using C++ by D. S. Malik (0-619-15907-3), Thomson £41-80 (Blackwells)
Reviewed by James Roberts

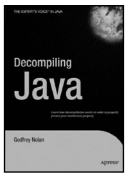
[This review covers both the C++ and Java versions of this book.] These books are clearly part of a computer science course. They both provide an introduction to the language in hand, and then describe basic data structures - together with an explanation of the pre-packaged (STL or Java class) equivalents.

I am not sure about the effectiveness of trying to incorporate C++ (or Java) explanation into the book. I think that either it will be skipped, or will just confuse a reader by trying to pack far too much information into much too little space. Perhaps a shorter overview of apposite topics (e.g. shallow versus deep copy) would have been more sensible in its place. I found that in the C++ book, there were some nasty errors (non-virtual deletion, in a class which was later subclassed), which were an annoyance, especially in a teaching manual.

On the plus side, the data structures/algorithms are reasonably clearly presented and illustrated with implementation examples. The efficiency of the algorithms is covered with some indications of big-O notation (although the proofs are often skipped). The author also covers the STL equivalents of his own implementations.

My main grouse is that there seems to be a lot of code, and a lot of example, but not very much explanation of why things are the way they are.

Although these are not bad books, there are better books available (In C++ between the Drozdek and the classic Josuttis, the theory and the practical sides are much better served).



Decompiling Java by Godfrey Nolan (1-59059-265-4), Apress* £27-50 (Blackwells)
Reviewed by Christer Lofving

This volume has many appetizing features. First it is bounded, a rare feature for computer books nowadays. Also it's size, colours and the balanced number of pages (260) all builds to make it a delightful book to look at and keep in your hands. More important, it has high-quality content as well.

I cannot remember when I last got this devouring feeling by reading a technical book. For myself I finished it on my spare time in 2-3 days. One explanation for the high quality is probably that the work on the book has been evolving under many years. Even since the advent of the first Java applets in mid 90?s. The core of the Java Virtual Machine (JVM) has not changed much since these days.

That means the accuracy and topicality of the content is in no risk of becoming outdated. The first part gives a fascinating journey inside a Java technology cornerstone; the class file. After that the book continues with an overview of existing Java decompilers, different protection methods (i.e. obfuscators, fingerprints) and some legal issues about decompiling other?s code (property).

The second half is about building your own basic Java decompiler, mainly for pedagogical purposes, as I understand it. But for this task the reader is indeed given a detailed step-by-step tutorial. No critical code or necessary explanations are left out. Unfortunately the contrary use to be a more common case for computer books; the flaw to present only fragments of code and abandon the reader when it comes to doing the real work.

But it is also important to highlight who this book is NOT aimed for. It is not for the cracker/hacker. OK it is about decompilation, but no undocumented ?secrets? are revealed here. Nor is it either for the reader mainly interested in the Java Virtual Machine itself. The JVM is mentioned but merely analysed. Instead the content is built around the architecture of a Java class file, decompilation techniques and code protection strategies.

Finally, it is not for the beginner, despite the author is suggesting this approach of learning Java in his introduction. To learn Java this intriguing way is only for a true Assembler guy!

Java Event Handling by Grant Palmer (0-13-041802-1), Prentice Hall £31-99 (Blackwells)
Reviewed by Paul Thomas

With Java, the language itself is relatively simple but the sheer volume of library classes is overwhelming. Once you have picked up the basics, there is still a long way to go before you can feel at home writing large applications. I had hoped that Java Event Handling would help me learn an important aspect of Java in depth. What I got instead was a reference book comparable to the JDK documentation but without the accuracy or completeness.

This book has a large, dry reference section with the inevitable repetitions and a small tutorial introduction. The really useful introductory explanations are embedded in the reference section and the first five chapters scan more like an afterthought.

If you intend to buy this book to gain a deep understanding of event handling in Java, then forget it. The writing is so dry as to be nearly unreadable. That in itself would be excusable if the information was accurate or in anyway complete. But it is not.

Throughout it all, the author uses a procedural style of explanation that leaves the reader with little context and the non-standard jargon adds to the confusion. I read the first three chapters without any real clue as to what was going on. A picture or two might have helped.

If you want a reference for all things event in Java, then this book might offer you something. Most of the examples provided with the class descriptions are of a high standard and you can pick it up second hand for next to nothing (unsurprisingly). Just remember to rip out chapters one through five.

Robust Java by Stephen Stelting (0-13-100852-8), Prentice Hall* £39-99 (Blackwels)
Reviewed by Alistair McDonald

This is a very well produced book. It is published in conjunction with Sun Microsystems, and the text is very easy to read. The style engages the reader without too many jokes or witticisms.

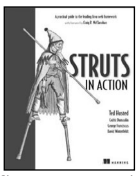
The first of three parts covers the fundamentals. This includes what exceptions are, how to specify and handle them, and so on. This is basic stuff, and I expect any serious Java developer to already know this material, so it can be viewed as somewhat superfluous. However, it is thorough and well presented.

Fortunately, the material becomes more specialised. Stack traces, thread-safety, and creating exception classes are all covered as part of the first section. This section also looks at the Logging API introduced with the JDK 1.4.

Part 2 is a bit of a mixture. It begins with a chapter on designing exception handling into code. This material is far more useful for a typical Java developer. Other chapters include a comprehensive discussion of the exceptions thrown by various standard Java classes, including the collection classes. J2EE and distributed APIS are also covered.

Part 3 is also a bit of a mixture. The material is more advanced than Part 2, but again the chapters appear unrelated at times. The subjects covered include a chapter aimed at software architects, a chapter on exceptions in design patterns, and chapters on testing and on debugging.

Although I recommend this book, it does not tie together as a whole cohesive unit, rather it is a collection of loosely related chapters. However, it does its job well and should be a useful addition to a journeyman Java programmer’s bookshelf.



Struts in Action by Ted Husted et al. (1-930110-50-2), Manning £40-50 (Blackwells)
Reviewed by Christer Lofving

This is my second book about Struts, but I wish it had been the first one. Having now worked with struts in two different J2EE-project I must say the concept is not very intuitive.

It is an excellent idea by itself. Using form classes as a multifunctional tool (value object, fire wall, data validator..) and ?action classes? to separate the user interface from business logic. But still the order of actions to take when implementing struts in an ordinary web application can be confusing.

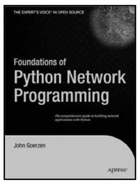
You cannot learn Struts architecture from a book. The common truth that you need to get your hands dirty with code to really grasp a new programming concept is indeed relevant with struts. But still, this title is a good companion on the way. Due to the pretexts it has a told-out ambition to become a project groups ?bible? for struts. I think it fulfils this goal pretty well.

For the already fluent in struts there are chapters that cover validation, error handling and data services in depth. Also, one of the most comprehensive parts of struts and I believe the future of we -programming, tiles, is covered in its own in-depth tutorial.

The beginner will find a detailed step-by-step approach in the first part. The recommended route is working out these examples with a local tomcat-server and text editor (and a download struts, of course). Also the project’s system architect will find a lot of good stuff here. At the end there is a short but information appendix about the use of struts in conjugation with different design patterns.

A drawback is the shortage of a more detailed reference for the different strut tags, preferably with small examples. It is also hard to find what you are looking for in the book when it comes to real-life problem solving. There are so many parts on different levels in the book, making it hard to get an overview. And the index is definitely too small for a +600 pages book. Add to this the fact that struts terminology itself is confusing before you get used to it and you get the picture. But definitely worth its price.

Python



Foundations of Python Network Programming by]ohn Goerzen (1-59059-371-5), Apress £40-50 (Blackwells)
Reviewed by Alan Lenton

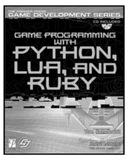
As someone who is mainly a C++ programmer, with a relatively small amount of Python usage, this book triggered my envy of the high level facilities that Python has built into its standard libraries. In this case it was, of course, Python's networking libraries. But I am left wondering when C++ is going to stop merely providing libraries that mainly address the last decade?s problems, and start to tackle the problems routinely faced by today's application programmers.

The book is very useful, not just because of its excellent explanation of how to use the Python networking libraries, but also because it contains very clear and understandable explanations of network issues and jargon. Indeed, this has become the book I would recommend to a programmer wanting to familiarise him or herself with network programming.

Topics covered include low-level socket programming, DNS, basic web, e-mail, protocols, server side programming (including CGI), database clients, SSL, and multi-tasking servers. I would have preferred a little more discussion of asynchronous servers, but I realise that threading is all the rage at the moment! The only noticeable absence is that there is no section on peer-to-peer programming, but I guess you can't cover everything in one book.

There are a few idiosyncrasies. For instance, introducing the select() and poll() system calls as a method of multiplexing user client input is a somewhat novel way of going about things - the calls would have been much better introduced in the section on asynch servers, which is their more normal use.

This does not, however, detract from the book?s usefulness. I enjoyed reading this book and it filled in several gaps in my knowledge. Recommended.



Game Programming with Python, LUA and Ruby by Tom Gutschmidt (1-59200-077-0), Thomson £34-00 (Blackwells)
Reviewed by Derek Graham

I was looking forward to getting this book for review but after starting to read it I quickly became disappointed. It begins with a look at high-level languages which will not be of any interest to anyone who already knows anything about programming. After this is a chapter comparing the three languages and demonstrating the obligatory Hello World example. I have a degree of familiarity with Python and Lua but after reading this chapter, with code examples for each language juxtaposed so tightly, I was beginning to become confused about which piece of syntax belonged to which language. At the end of the book, I actually feel that I know less about three languages and feel more confused about the whole subject.

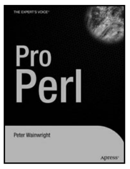
After the introductory chapters, the rest of the book is divided into three sections, starting off with the runtime environment for each language, a more in depth look at some programming constructs, the available games libraries for the language and finishing off with a simple game example.

The book purports to teach game programming to anyone no matter what their current level of programming skill. The slightly odd thing about this book is that it feels like it started out as a much longer book and has been edited down to a specific number of pages. It contains some beginner information, some material for the advanced beginner but has had the intervening explanations ruthlessly edited out. It is therefore difficult to see that it would be useful to absolute beginners or experienced programmers learning game programming; there is just not enough information for either one.

This is particularly noticeable in the introductory chapter which explains modern high-level languages in absolute layman?s terms with some gross over simplifications you could forgive a true beginners book but then talks about addresses, registers and binary as if the reader is an accomplished at assembly language. All this within a page and sometimes within a single paragraph. There are also a number diagrams showing the lineage of each language with no explanation of what the other languages were or what they contributed to the language in question.

In the end I did not feel anyone would learn anything more from this book than they could have with Google and a free afternoon. For the length of book it is, it should have stuck to one language and explored the subject properly. Not recommended.

Perl



Pro Perl by Peter Wainwright (1-59059-438-X), Apress £37-50 (Blackwells)
Reviewed by Derek Jones

This book aims to be a comprehensive guide to Perl and with 1,037 closely typeset pages it is a serious contender for that role. The introduction points out that this is actually the second edition of Pro Perl; the first edition being published as Professional Perl Programming.

The obvious alternative to buying this book is the ?Camel' book, i.e., Programming Perl by Wall et al. (which is 52 pages larger, around 10% cheaper {depending on discounts}, in its third edition, and has a Camel on its front cover). If you are looking for a single Perl reference book then Pro Perl is good enough to be given serious consideration.

As somebody who is not a dedicated fan of Perl I found Pro Perl to be easier to follow than the Camel book (those with a Perl mentality will probably claim that this books in jokes add to, rather than detract from, its readability). Heavy Perl users will probably learn something from reading Pro Perl, even though they have a copy of the Camel book.

If you are looking to buy a Perl reference book then Pro Perl and Programming Perl are the two main contenders. My advice is to look at both books and choose the one that appeals to you. It is a pity that Apress do not make available a sample chapter for download (O'Reilly do for Programming Perl).

Due to lack of space, not all book reviews could be printed in this issue.
Reviews of the following books can be found on the website (www.accu.org) and will be printed in the next issue if space permits.

Linux

ia-64 linux kernel : design and implementation by David Mosberger & Stephane Eranian (0-13-061014-3), Prentice Hall £47-99 (Blackwells)
Reviewed by Ian Bruntlett

Teach Yourself Red Hat Linux Fedora in 24 Hours by Aron Hsiao (0-672-32630-2), SAMS* £21-99 (Blackwells)

Regular Expression Recipes: A Problem-Solution Approach by Mathan A. Good (1-59059-441-X), Apress* £22-00 (Blackwells)
Reviewed by James Roberts

Systems & Design

Modernizing Legacy Systems by Robert C. Seacord et al (0-321-11884-7), Addison-Wesley £34-99 (Blackwells)
Reviewed by Mark Symonds

Simple Program Design: A Step-by-Step Approach by Lesley Anne Robertson (0-619-16046-2), Thomson £18-89 (Amazon)
Reviewed by Mike Pentney

Use Cases: Patterns and Blueprints by Gunnar Overgaard & Karin Palmkvist (0-13-145134-0), Addison-Wesley* £39-99 (Blackwells)
Reviewed by Giles Moran

Web Services & Web Development

Tapestry in Action by Howard Lewis Ship (1-932394-11-7), Manning £23-19 (Amazon)
Reviewed by Andrew Marlow

Understanding SOA with Web Services by Eric Newcomer & Greg Lomow (0-321-18086-0), Addison-Wesley* £28-99 (Blackwells)
Reviewed by Rick Stones

Misc.

Joel on Software by Joel Spolsky (1-59059-389-8), Apress* 10-85 (Amazon)
Reviewed by Pete Goodliffe