The magazine of the ACCU

www.accu.org

{cvu}

Volume 19 Issue 5 October 2007 £3

Features

Regulars

{cvu}
Volume 19 Issue 5
October 2007
ISSN 1354-3164
www.accu.org

How about letters?

It seems to me that the months are just flying by. Some part of me thinks that it is due to general work pressures or children growing, but I think perhaps it is just me getting older. They (the old fogies) say that time speeds up as you get older.

When I was first considering writing this editorial I was hoping to say that 'By the time you read this, the first Otago ACCU meeting should have happened', but the organisation of it is taking longer than I had hoped. It seems that it will now be 'Before the next editorial, the first Otago ACCU meeting should have happened.' I was reading our local newspaper and came across an article about Dunedin attempting to place itself as a ICT centre of excellence, and it mentioned 30 companies in Dunedin that had professional programmers. Now I live in Dunedin and I didn't realise that there were that many hackers around. It spurred me on to set up the inaugral Otago ACCU meeting. I was put in touch with the head of the Otago chapter of the NZ computer society, and it looks like we'll be putting them on as a joint effort. I'm going to be giving the initial talk as its hard enough trying to get these things rolling without having to find a speaker too.

I happened to read a posting on accu-general recently where it was mentioned that there was not a lot of feedback or comments on articles that were submitted to CVu or Overload, and the author of the email found that it did not really encourage them to write more. I can sympathise with this, but also you need to work out why you are writing. Unfortunately, writing for ACCU to get immediate (for some value of immediate) recognition is never going to end well. Most, if not all, the authors write with very little recognition. I'd love to be able to have a letters section in CVu with reader's responses to articles, and this may even encourage more people to write. I remember being really chuffed with my first email response to an article I wrote (and I don't think it was for the first article I had published).

The deadline is approaching to have submissions in for the ACCU 2008 conference. May I suggest that the speakers write up their talks first as articles? Well, I can try.

TIM PENHEY,
EDITOR

The official magazine of ACCU

ACCU is an organisation of programmers who care about professionalism in programming. That is, we care about writing good code, and about writing it in a good way. We are dedicated to raising the standard of programming.

ACCU exists for programmers at all levels of experience, from students and trainees to experienced developers. As well as publishing magazines, we run a respected annual developers' conference, and provide targeted mentored developer projects.

The articles in this magazine have all been written by ACCU members – by programmers, for programmers – and have been contributed free of charge.

To find out more about ACCU's activities, or to join the organisation and subscribe to this magazine, go to www.accu.org.

Membership costs are very low as this is a non-profit organisation.

DIALOGUE

REGULARS

FEATURES

COPY DATES

C Vu 19.5: 1st November 2007
C Vu 19.6: 1st January 2008

IN OVERLOAD

Kevlin Henney continues the PfA Papers, Allan Kelly introduces an Agile process, Stuart Golodetz looks at C++ templates and Richard Harris concludes his auto_transfer series.

ADVERTISE WITH US

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

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

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

COPYRIGHTS AND TRADE MARKS

RegexBuddy

Christer Löfving introduces an inexpensive tool for learning and using Regular Expressions.

A year ago I worked with Tcl[1] , one of the shell scripting languages that actually has support for Regular Expressions built into its semantics[2]. That awakened my interest for the topic in a more serious way, and I decided to dig a little deeper.

A Google search led to an excellent piece of software that I think deserves an article of its own, namely RegexBuddy by JGSoft [3]. Alongside Cygwin, Edit Plus and Mozilla Firefox, it quickly became a priority from amongst my existing software to install on a new computer. It's also a clear candidate for the USB stick, where it is easily installed and configured as well. True, we really aren't pampered with Windows applications from heaven but RegexBuddy is designed in a way more Windows applications should be. It is stand-alone, lightweight yet powerful and has an overall user-friendly appearance. Its footprint is small, actually very small, for a Windows application – only 20 kB memory usage and a 10 MB space on the hard drive is occupied after a full installation.

I have personally come to be a bit addicted to Regular Expressions, but to be honest it can feel tricky to get started with them. This is mainly because of the somewhat obscure syntax.

To take an example, a RE (Regular Expression) to match the pattern of a valid Visa credit card looks something like this:

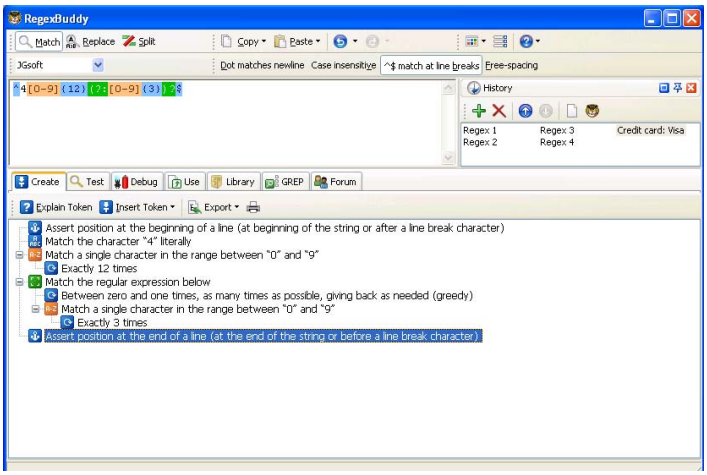
```
^4[0-9]{12}(?:[0-9]{3})?$
```

And that's the expression in a 'basic' form. To test it in reality using Perl as tool of choice, one has to include at least the following lines in a script[4]:

```
if ($subject =~ m/^4[0-9]{12}(?:[0-9]{3})?$/) {
  # Successful match
} else .....
```

Of course this makes it cumbersome for a beginner to even *test* a RE, never mind experimenting with REs and designing his or her own. Practice is, as we know, the best way to learn anything and that is certainly particularly true when it comes to hand crafting a RE. In this situation, RegexBuddy provides invaluable aid. Just cut and paste the above expression into the RegexBuddy 'Create' view, and the overall picture is visually laid out (Figure 1).

Figure 1



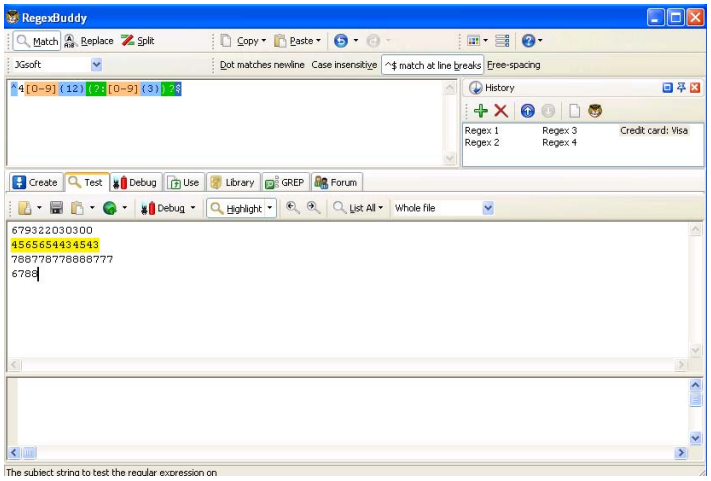


Figure 2

All of the obscure syntax is translated to plain English. It is also possible to build an RE from scratch, or modify an existing one, using the 'Insert token' menu option in this view.

Testing the same RE against a bunch of text becomes piece of cake (a match is highlighted in yellow – Figure 2).

There is even a debugging facility that visualizes how the RE matching machine works. With our simple sample it looks like Figure 3.

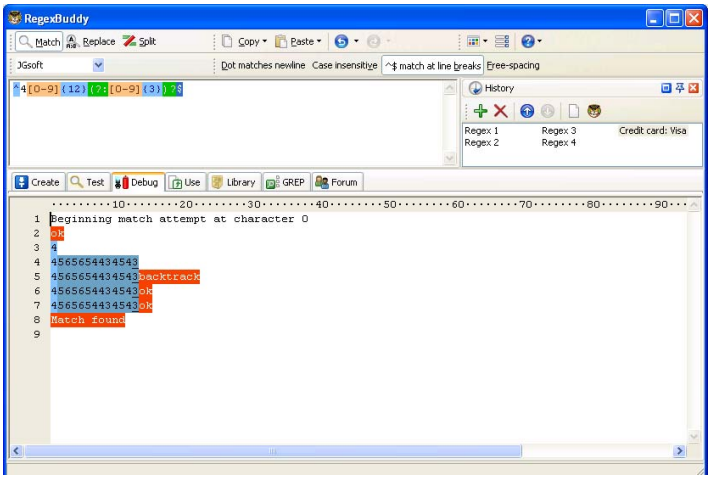


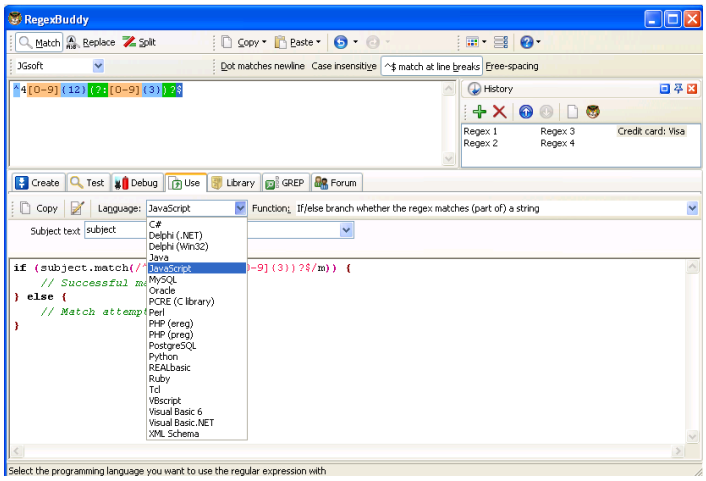
Figure 3

When it comes to more complicated REs, this can be an efficient tool to track down otherwise hard-to-detect bugs. In the 'Use' view, a choice can be made among a few RE flavours for immediate code generation, then just cut and paste the new or modified RE into its coding context. The rather impressive list of options includes Ruby, Oracle and XSD among others. (Option PCRE , a C library, seems to have a bug, though. It only generates the `#include <pcre.h>` header). (Figure 4.)

CHRISTER LÖFVING

Christer is a full time consultant in the Nordic branch of LogicaCMG, and has contributed many book reviews since he joined ACCU in 1995. Christer can be contacted at chlof@wmdata.com.

Figure 4



From the above illustrations, it is easy to understand the potential of this tool for the beginner struggling with his or her first RE and for the fluent RE user as well. The final view includes a GREP utility – I haven't tried that one out, though. RegexBuddy comes with an extensive Regex library ready to use, and it is possible to create new libraries.

In the earlier versions, one had to work with JGSofts 'generic' RE flavour during the design and testing phases, finally converting to a particular flavour. But in the latest version 3.03 (July 2007), RegexBuddy easily adapts to a language of choice from the first step of creation. (Note, this choice is only for the most commonly used flavours like Perl, Python, Java and .NET.)

RegexBuddy makes a nice use of the nowadays more or less forgotten Windows help files tool Winhelp (deprecated with the launch of Vista). A RE reference and a tutorial really worth its name are provided in Winhelp format. In fact, I studied this tutorial for myself several weeks before proceeding to a book.

A drawback is that RegexBuddy is neither freeware nor shareware. There is NO free trial version to download, not even for a limited period of time. However, the price for a single user licence is surprisingly low (29.95 Euro in August 2007).

Another potential minus is that the Linux version has been removed in the latest release. However, according to RegexBuddy documentation, it is possible to run the Windows version on Linux using Wine [5] – a tool for running Windows applications in a Linux desktop environment. Ubuntu 7.04 seems to be the only tested distribution, but it should work with most other distributions. (Wine has ready to use installation packages for all the major Linux distributions.)

Of course RegexBuddy can be run as a standalone tool, but it is also designed to be used as a companion to whatever software's REs it is used in conjunction with.

A lot of flexibility is offered. Earlier I mentioned RegexBuddy can be installed on and runs fine from a USB-stick. When it comes to IDEs, RegexBuddy can be added to the actual software's Tools menu then command line parameters are used to transfer the RE in question to RegexBuddy using the clipboard. There is even support for a COM automation interface (if anyone happens to be adventurous enough to try that out ☺).

I got another small surprise when researching for this article and found the physical location of the RegexBuddy creator, JGSoft. It's a street address in Thailand! Not that there's anything wrong with Thailand, not at all. But if I had searched for RE tools by country, I think my choice would have been US, UK or any other European nation. Probably I would never have thought to see what a small software company in Thailand might have to offer. That's the beauty with the new wonderful Internet world and its unprejudiced search engines. It allows quality to shine through our delusions. ■

I would never have thought to see what a small software company in Thailand might have to offer

References

[1] Tcl , Tool Command Language

[2] Perl obviously belongs to this group too.

[3] http://www.regexbuddy.com

[4] I know, there are 'one liners' in Perl, but in this context I am referring to a RE beginner.

[5] http://www.winehq.org/

Recommended reading

Mastering Regular expressions, Jeffrey Friedl, O'Reilly Media, Inc.; 3rd edition (August 8, 2006)

An NSLU2 "Slug"

Silas Brown suggests some tinkering.

After reading an article in the Linux Gazette called 'Debian on a Slug' (you can Google for it), I bought a Linksis NSLU-2 (a small embedded device used to connect USB disks to Ethernet and act as a storage server) and reprogrammed it to run Debian Linux. I now wish to recommend this to other ACCU members for the following reasons.

Most developers need servers, be they web servers, version control servers, or whatever. Many will put the server on a PC that's left switched on 24 hours a day, or at least whenever they're developing. But PCs are often big and noisy and anyway they guzzle far too much electricity: at UK Economy 7 prices, a 24-hour NSLU2 server should pay for itself in less than 2 years when compared with a 24-hour PC, even accounting for the fact that the PC will still be switched on some of the time anyway. Perhaps more importantly from a developer's point of view, if you have an NSLU2 then I find you tend to be more willing to start servers on it, as long as they're lightweight enough. An NSLU2 using Flash media makes a very good home server (and router etc) because it's totally silent and you don't have to worry about the power it's taking. That means there's nothing to stop you running a server when you need to, which tends to improve productivity if noise/power/size etc was an obstacle before. Also, running it on an NSLU2 makes you think about making it lightweight, which could help with scalability later if you're doing any server work. And setting up and customising an NSLU2 is a good exercise in Linux tools. ■

The Version Control Job

Steve Love continues his series on version control.

In the last article [1], I gave a short introduction to the Bazaar Version Control System, and why it meets my needs as a developer. In short, the decentralised nature of Bazaar means that it is easy for me to work on my laptop away from my main PC – which 'serves' my source code repositories – and perform local commits which I can later merge back to the main repository, losing no history or version information in the process.

In this article, I want to take a more detailed look at how I use Bazaar; as a result, this is partly a user-guide, partly a how-to and partly a look at the approach that Bazaar takes to certain version control tasks. Along the way we'll look at how projects get versioned, using a central repository, branching and merging, and release management of projects versioned under Bazaar.

First things first

As with any version control system, the first order of business [2] is to put something in it. Suppose we start from scratch with a whole new project and assume that some source files and build files have already been created, which require versioning. In the project's top-level directory, the following command makes it a Bazaar stand-alone branch:

```
bzr init
```

There are various options to this command, but the only one that may interest us for new projects [3] is **`--dirstate-tags`**, which gives the branch support for tagging revisions. Tagged revisions are similar in style to CVS tags, being named revisions of the project. I actually prefer the Subversion style of tagging, which is a copy of a branch at a certain point, but for that to work efficiently we need a central repository, which we get to later. For now, the defaults are fine.

So far, we've made an empty branch, so next we need to put any files already in the project under control.

```
bzr add
```

This command recursively adds everything in the current folder to the branch. Well, more specifically, it adds everything that is currently unversioned (unknown) and not ignored. Bazaar reports what it's just done, and it's possible that some files were added at this point that we don't want versioned, and so they can be reverted:

```
bzr revert [filename]
```

and subsequently ignored – see the sidebar. We can also look at the status of the branch, to see which files have been added, removed or modified since the last check-in, and which files are unknown:

```
bzr status
```

Finally, we need to commit the changes to the branch [4].

```
bzr commit -m "Initial project versioned"
```

If Bazaar finishes this with a report that it's committed revision 1, then all is well – the project is now versioned under Bazaar. Note that the **`-m`** command requires a non-empty message – there are some useful things that Bazaar insists upon!

Branchology

Having set up our initial project, we now have a Bazaar stand-alone branch which represents the initial branch: the trunk. It contains the working copy, the branch administrative area and the repository all in one place. There is no need to check-out the working copy in order to work on it – it already exists inside the branch directory. Simply modifying the files here and performing a check-in commits changes to the branch.

To jump ahead of ourselves just a little, let's get some terminology out of the way.

A repository is the database of revisions, and may contain many branches. A branch always has an associated repository, and it is usually in the same place. This is called a stand-alone branch. A branch in a separate location to its repository is called a check-out (see later).

In our simple scanario, the repository contains a single branch, the branch consists of a single revision (the first), and the working copy, the branch and the repository all share the same location – stand-alone. Official terminology aside, I think of a repository as a collection of branches, and a branch as a collection of revisions. Thinking this way makes the whole idea of branching easier to manage.

Project management

A project, in the Bazaar world, is a directory managed by Bazaar. Each branch represents a single project, so where you have multiple projects to control, each would reside in its own branch. To make this all a bit more concrete, consider the directory tree in Figure 1.

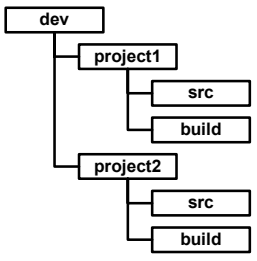


Figure 1

In a centralised VCS such as CVS or Subversion, it would be common practice to version the entire tree from `dev` down, and make that tree the trunk. Checking out an individual project is just a matter of supplying the correct URL of the directory within the repository. Similarly, checking out *every* project is just a matter of asking for `dev` itself.

In Bazaar, it's more common to individually version each project, so in the above tree, `project1` and `project2` would each have their own branch and repository, and `dev` wouldn't be versioned at all.

This is really an efficiency issue, because getting a branch typically creates a local copy of the entire project and branch history; it is this feature that distinguishes distributed version control from centralised version control, allowing a local branch to be merged successfully with a remote one. If you really *need* a local branch of everything, each project needs to be branched individually.

Bazaar Ignores

Deliberately ignoring files in Bazaar works much the same way as for CVS. A global ignore file is located at the user's home directory (which varies for Unix and Windows users – see the bzr documentation), and contains some common patterns. Individual projects may have custom ignores which reside in the text-format file `.bzrignore` in the project's branch root. This file gets automatically added to the branch if you issue the command

```
bzr ignore [patterns]
```

where **`"patterns"`** might be a specific file, or a wildcard match of a set of files, e.g. **`"*.exe"`**.
Items can be un-ignored by editing this file. Whole directories can be ignored too.

STEVE LOVE

Steve Love is an independent developer constantly searching for new ways to be more productive without endangering his inherent laziness. He can be contacted at steve@arventech.com

Branching and merging, pushing and pulling

A primary reason for using distributed version control in the first place is that it allows me to get a local copy of some branch, perform local commits to it, and merge back to the original branch at some later date. Whether the original branch is a public project hosted on the Internet, or is on my home PC and I need a local branch on a laptop, the principle is the same.

My home LAN has a PC named `server`, which shares the `c:\dev` directory as `dev` [5]. It contains several projects managed by Bazaar, which I'll call `paperclip` and `post-it`. I have a laptop – called `laptop` – which I connect to the LAN when I'm at home. Suppose I want to work on the `paperclip` project, and I'll be away from home. From my laptop, I can get a local copy of the branch:

```
bzr branch //server/dev/paperclip
```

Now changes I make to the `laptop` working copy can be committed off-line to the local branch.

Merging

When I am next at home, reconnected, I can **`merge`** the changes I have made to the `laptop` branch into the branch on `server` (where `laptop` also has a `c:\dev` directory shared as `dev`). Note – this command is executed on `server`:

```
bzr merge //laptop/dev/paperclip
bzr status
```

Any local changes in the `server` working copy must be committed before the **`merge`** command can succeed. After the merge, the working copy contains changes from the `laptop` branch. The status of the working copy explicitly shows any pending merges.

Conflicts

If conflicts have arisen as a result of the **`merge`**, they are indicated in the output from the **`merge`** command, and need to be manually rectified before a **`commit`** can take place. The files which conflict can be listed:

```
bzr conflicts
```

Each conflicted file will now have grown some friends with the same base name but different extensions. The original file contains the conflicts in-place, in the usual diff format. The new files are:

- `<filename>.BASE` which contains the file as it existed before the last commit in the current branch.
- `<filename>.THIS` which contains the file with the commits made on the current branch.
- `<filename>.OTHER` which contains the file with the commits made on the branch being merged *from*.

These files are useful if you have an external conflict editor [6].

Once the conflicts have been resolved, Bazaar needs to be told:

```
bzr resolve [file]
```

or

```
bzr resolve --all
```

Finally, the merged changes need to be committed to the repository:

```
bzr commit -m "Merged from laptop"
```

A look at the log shows that not only changed files were merged, but the version history from the remote branch is also available:

```
bzr log
```

The message for the commit of the merge operation is the top entry, followed by the indented commit messages from the remote branch. Branches may be significantly more complex than this, with many levels of branching having occurred in development. Because a merge operation only brings *committed* changes from remote branches, the complexity of the branches isn't an issue.

Bazaar implements Smart Merging: that is, a branch remembers the revision range at which a previous merge occurred. If changes are made to the remote branch, on the laptop, a subsequent **`merge`** from `server` only brings across the changes since the previous merge.

Pulling

Had changes occurred in the `server` branch, they would naturally need to be reflected in the 'remote' branch on `laptop`. From the laptop, I can merge *back* from the main trunk:

```
bzr merge //server/dev/paperclip
bzr commit -m "Merge back from server"
```

If no divergent changes have occurred in the laptop branch, however, these two steps can be performed with a single command:

```
bzr pull //server/dev/paperclip
```

This command makes the current branch a mirror of the branch specified in the command. If there are changes in the current working tree that the other branch does not have, the **`pull`** command will fail with a message informing you that you need to merge from the other branch instead.

Note that **`pull`** doesn't just do a one-pass synchronise and hope that's enough; it's truly a merge of the remote commits including the revision history of each commit on the remote branch, and any branches already merged into it. Making the local branch a mirror of the remote one includes making a mirror of its *history* as well.

Pushing

The reverse is also true if changes have occurred to the `laptop` branch that need to be reflected in the main branch on `server`, as long as no divergent changes exist on the remote branch:

```
bzr push //server/dev/paperclip
```

The **`push`** command automatically commits the changes on the remote branch. As with **`pull`**, if divergent changes exist in the remote branch, the command will fail. There are three solutions to this:

1. Merge back from the remote branch, and then re-do the **`push`**
2. Perform the merge from the remote branch, and check-in
3. The **`push`** and **`pull`** commands have a **`--overwrite`** argument, which has the expected result.

Pushing a branch causes the remote branch to become a mirror of the current one, and as with **`pull`**, it merges the history along with the changes. Both **`pull`** and **`push`** update *both* the target branch and its working copy.

Release management

It is common in development projects to have release branches and tagged revisions to manage releases. A simple use-case is a branching from the main-line of development for a release-candidate, allowing new development to continue on trunk without interfering with the release. When the release is deemed fit for public consumption, it is tagged and named, with any changes merged back to the main-line.

A Bazaar branch is exactly that – a branch – anyway, so creating a release branch, for instance, is a matter of branching from the main-line.

Bazaar supports tagged revisions (in versions of Bazaar after 0.15) if you initialise the branch using the **`--dirstate-tags`** argument to the **`bzr init`** command.

```
bzr tag v0.1_rc1
```

creates a named revision in the branch. The tag is copied along with everything else when the branch is merged, pushed or pulled, or subsequently branched, so it's available to everyone. This follows the CVS mode of tagging, inasmuch as the tag is really just a convenient name for a revision.

In Subversion, a tag is a copy of the tree at a certain revision – as is a branch. Did I already mention I prefer this method to the CVS one?

It should be clear that having multiple branches of a development project just 'lying around' is not the most efficient use of disk space when using Bazaar. The distributed nature of Bazaar brings many benefits, but the

main cost of those advantages is that each branch needs to know the entire history of the branch, which has got to be stored somewhere. A significant project might have several branches representing release candidates, feature exploration, sandboxes and more. Handily, Bazaar mitigates the need to store the *entire* revision history in every branch.

having multiple branches of a development project just 'lying around' is not the most efficient use of disk space when using Bazaar

Shared repositories

It's already been noted that each branch is associated with a repository. Furthermore, several related branches can share a repository, so that the revision history of each is stored in a single place. In the case of a project with lots of branches this can translate to a large saving in disk-space, resulting in an associated saving in the time needed to perform operations on a branch.

A shared repository is not itself a branch, rather it's a container of branches. To create a shared repository, in a directory called `dev`, issue the following command:

```
bzr init-repo dev
```

If you branch to a location inside the repository directory, the new branch will automatically check to see if there is a repository it can share. For example:

```
bzr branch //server/dev/paperclip dev/paperclip
bzr branch dev/paperclip dev/paperclip_fab_new_
   feature
```

will cause both the `paperclip` and `paperclip_fab_new_feature` branches to share space in the repository for their revision information.

Shared respositories also provide the means by which Bazaar can operate as a centralised versioning system. Part of the attraction – for me – of Bazaar is its flexibility, which includes the idea that you may not always want to pay the price for decentralised branches and repositories.

Goto check-out

Now that the concepts of branches, repositories and basic operations are clear (right?) it's time to look at some more advanced cases. If you're more familiar with CVS, Subversion or other centralised version control systems, this section might come as a welcome return to sanity.

In the usual case that we've already covered, a stand-alone branch's working copy can be modified directly, or it can be branched to a different location, resulting in merging, pushing and pulling between the branches. The question arises, of these two (or more) branches, which one represents the main-line, or trunk? The question becomes extremely important where stable releases from predicatable source versions are needed – which is what most people want.

Of course this question can be answered by convention – 'The `paperclip` branch on `server` is the trunk' and releases are made from there. An alternative is to use a centralised repository.

Centralised repositories

Instead of branching from the repository, a check-out is made.

```
bzr checkout //server/dev/paperclip
```

This creates a check-out – including the working tree – called `paperclip` on the local machine. The main difference between this and a regular branch is that changes to the working copy are committed remotely to `//server/dev/paperclip`, not locally.

Bringing the local working tree up-to-date with respect to the repository is similarly familiar [7]:

```
bzr update
```

This, of course, implies the availablity of the remote repository; if it is on a network, then the local machine needs access to the network.

You might think then that a check-out is not as useful as a local, stand-alone branch. The utility of a decentralised version control system is that I can make local commits off-line. However, it can be easier to work with a centralised repository if you commonly work 'connected'. A check-in immediately commits to the remote repository, so there is no need to **`merge`**/**`push`**/**`commit`**. Perhaps more importantly, an update brings *all* commits from a whole team of developers in one go.

The usual work-flow for this case would be:

```
bzr checkout //server/dev/paperclip
```

Work on the project within the working copy. When it's time to commit those changes, first update to get anyone else's changes:

```
bzr update
```

Fix any conflicts resulting from the update, check that the working copy isn't now 'broken', then commit:

```
bzr commit -m "My changes"
```

It is possible to force a check-in to be local only.

```
bzr commit -m "Local changes" --local
```

Such local commits are sent to the repository as a batch when a normal commit is performed (i.e. without the **`--local`** argument).

No working tree

A standalone branch includes its working copy, which is what makes it standalone. A shared centralised repository would not normally have working copies in the branches it manages, since they take up unnecessary disk space, and are a resulting burden on the back-ups (you have them, too, yes?). To work on a shared repository branch you check it out, and make changes to the checked-out working copy.

The Bazaar solution to this is a repository with no working trees, created as an argument to **`bzr init-repo`**:

```
bzr init-repo --no-trees [folder]
```

Branches created within such a repository have only the branch administrative files in the `.bzr` directory. A useful side-effect of this – which we'll expore a bit later when we revisit Release Management – is the idea of nested branches inside the repository.

Unbound branch

Using a centralised shared repository is useful for teams and sole developers alike, but as I've already mentioned, you lose the ability to perfom local commits when you are disconnected from the repository.

Bazaar combats this by allowing you to freely switch back-and-forth between connected and disconnected modes of work. When connected to the remote repository you make a check-out of the branch as already shown. If subsequently you need to disconnect from the network – for example to take your laptop on the train – you can disconnect the check-out from the repository:

```
bzr unbind
```

The effect of this is to make your check-out into a regular stand-alone branch. Commits made to the branch are now made locally, and must be subsequently merged or pushed back to the main repository when you're connected again. Alternatively, if you're likely to remain connected again for a while, you can rebind to the main repository, and check your changes in:

```
bzr bind //server/dev/paperclip
bzr update
bzr commit -m "Remote changes"
```

Therefore, a check-out is effectively the same as a branch that has been bound to a shared repository using **`bind`**.

Lightweight check-out

A normal check-out brings with it most of the branch administration needed to perform some operations without the need for communicating with the remote repository, for example **`status`** and **`diff`** operations can be performed locally. If this is not really a concern, and you will always be connected to the repository – you may even have a check-out on the same machine as the repository itself – you might consider using a lightweight check-out:

```
bzr checkout //server/dev/paperclip --lightweight
```

This results in a much smaller working copy, but one that requires access to the repository for every operation. In addition, a lightweight check-out does not support local-only commits.

Release management revisited

Near the beginning of this article, I made the comment that I prefer the Subversion approach to tagged releases over the CVS – or indeed the Bazaar – method of just naming a revision. We now have all the pieces to make this as efficient as Subversion, and with just one remaining piece, we can make it better than Subversion. Well, in my opinion anyway. To recap:

- Branches in a shared repository share the repository for their revision histories.
- Shared repositories can be created such that the included branches have no (need of a) working copy of their own.
- Merging between branches is simple and efficient.

Given a shared repository with no working trees, in a directory called dev with a project within it called `paperclip`, a Subversion-like tag becomes a branch of the main-line:

```
bzr branch dev/paperclip dev/paperclip_1.0
```

Since the two branches share the same repository, the `paperclip_1.0` branch is a simple copy of the original. If no changes are made to the branch – and it is intended to be a tag, remember, so strictly read-only – this copy will never get any bigger. It does mean we may end up with a repository within which are loads of branches with similar names, all at the same (root) level.

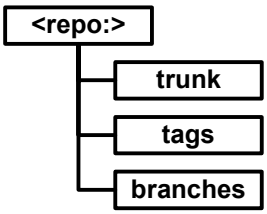


Figure 2

This might suggest that shared repositories are made with the scheme as suggested by the Subversion documentation (Figure 2).

This at least keeps tags, branches and the mainline independent physically of each other, but there is another feature of Bazaar we have yet to see, which makes this structure superfluous.

Nested branches

In a repository created with the **`--no-trees`** option, if we create a branch or directory within another branch, it is still independent of the parent branch. By way of example, consider a regular stand-alone branch. A new directory added to the branch will show up as 'unknown' in a status output – the new directory is part of the branch, just not formally added yet. In the case of the shared repository, the branch *knows* it has no working tree, so directories within it *cannot* be part of the branch.

This is a rather nifty feature when we consider the Subversion repository layout scheme shown above. A typical Subversion repository might look like Figure 3.

To refer to the 1.0 release branch of paperclip might require a URL like:

```
svn://localhost/repo/branches/releases/paperclip/1.0
```

By using nested branches in Bazaar we can create a repository layout like Figure 4. Entries in grey are Bazaar branches. Other items are regular directories. The top-level `paperclip` branch is the trunk, with nested branches beneath it representing releases and features.

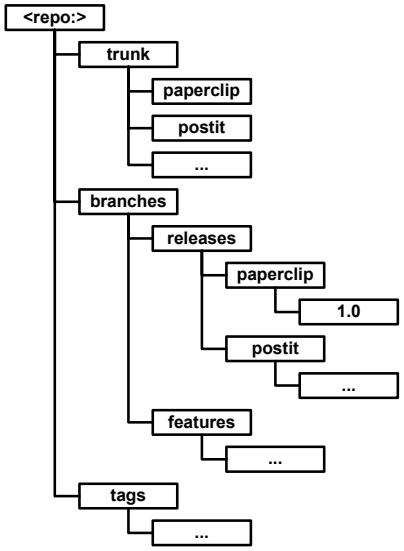


Figure 3

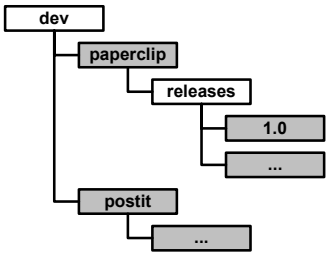


Figure 4

To get hold of the working copy of the main line, we check-out the project:

```
bzr checkout //server/dev/paperclip
```

This brings just the `paperclip` working copy, and does not include the `releases` sub-tree. To check-out the `1.0` release branch we could issue a command like:

```
bzr checkout //server/dev/paperclip/releases/1.0
```

Creating a new branch is similarly intuitive:

```
bzr branch //server/dev/paperclip //server/dev/
   paperclip/features/new_ui
```

Note that this creates a branch directly within the repository, and the new branch needs to be checked-out so it can be worked upon. Merging changes on the branch back to the trunk requires a working copy of the trunk to stage the merge:

```
cd paperclip
bzr merge //server/dev/paperclip/features/new_ui
bzr commit -m "Merge from features/new_ui"
```

Listing which branches (in the Subversion or CVS sense) or tags are available for a project becomes an exercise in looking at a directory on disk.

No one true way

Bazaar supports many ways of working, because ultimately everything is a branch. You can **`merge`** and **`diff`** between any related branches, so how you organise your repository is up to you, and how you think it best for the way you work.

Nested branches can help a great deal with organising the repository, giving you yet more flexibility in how you do it. In the end, how you make the best use of Bazaar is up to you, rather than being imposed by the tool.

Experiences of a First Time Presenter at the ACCU Conference

The title says it all. But did the speaker?

I've just checked and the sun hasn't started shining out of my backside. That's just one of the things that doesn't change, when you go from being an attendee to a presenter at the ACCU conference. I wouldn't like to generalise prematurely though and declare that one's rear end remains unilluminating post OOPSLA, SPA or EuroPython presentation. Maybe next year the committee could put a special swallowable bio-fuelled floodlight in all the speakers hotel rooms instead of this year's Easter egg. Myself and co-presenter Jan Kollhof found that the milky bar egg was a definite step up from the mint chocolate that was refreshed daily on our pillows last year.

It's not just presenting that yields new experiences though, some were simply a product of it being my second year of attendance. New to me was the feeling of slight awkwardness upon meeting someone from last year's conference, who doesn't have the foggiest who you are, despite having shared an "AHA!" moment. "We were stood in the bar, it was 2am! What's-his-face from what-are-they-called introduced us, we chatted for 3 hours!" I was waving my arms wildly at the time. Somewhat symmetrically, moments earlier I was also introduced to the name badge boogie. It's a strange number performed mainly by your eyes with a slight feet shuffle thrown in. It begins when someone comes up to you and starts talking animatedly about a conversation you had at 2am in the bar last year and you swear you've never seen them before in your life. I think it's a skill you acquire over time so next year I reckon I'll be pretty good at it.

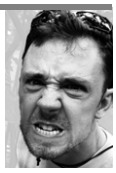
Let me tell you now, I need the fear of public failure and ridicule amongst my peers to motivate me. I have, for a number of years now, wanted to resurrect my interest in artificial intelligence and biologically inspired algorithms. Jan and I had an idea for what we thought was a really interesting project involving distributed computation using JavaScript and Python. We thought that someone else might be interested in it and given that the idea was gestated at ACCU, why not present it the following year? In doing so I would improve my presentation skills, learn to program in Python and learn something by sharing that which we learned. It's also the kind of thing you can put on your CV and show off about to employers.

So what did we learn? We learnt that if you're going to present at ACCU you have to be prepared to go up against some pretty big names, but don't worry about it! Think about it this way: the bigger the competition, the more interested the attendees must be in your talk. I have to admit I did wonder if we'd get anyone at all in our talk but thankfully we got about a dozen and they were all fully engaged. As a result our talk was dynamic and interactive and I thoroughly enjoyed my part in it. Let me drop a hint if you're thinking presenting might be for you, be careful what you name your talk. We thought we'd satirise AJAX by giving ours a web2.0 style acronym, PAAJFDSCICOP. It turns out this was a really bad idea and put people off. Our talk seems to be known as 'the one with the unpronounceable name' and not 'the one about using an ant colony as a model for distributed computation'. We also learnt that the people at ACCU tend to have a certain similarity in attitudes and practices. There's nothing quite like the look of boredom on your granny's face as you tell her how to suck eggs.

So I'm pretty enthusiastic about my presenting experience, it was a winner all round. I learnt, I think the audience learnt and now I can get cheap or free entry to a bunch of conferences by hawking our talk around the circuit. If I can do it, you certainly can. I urge you to just think about something that you get excited about and give it a whirl. What have you got to lose? Nothing, oh apart from having a room full of people point and laugh at you. I'm kidding of course, no one points and laughs they just rib you in the bar afterwards. ■

ED SYKES

Ed writes software for the telecoms industry where he's interested in getting the most out of people and computers. When not writing software, he's in his studio writing music. He can be contacted at ed.sykes@gmail.com



The Bazaar Approach (continued)

Next time...

There is yet more Bazaar stuff to explore:

- More on the format of the bzr log, and how to track revision history.
- Some more uses of lightweight check-outs.
- Nested branches turn out to be useful beyond release management when we look at by-reference dependent branches.
- Bazaar has its own smart server – similar in nature to the Subversion server svnserve – which helps to restrict remote access to just your Bazaar repositories and minimise network round-trips.
- Bazaar supports plugin components which extend its core functionality, a few of which are indispensable.
- Using the Bazaar configuration files to configure Bazaar itself, and customise the behaviour of some of the plugins. ■

Acknowledgements

Many thanks to Tim Penhey and Pete Goodliffe for their comments on drafts of this article.

References

[1] 'The Bazaar Thing', *C Vu 19.4*, August 2007.

[2] OK, so you need to actually *get* Bazaar first – http://www.bazaar-vcs.org is the place for that.

[3] Unless you specifically need to allow for support of old bzr clients.

[4] Many of the Bazaar commands have aliases which will be familiar to CVS/SVN users, e.g. `ci` for `commit` and `co` for `checkout`. You can even create your own aliases.

[5] Bazaar is cross-platform – written in Python, it will run anywhere Python can run. The examples shown are for the MS Windows version, but note that I've used the `//` convention here for network shares. The Bazaar commands are the same whichever platform you run, and you can still use `\\` if you prefer.

[6] I personally still use TortoiseMerge for Windows – part of the TortoiseSVN project – for conflict editing, even though it's supposed to be for Subversion it handles Bazaar conflict files just as easily.

[7] Note that update works for regular branches as well. It brings the working copy up-to-date with respect to its associated branch – in the case of a central repository, the branch is remote.

Please Release Me #2

Pete Goodliffe continues to throw his software out the door.

In the previous column [1] we began a foray into the unnecessarily murky world of software releases – a simple act that many software houses get monumentally wrong. We discovered what a software release is, the common types of software release, and worked out some prerequisites for a good software release. In this article we'll continue to explore this territory, and investigate the mechanics of releasing the code that we craft.

For those who have forgotten, or not read, the previous article, I have defined a software release as: the process of obtaining, building, packaging, and deploying/distributing a versioned software product, for an agreed purpose.

It's probably already clear that making a software release requires a high level of care and particular eye for detail. Essentially, it requires you to be anal. But we all needed an excuse to be anal, right? A good software release process ensures that we have:

- an audit trail – we can see what went into an important external build from three years ago to help us track bugs in it,
- a repeatable software build procedure,
- a reliable and understood release procedure,
- a person designated to make a release (ideally picked from a pool of people who know how to do it – never rely on one individual),
- enough time to do it in,
- enough time to test the release, and
- a reliable publish/deployment mechanism.

These will all be addressed in this article. I'm sure you can't wait...

very few programmers will slave over a codebase for months without actually intending to release it at some point

When do we make releases?

Before we start to make releases, we need to know when to do so. This is a process issue, but it is so important that we need to mention it here – if only briefly.

A software release should never take you by surprise. Your development process must plan for it – very few programmers will slave over a codebase for months without actually intending to release it at some point. There should always be an 'end goal' in sight.

You should plan to make a release when the codebase is ready. That might mean:

- All of the outstanding features slated for this release have been implemented
- All of the code's unit tests pass (and of course, you are enlightened programmers – aren't you – and everything is unit tested) so you know the code is mechanically sound at the unit level
- All automated integration tests pass (of course, you have these too, don't you?)
- All the bugs have been removed, or any remaining known bugs are deemed not to be 'show stoppers' by the powers that be (officially these are sold to customers as features)

... and ...

- The development organisation (e.g. the company or open source project) are ready to make the release (the marketing and distribution channels are ready, for example).

It's quite a dance to get all of these things lining up at once, and how to do so is well outside the scope of this article (although I can recommend a good book that will help with this!) Clearly there is a large element of planning involved with software releases. Cue software release golden rule number one:

> A software release should never take you by surprise.

But (there's always a 'but'), sometimes this ideal might not be possible. For example, if an important customer discovers an urgent problem in a release product and is clamouring for a quick fix, you might investigate the fault, fix the bug, perform a quick test, and issue a release in a very short cycle (this depends in part on how thorough your testing procedure is – and how willing the customer is to accept code without a full regression test cycle).

There is another important angle to the 'when do you make a release' question: you should be making releases all the time. Perhaps this sounds counter-intuitive, but it's a really good idea. Even if your releases are not going outside the development team, you should make full production-worthy releases often. There are many benefits:

- It proves that the build systems are working perfectly, and that no one has broken the build scripts.
- It proves that you can repeatedly release working software, and that your release process does not include an element of chance (too many release processes I've seen rely on luck, whether it's a developer remembering some sketchy build runes, or a build machine hard drive that might occasionally fill up and silently corrupt the build).
- It gives you a chance to practise the release process at 'unimportant' times, so that 'important' releases can't possibly go wrong.
- In the run-up to a major release, regular internal interim releases can provide a development heartbeat to help focus programmers' minds towards the end goal.
- Regular 'releases' afford new developers the chance to run through the build procedure so that everyone knows what's involved. Then you can be assured that you'll never be left disaster planning when Buildmaster Fred walks under a bus, or is hit by a cruise missile.
- When we perform 'practice runs' we learn how long the release process takes, so when we have to make an important release, we know how long to schedule and can avoid a last-minute rush (and the inevitable consequent silly mistakes).

PETE GOODLIFFE

Pete Goodliffe is a programmer who never stays at the same place in the software food chain. He has a passion for curry and doesn't wear shoes. Pete can be contacted at pete@cthree.org

- Interim releases are great to hand off to the product test team, or external beta testers to get a feel for the quality and stability of in-development code. Doing this regularly also allows us to determine how long a reasonable test (or regression test) phase should take.

So that's the rationale for golden rule number two:

> Make regular software releases, even when no major external releases are imminent.

'Regular' for you might be hourly, daily, weekly, or even monthly, depending on: your organisation, the codebase, the build technology, and the the quality of your software release process. It's important to do this as frequently as you possibly can.

How to make a software release

So here it is. The nitty-gritty how-to section. In about as much detail as we can stand, this is the theologically correct way to make a software release. I've distilled 'good practice' into the five 'P's of software release.

1. Plan it

We've already looked at what it means to plan a release. But it's important to understand that a release does not start when the code is deemed 'ready to ship', but long before. The planning of a release answers the following questions:

- What software will be released. This sounds like a dumb question, but often when you have a codebase that can build multiple product variants it can be spectacularly easy to make a small mistake with comic (or tragic) consequences.
- The specific version of the software that the release will be built from. This is the exact source control location of the software – the branch and version number on the branch (details vary depending on the version control system you are using). Now is not the time to preach about source control, but this presumes that you are already following golden rule number three:

> Software releases must be made from code held in a source control system.

there should be very little variation in the actual release procedure, just in the final destination of the software release

This provides a clear audit trail (so you can see what changed leading up to this release, and exactly which bugs have – and haven't – been fixed), an unambiguous description of the code to be built (the location and version number in the repository), and an obvious way of going back to find the exact code that went into historical release at any time in the future. Never, ever, release code that isn't stored under source control. OK, sorry, I preached.

- Who will say when the code is ready (a selection of product managers, QA engineers, developers).
- When will they say that it is ready – and how you will know? Too many software factories have such bad lines of communication that one developer doesn't know what another is doing, let alone one department and another.
- Where the release is going once it's made. Is this an internal 'test' release, a 'release candidate' that will be sent to beta testers and/or sent for acceptance tests, or is this the final real-deal release? Will this release be distributed internally, pushed to an update server, put on a public webserver, or sent to a CD mastering house to be manufactured onto physical media and distributed to retailers?

2. Prepare it

By now you know that it's time to build your release, but first there are a few preliminary steps. Before you get anywhere near a compiler you must mark the code that the release will be built from in your source control system. In many revision control systems this means tagging (or labelling, depending on nomenclature) the set of files to be released [2]. Pick the files from the appropriate branch – many times you will have a specific code branch dedicated to the release code.

You can mark the code from anywhere; you might make a clean checkout to create the branch, or you might carefully do this in a developer's personal workspace. Better still, many revision control systems allow you to do this outside of a code checkout, and for simple tag operations this is best.

So now you have the source set that will be used to make the release unambiguously marked before you've started to build it. (Remember that you've got to be anal – now check that you have the right set of files marked!). There is a good reason to do this step *before* you make the release. Other developers may continue to work on the release branch, and if they performed a checkin as you were building the release there could be some confusion as to which exact code set you built the release from. Marking the code first avoids this confusion.

3. Produce it

Now, on a 'clean' computer [4] make a fresh check out of the release tag you made in the previous step. Start from scratch. Nothing should be reused from a previous release build – even if you are sure that it hasn't changed. Delete all the old stuff and go back to square one.

> Always build your software in a fresh checkout, from scratch. Never reuse old parts of a software build.

Now build it. What 'build' means to you depends on the kind of code you've got. C and C++ code needs compiling into libraries and executables. Java code needs byte-compiling. Many other languages don't have such a requirement, but more often than not they do have some kind of 'build' step, to prepare the software for release.

Release Branches

Good version control is a topic that I've covered in a previous set of C Vu articles [3], but it's worth reminding ourselves about 'release branches'.

You can make a branch in your version control system for just about any reason: to work on a new feature independant of the rest of the codebase, or to work on a bugfix or exploratory refactor. Here you make the branch to encapsulate a set of changes, until they are 'done'. When you've completed this kind of work, you will typically merge the branch back onto the mainline (or trunk). Standard stuff.

Release branches are often made for the exact opposite reason – to mark a point in the codebase where you want to ensure stability. Once you have made a branch for your software release then other new features can be developed on the mainline of code development without fear of compromising the impending software release. The QA department can perform regression testing of the branch code without worrying that any new work will move the goalposts under their feet. Very occasionally an important bugfix from the mainline development may get merged into the release branch, but each such change is made carefully, with the consent of all people who have an interest in the quality of the software on that branch.

Typically you will not merge a release branch back down onto the trunk, as you perform no fresh development on it. All development occurs on the trunk, is sanity tested there, and then merged onto the release branch.

The quality of this 'Produce it' step depends entirely on the quality of your build system. It should be designed to provide absolutely no room for human error:

- The best build systems work in a single step – one command line call, or a single IDE button press.
- Any build configuration (e.g. makefiles, build scripts, or IDE config files) MUST be checked into source control and used to generate the build.
- There should be no requirement for a user to select build types (debug/release/full/demo/whatever), update build identification strings, or to fiddle with anything.

The more manual steps your build process has, the more room there is for human error. All too often, the build is pushed out to the very last minute and rushed mistakes here are very common. More manual steps also require more documentation, and this documentation is open to misinterpretation.

> Automate your build process as much as possible. Ideally it should require a single step.

An automated build system has less room for error and doesn't need reams of documentation (as the build script is an unambiguous description of how to build the release). Once you have an automated build system then it is then very easy to run the build system automatically every night. It's great to be able to come into work in the morning and find a full release of your software waiting for you. In many projects the releases are actually taken from a nightly build server, rather than built manually!

This step should include building any installer images that tie together all the programs, graphics, registry keys, script files and other assorted miscellany that constitutes an entire release. Ideally these installers should all also be generated by the one single build command, and have appropriate configuration set up by the build process (e.g. version numbering is adjusted automatically).

4. Package it

The output of your build process will often be a selection of files (a number of executables, for example), or it may be a single Windows installer program. Whatever you've just built there is another consequent step – to package it up in the file format that the user will receive it in.

Often this is not treated with as much importance as the previous 'build' step. Even though we're now on the easy home-stretch sloppy mistakes here can ruin everything. Packaging the code requires:

- Constructing a set of release notes that describe the changes that compose this release, the end-user licence information, installation instructions and other useful information.
- Constructing the final distributable image, perhaps a CD ISO image, an update server package, or a downloadable zip file.

Again, this step should ideally be done automatically, although it is not always practical. For example, it is perfectly possible to automatically document the changes that have occurred in this release by pulling log information out of your source control system. However, these log messages are usually developer-oriented drivel and need distilling into a user-friendly form.

it's great to be able to come into work in the morning and find a full release of your software waiting for you

5. Publish it

The final step. You've made the release. Now release it!

Every release should be archived in a well-known place for later retrieval. Perhaps this is an archive drive on the network, or perhaps your company lodges physical CDs in a 'drawing office'. Make sure that you have a controlled copy of your release lodged somewhere where it will not get lost.

Then – publish. Whatever that means for you. And tell people about it. Then have a nice cup of tea and a sit down.

Is that it?

Clearly constructing a software release is not something that happens in one day. The act of preparing a release spans the entire development process. You've been developing with the release in mind from the very beginning. The release procedure was already known when you started development, don't try to work it out in a mad rush at the end.

Most problems with software releases come for that very reason – they are rushed out at the last minute. No one worked out how to do it properly, and so when the release needed to be made it was cobbled together in an ad-hoc way. But we all know better now, don't we?

And we have another great reason to be anal. ■

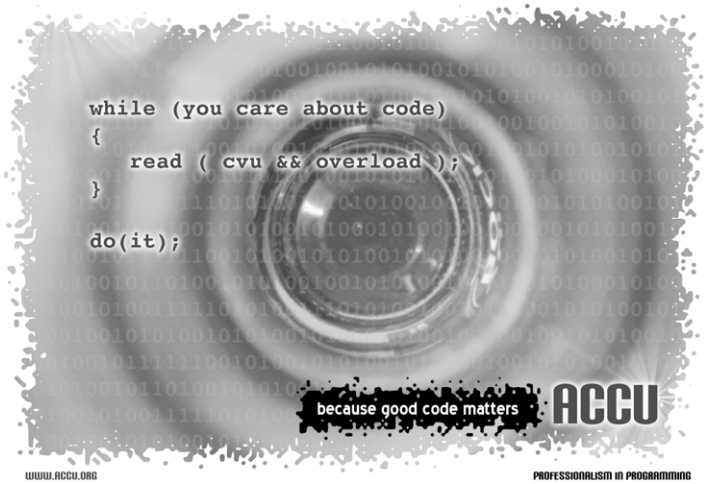
Notes

[1] 'Please release me' (Pete Goodliffe). In: *C Vu* Volume 19 Issue 4.

[2] In subversion, for example, you needn't actually label the code – you can just remember the revision number of the filesystem at the release point. However, it is still good practice to create a tag in the repository, as this records the release's build point unambiguously in the one place that it can't get separated from the source code. Perhaps this tag would only be made for major external releases, though, rather than for every internal interim release build.

[3] 'Effective Version Control' (Pete Goodliffe). In: *C Vu* Volume 18 Issues 4-6.

[4] Often we set aside a computer specifically for release builds – the build server.



Tracing Function Calls Using Python Decorators

Thomas Guest hears the echo of a python.

Let's suppose you want to trace calls to a function. That is, every time the function gets called, you want to print out its name and the values of any arguments passed by the caller. To provide a concrete example, here's a function which calculates the dot product of two vectors.

```
def dot(v, w):
  """ Calculate the dot product of two vectors.

  Example:
  >>> dot((1, 2, 3), (3, 2, 1))
  10
  """
  return sum(vv * ww for vv, ww in zip(v, w))
```

To trace calls to the function you could just edit it and insert a print statement.

```
def dot(v, w):
  print "dot(v=%r, w=%r)" % (v, w)
  return sum(vv * ww for vv, ww in zip(v, w))
```

When you no longer want calls traced you can remove the **print** statement or even comment it out. This approach works well enough for a while but you soon discover there are more functions you want to trace; and you'll eventually end up with lots of functions being traced and lots of commented-out tracing code. You might even end up with broken commented-out code:

```
def dot(vec1, vec2):
  # print "dot(v=%r, w=%r)" % (v, w)
  return sum(v1 * v2 for v1, v2 in zip(
     vec1, vec2))
```

At this point, you realise that calling a function and tracing these calls are orthogonal operations. Isn't there a less invasive way to do this?

A less invasive way

Rather than change the original function you can simply wrap it with the code which prints out the inputs (Listing 1).

Listing 1

```
def echo_both_args(fn):
  """ Returns a traced version of the input
      2-argument function.
  """
  def wrapped(arg1, arg2):
    name = fn.__name__
    print "%s(%r, %r)" % (name, arg1, arg2)
    return fn(arg1, arg2)
  return wrapped

def dot(vec1, vec2):
    ....
dot = echo_both_args(dot)
```

The **echo_both_args** function accepts a single function as a parameter and returns a new function which wraps this original function by adding the desired trace functionality. By the way, I've introduced the term 'echo' for this action rather than the more conventional 'trace' since Python already has a **trace** module which does something rather different. The idea is that when you call a function, you hear your call repeated – like an echo.

C/C++ programmers will have noticed the inner function object, **wrapped**, which **echo_both_args** returns. Returning inner functions is a common Python idiom, and is the way language implements the closures found in other high-level languages such as Scheme or Ruby. The inner function doesn't go out of scope – in Python, objects persist as long as needed.

The final line of the code snippet simply rebinds **dot** to the echoed version of itself. We don't need to pass the name of the function to be traced (**dot** in this example) into **echo_both_args** since in Python a function object carries its name around with it in a **__name__** attribute. If we now call **dot** in an interpreted session, we'll see the call echoed:

```
>>> dot((1, 2, 3), (3, 2, 1))
dot((1, 2, 3), (3, 2, 1))
10
```

The inner function, **wrapped**, should be capable of wrapping any function – so long as that function accepts just two parameters, that is. (Note the use of the **%r** print formatter which converts any Python object into its canonical string representation). If we want a more general purpose **echo** which can wrap any function with any signature, we might try something like Listing 2.

Here, the inner function, **wrapped**, has parameters **(*v, **k)**, which contain the positional and keyword arguments passed to this function respectively. It doesn't really matter how many arguments the function being wrapped has, or indeed if this function itself accepts arbitrary positional and keyword arguments: it just works – have a look in the Python

Listing 2

```
def echo(fn):
  """ Returns a traced version of the input
      function.
  """
  from itertools import chain
  def wrapped(*v, **k):
    name = fn.__name__
    print "%s(%s)" % (
      name, ", ".join(map(repr, chain(
         v, k.values()))))
    return fn(*v, **k)
  return wrapped
```

THOMAS GUEST

Thomas is an enthusiastic and experienced programmer. He has developed software for everything from embedded devices to clustered servers. His website is http://www.wordaligned.org Contact him at thomas.guest@gmail.com

Listing 3

```
def f(x): pass
def g(x, y): pass
def h(x=1, y=2): pass
def i(x, y, *v): pass
def j(x, y, *v, **k): pass

f, g, h, i, j = map(echo, (f, g, h, i, j))

f(10)
g("spam", 42)
g(y="spam", x=42)
h()
i("spam", 42, "extra", "args", 1, 2, 3)
j(("green", "eggs"), y="spam", z=42)
```

reference manual for details. We've used **chain**, one of the handy iterator tools from the **itertools** module [1], to iterate through all the positional and keyword argument values; then a combination of **string.join** and the **map** and **repr** built-ins produce a string representation of these values.

We can now define, wrap and call some functions with more varied signatures (Listing 3).

Running this code produces output:

```
f(10)
g('spam', 42)
g('spam', 42)
h()
i('spam', 42, 'extra', 'args', 1, 2, 3)
j(('green', 'eggs'), 'spam', 42)
```

This does something like what we want, but note a couple of problems:

- the second call to **g** uses keyword argument syntax, and the output **g('spam', 42)** is exactly what we saw echoed by the previous call to **g** even though the arguments have been swapped around. We'd better try and echo argument names as well as their values (just as we did when we first modified **dot** to **echo** calls).
- the output doesn't show the (defaulted) arguments passed to **h** [2].

There's another more subtle problem too. (Listing 4)

That is, by wrapping **f**, we've tinkered with its identity: its name has changed and its documentation has disappeared. This is bad news in a dynamic language, since any code introspecting **f** will get the wrong idea about it. We should really have modified **wrapper** to copy various attributes from **f**.

Decorators

At Python 2.3, the **@** character was pulled out of the bag to provide a special syntax for decorating functions in this way. Rather than retrospectively rebinding a function **f** as shown in the examples so far, we can explicitly decorate **f** up front with the **echo** wrapper like Listing 5.

Listing 4

```
>>> def f(x):
...     " I'm a function called f. "
...     pass
...
>>> f.__name__
'f'
>>> f.func_doc
" I'm a function called f. "
>>> f = echo(f)
>>> f.__name__
'wrapped'
>>> f.func_doc
None
```

Listing 5

```
>>> def echo(fn):
...     from itertools import chain
...     def wrapped(*v, **k):
...         name = fn.__name__
...         print "%s(%s)" % (name, ",".join(map(
...             repr, chain(v, k.values()))))
...         return fn(*v, **k)
...     return wrapped
...
>>> @echo
>>> def f(x): pass
...
>>> f('does it work?')
f('does it work?')
```

Of course decoration isn't reserved for the particular use case shown in this article: we might decorate functions to time their execution, to protect them against simultaneous access from multiple threads, to check the validity of input arguments, to dynamically analyse code coverage, and so on.

Note also that there's a special decorator in the **functools** module [3] which does the job of making a wrapper look like its wrappee. (Listing 6)

Listing 6

```
>>> import functools
>>> def echo(fn):
...     @functools.wraps(fn)
...     def wrapped(*v, **k):
...         ....
...     return wrapped
...
>>> @echo
>>> def f(x):
...     " I'm f, don't mess with me! "
...     pass
>>> f.__name__
'f'
>>> f.func_doc
" I'm f, don't mess with me! "
>>> f(('spam', 'spam', 'spam!'))
f(('spam', 'spam', 'spam!'))
```

Improving echo

Listing 7 is a version of **echo** which fixes the remaining problems: that is, it prints out argument names as well as argument values, and it also prints out any arguments defaulted in the call to the wrapped function. This version is a little longer and considerably more fiddly [4], since we need to dig deeper into the function's code, but it follows the same basic pattern as its predecessors.

While we're improving things, note that we've also parameterised the **write** function, rather than naively hijacking **sys.stdout** to print out function calls.

Listing 8 shows the new version being tried and Listing 9 the resulting output.

As you can see, it works on methods and classmethods too.

By the way, if you're not familiar with classmethods, they're created using Python's built-in **classmethod** function, which transforms functions in the required way; and, as shown in the example above, the decorator syntax is the recommended way of applying this transformation.

Wider application

Using this final version of **echo** to decorate functions is simple enough. All you have to do is precede the function(s) you want to echo with the **echo** decorator. What, though, if these functions are in a module you don't want to modify – one of the standard library modules, for example? What if you want to echo an entire class by echoing all of its methods without

Listing 7

```
import sys

def format_args(args):
  """ Return a string representing a sequence of
      function arguments.

      Each argument is either a (name, value)
      pair or simply a value.
      >>> format_args((('x', 2), ('y', 'spam')))
      "x=2, y='spam'"
      >>> format_args(1, 2, 3)
      1, 2, 3
  """
  def format_arg(arg):
    try: return "%s=%r" % arg
    except TypeError: return "%r" % arg
  return ", ".join(map(format_arg, args))

def echo(fn, write=sys.stdout.write):
  """ Echo calls to a function.

      Returns a decorated version of the input
      function which "echoes" calls made to it by
      writing out the function's name and the
      arguments it was called with.
  """
  import functools
  from itertools import chain, izip

  # Unpack function's arg count, arg names,
  # arg defaults
  code = fn.func_code
  argcount = code.co_argcount
  argnames = code.co_varnames[:argcount]
  fn_defaults = fn.func_defaults or list()
  argdefs = dict(
     zip(argnames[-len(fn_defaults):],
     fn_defaults))

  @functools.wraps(fn)
  def wrapped(*v, **k):
    formal = izip(argnames, v)
    defaulted = ((a, argdefs[a]) for a in
       argnames[len(v):] if a not in k)
    args = chain(formal, defaulted, v[argcount:],
                 k.items())
    write("%s(%s)\n" % (fn.__name__,
       format_args(args)))
    return fn(*v, **k)
  return wrapped
```

editing the class itself; or all the classes and functions in a module, again without editing the module itself?

A little introspection works the required magic. All we have to do is work our way through all the functions and classes in a module, and all the methods in these classes, rebinding these functions to their echoed versions. Extra attention is needed for class-, static- and private-methods, but otherwise this code contains few surprises (Listing 10).

Work in progress

I've used the **echo** library successfully on a number of occasions. Usually, I simply want to capture all function calls made to a module. Echoing the module generates a log file which I can then examine offline using the standard unix shell tools.

I've resisted the urge to add options to fine-tune exactly which functions get echoed, or to control whether or not private methods get echoed – I prefer to generate a surplus of information then filter it as needed. (Though note that the code, as it stands, *doesn't* echo nested functions and classes.)

Listing 8

```
@echo
def f(x): pass
@echo
def g(x, y): pass
@echo
def h(x=1, y=2): pass
@echo
def i(x, y, *v): pass
@echo
def j(x, y, *v, **k): pass
class X(object):
  @echo
  def m(self, x): pass
  @classmethod
  @echo
  def cm(klass, x): pass

def reversed_write(s):
   sys.write(''.join(reversed(s)))
def k(**kw): pass
k = echo(k, write=reversed_write)

f(10)
g("spam", 42)
g(y="spam", x=42)
h()
i("spam", 42, "extra", "args", 1, 2, 3)
j(("green", "eggs"), y="spam", z=42)
X().m("method call")
X.cm("classmethod call")
k(born="Mon", christened="Tues", married="Weds")
```

I discovered that if a class customises the special **__repr__** method, then trying to echo calls to this method leads to recursion and a runtime error. As a consequence, **echo** avoids echoing **__repr__** and, for good measure, **__str__** as well. I suspect that **echo** would lead to similar problems if applied to another introspective module. Echoing **doctest** wouldn't be very clever, and echoing **echo** itself is surely doomed. ■

Source code

The code for this article is available using anonymous SVN access at http://svn.wordaligned.org/svn/echo. A Trac project supporting this article is available at http://preojects.wordaligned.org/trac/echo

Thanks

Thanks to Dan Tallis, Kevlin Henney and all at CVu for their help with this article.

Listing 9

```
f(x=10)
g(x='spam', y=42)
g(y='spam', x=42)
h(x=1, y=2)
i(x='spam', y=42, 'extra', 'args', 1, 2, 3)
j(x=('green', 'eggs'), y='spam', z=42)
m(self=<__main__.X instance at 0x7837d8>,
   x='method call')
cm(klass=<class __main__.X at 0x785840>,
   x='classmethod call')
)'seuT'=denetsirhc ,'sdeW'=deirram ,'noM'=nrob(k
```

Decoration, annotation, invasion and the @ symbol

Other languages also lay claim the to the term 'decorator', and indeed use of the `@` symbol for this purpose. For example, a statically-typed language such as C++ can utilise the DECORATOR [5] pattern to adapt objects at runtime: the classic example would be a window in a graphical user interface, which may be dynamically decorated with (combinations of) borders, scroll-bars, and so on.

Java makes special use of the `@` symbol in its annotations, a language feature related to Python's decorators. You'll also find the `@` symbol used in Java comments, where it augments the code without changing its runtime-behaviour, enabling the Javadoc [6] tool to generate better documentation. I've seen this idea stretched further by the XDoclet [7] family of tools which perform tag-based tricks on Javadoc-style comments in order to generate boiler-plate code – though in this case my response is closer to bewilderment than wonder.

Elevating the concept of decoration further brings us to aspect-oriented programming, a paradigm which provides language/framework support for the separation of concerns – and if that sounds like a collection of buzzwords, it's because I don't know much about it!

For some readers, Python's ability to tamper with classes and functions in ways unforeseen by their original implementer may seem dangerous, and I regard this a fair reaction. Even at an event as Python-friendly as the recent PyCon UK [8], some present argued that static languages were a safer choice for very large projects, primarily because such languages simply don't allow you to write code such as **`change_everything(world)`** – which is really what **`echo_module(module)`** does.

Meta-programming techniques like these need taste and restraint. As we've shown, in Python, the `@` symbol is pure syntactic sugar; sweet as it may seem, we should use it to improve, not impair.

Listing 10

```
import inspect
import sys

def name(item):
  " Return an item's name. "
  return item.__name__

def is_classmethod(instancemethod):
  """ Determine if an instancemethod is a
      classmethod. """
  return instancemethod.im_self is not None

def is_class_private_name(name):
  """ Determine if a name is a class private
      name. """
  # Exclude system defined names such as
     __init__, __add__ etc
  return name.startswith("__") and not
     name.endswith("__")

def method_name(method):
  """ Return a method's name.
      This function returns the name the method
      is accessed by from outside the class (i.e.
      it prefixes "private" methods
      appropriately).
  """
  mname = name(method)
  if is_class_private_name(mname):
     mname = "_%s%s" % (name(method.im_class),
     mname)
  return mname

def format_args(args):
  ...

def echo(fn, write=sys.stdout.write):
  ...

def echo_instancemethod(klass, method,
   write=sys.stdout.write):
  """ Change an instancemethod so that calls to
      it are echoed.
      Replacing a classmethod is a little more
      tricky.
      See: http://www.python.org/doc/current/ref/
      types.html
  """
  mname = method_name(method)
  never_echo = "__str__", "__repr__",
  # Avoid recursion printing method calls
  if mname in never_echo:
    pass
  elif is_classmethod(method):
    setattr(klass, mname, classmethod(
       echo(method.im_func, write)))
  else:
    setattr(klass, mname, echo(method, write))

def echo_class(klass, write=sys.stdout.write):
  """ Echo calls to class methods and static
      functions
  """
  for _, method in inspect.getmembers(
     klass, inspect.ismethod):
    echo_instancemethod(klass, method, write)
  for _, fn in inspect.getmembers(
     klass, inspect.isfunction):
    setattr(klass, name(fn),
            staticmethod(echo(fn, write)))

def echo_module(mod, write=sys.stdout.write):
  """ Echo calls to functions and methods in a
      module.
  """
  for fname, fn in inspect.getmembers(
     mod, inspect.isfunction):
     setattr(mod, fname, echo(fn, write))
  for _, klass in inspect.getmembers(
     mod, inspect.isclass):
     echo_class(klass, write)
```

Listing 10 (cont'd)

Notes and references

[1] http://www.python.org/doc/current/lib/module-itertools.html 'itertools – Functions creating iterators for efficient looping'

[2] Strictly speaking, the default arguments aren't 'passed' to the function; they're stored by the function object when the function code is parsed.

[3] I suspect that inside this rather contorted function a simpler version is begging to get out. I'd be pleased to accept any suggestions.

[4] http://docs.python.org/lib/module-functools.html 'functools – Higher order functions and operations on callable objects'

[5] A description of the Decorator pattern can be found in *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

[6] http://java.sun.com/j2se/javadoc 1Javadoc is a tool from Sun Microsystems for generating API documentation in HTML format from doc comments in source code.

[7] http://xdoclet.sourceforge.net 'XDoclet is an open source code generation engine.'

[8] http://pyconuk.org/index.html 'The Python UK Conference'

On The Powell Cable Car at JavaONE 2007

Peter Pilgrim proposes Java is still a major player.

In early May, yours truly attended the annual JavaONE Conference in San Francisco, USA. This event is truly the centre of the Java universe. It has the same heartbeat and swell of excitement as the annual Apple WWDC or the Microsoft Tech Ed conferences. This year over 15,000 attendees gathered together under one roof. JavaONE takes place over four days and features 250 technical sessions on Java technology, scripting, open source, Web 2.0 and more. It is the biggest Java conference world-wide, in its twelfth year, and it is (under)written and produced by Sun Microsystems. We had motivational talks by Bob Brewin, the CTO of Sun Software; Rich Green, Executive VP of Sun; Jonathan Schwartz, CEO of Sun; and James Gosling, Sun VP and father of Java. We had also presentations from British developers such as Joe Walker and Stephen Colebourne. Developers such as Geert Bevin, creator of the RIFE framework, represented many Europeans who travelled over too.

The message of this year's conference was that Sun is focusing on the end user's computing experience by moving Java back to the desktop. Some will say it's about time. JavaFX is Sun's attempt to take Java to the typical web content programmer/designer. Also JavaFX is the renamed product for a 20% computing-time research project, dreamt up by Sun employee, Chris Oliver. Chris invented a self-named scripting language called F3 (borrowed from the phrase 'form follows function'; a modern architectural principle coined by Louis Sullivan), which made it easier for programmers to generate and control graphic elements in 2D. Using JavaFX you can easily associate graphic elements together, which are typically created in graphic content programs such as PhotoShop. You can take these elements, combine, program against them and create moving animations. In contrast, creating the equivalent code in the standard Java language, with the Java 2D API, would take far longer. With JavaFX, Chris and Sun hope to reinvigorate Java on the desktop. They also wish to capture the attention of web designers who are already conversant in Adobe Flash and ActionScript.

for a significant part of the global populace, a mobile phone is their sole connection to the wider Internet

Sun also announced JavaFX Script for mobile phones and other devices. You would be astounded by how many mobile phones out there that are Java compatible and can run the Java Virtual Machine. For a significant part of the global populace, a mobile phone is their sole connection to the wider Internet. So this explains why mobile devices are a very important market for Sun and the Java ME platform.

However, Sun has two well-known competitors in the market. Microsoft has its new SilverLight offering, and Adobe has its long standing Flash, Flex Core and Flex Data Services. Let's face it, Sun failed to set the world alight with its now decade-old Java Applet technology. So now in order to defend its own markets, it is computing with JavaFX. The reasons for the unpopularity of applets are three-fold: poor deployability, weak usability and straightforward performance. Java is perceptibly slow for most Internet users and that is a shame, even for Groovy developers. So at the conference, Sun announced its intention to fix the nagging JVM issues.

As one of the key notes of the conference, Jasper Potts and Kenneth Russell demonstrated IRIS, a new photo-sharing and uploading mash-up application. IRIS is hybrid of Java applets, HTML, CSS and AJAX integrated with the popular Flickr web services. Basically you can drag a JPEG photo from your desktop and drop it into an applet that resembles a camera shutter (hence the name, Iris), and the application then uploads your digital photo to your Flickr account. In their presentation, Jasper and Kenneth showed basic editing of the photos: sharpening, blurring, cropping, resizing and colour correction. Afterwards the results were saved back to Flickr. IRIS sports another brilliant applet that produce a fantastic full 3D slide show at full screen size. This was stunning graphical eye-candy, because this pure Java Applet displayed rotating cubes with texture maps of the digital images, with hardware acceleration. What amazed us all was that the Sun engineers found a way of getting this functionality running with Java Runtime Environment 1.4 (circa 2002). IRIS, therefore, will execute byte-code on earlier Java editions that are already installed in corporate environments. The rub of the green is that the engineers have still got to document their fabulous new techniques and release the demo as open source.

The second great demo of the conference was NASA Whirl Wind, which is in fact open source and available. Patrick Hogan, the leader of the project literally showed us the world. Whirl Wind is a Java Bean, which provides a 3D mapping of the entire globe (or for that matter any other spherical body like the moon or even Mars). It is very similar to Google Earth, which is a native desktop Windows application, except that Whirl Wind is a standalone visualisation component written in 100% Java. It can be integrated into either an applet or application. NASA's Whirl Wind is an awesome graphical application. The smooth zooming and rendering of the surfaces, whilst navigating across the globe has to be seen to be believed. It demonstrates the sheer power and speed of Java and OpenGL 3D API bindings and I bet someone out there soon will find a great use for this tool kit in a mash-up. Trust me – it hasn't been possible to describe Java as slow for a few years now.

Sun also announced that it will start addressing the download issues for Java inside the browser. There will be a Consumer JRE that will be modularised, in order to reduce the size of the Java download experience. This will also increase the speed-to-launch for users. So for example, people who do not require CORBA functionality, do not download that module.

How would this work together? First of all, there will be a Java Kernel: its task will be to reduce the time for installation and launch when the user needs to install the JRE, in order to run an application. Then followed by Quick starter, which will manage the execution of a Java application to launch. Part of the solution to the speed, will rely on optimising operating system to preload the JVM process, as pages from the disk.

I mentioned earlier that we also had several Brits at JavaONE this year. For the AJAX Web 2.0 track, Joe Walker, a fellow Java Champion, gave a presentation on his own creation Direct Web Remoting. He talked about DWR 2.0 and the concept of reverse-AJAX. Stephen Colebourne, who is known for his work on improving Java's data and times API, presented a late night BOF on JSR 310. Stephen is also a fellow Java Champion. Representing London, I myself presented a piece called 'Re-architecting

PETER PILGRIM

Peter is a Java EE software developer, architect, and Sun Java Champion from London. By days he works as an independent contractor in the investment banking sector. Peter can be contacted at peter.pilgrim@gmail.com

One Laptop per Child

Silas Brown talks about an interesting project.

In case you haven't heard of it already, the One Laptop Per Child project, whose website is www.laptop.org, aims to design a laptop with a production cost of around $100 (about 50 pounds) that can be given to children in the 'third world'. The laptops are meant to be rugged and will run on very little power; they have new-technology displays that work well in all lighting conditions, can be used in both with-keyboard and without-keyboard (e-book) positions, have 256M RAM and 1G of internal Flash memory (with USB ports that can take more), and run a version of Linux on a 400MHz+ x86-compatible low-power CPU. The machines also have WiFi (and will automatically create an ad-hoc network around the classroom if no access point is available) and support collaborative or group work.

I can't help wondering if anyone in ACCU would be interested in helping out with open courseware to help the brighter children to learn programming

The project is always looking for more volunteers to help develop its software, which is all free as in speech/GPL. The distribution can be downloaded and run in a virtual machine under QEMU or similar; if you have good enough hardware you can try it. I have it on a real prototype machine (called an XO because of the child-with-cross-legs logo); the prototype machines are more limited in supply but the organisers kindly sent me one so that I could do more realistic tests of its suitability for people with low vision (it's difficult to do that on an emulator because you can't be sure you've recreated the exact physical characteristics of the device). They won't be able to supply test hardware to everyone, but I'd like to encourage more ACCU members to try the software download if they'd like a good exercise or spare-time project that can also help a lot of disadvantaged people. Additionally, I can't help wondering if anyone in ACCU would be interested in helping out with open courseware to help the brighter children to learn programming.

The final choice of what courseware will be included with the XO will be up to the ministries of education of the countries that accept it, but it's interesting to consider the possibilities. The desktop is built on Python and the developers are keen to make the Python source available for inspection; there is talk of implementing a 'view source' option for every application. The distribution includes a Gecko-based browser, and a viewer for PDF and other formats. Currently gcc is not included by default although it can be downloaded as a package. The distribution is based on Fedora; I believe this choice of distribution was at least partly motivated by the fact that Red Hat are a major contributor to the specially-designed desktop front-end, which is called Sugar and is based on Python and GTK with the Matchbox window manager.

The best way to become familiar with the workings of the system (or at least the parts of it relevant to your interests) is to dive into a working system (on an emulator or whatever), press Alt-0 to get the developer console, look at the output of `ps auxwww` to see what processes are running with what parameters, and from there follow the scripts through the filesystem, trying things out in a Python shell where necessary (e.g. if it says `get_path_to_something()` try calling it yourself and see what it gives). It helps to look around the dev.laptop.org website also. Bugs are tracked with trac, and there is a wiki and mailing lists.

Definitely worth a look. ■

SILAS BROWN

Silas is partially sighted and is currently undertaking freelance work assisting the tuition of computer science at Cambridge University, where he enjoys the diverse international community and its cultural activities. Silas can be contacted at ssb22@cam.ac.uk

On The Powell Cable Car (continued)

legacy applications with Spring Framework and Hibernate', which was well received.

At the conference Neil Gafter, Google employee, and one of the original architects of generics for Java SE 5, talked about his proposal to add Closures to the Java language (Java SE 7). It was a packed enthralling lecture into the pros and cons of this language change. Closures will definitely be useful for both library writers and users. It brought out the semantic complexity of a few issues: the exact meaning of lexically scoped variables; the treatment of exceptions, which are raised in a closure; and abstract-code constructs. Incidentally Stephen Colebourne has also written an alternative Closure proposal.

As leader of a London Java User Group, it was pretty fascinating to put human faces to electronic addresses. I dined and drank with other JUG leaders from all over the world, and we shared great experiences. We talked about getting good quality speakers to events, hosting events and getting sponsorship from companies. In conclusion, I thought the user groups needs to collaborate more. The JUG leaders should all find out about who and what else is out there in our development community. By understanding technology and the affect of it on humankind, we are building a digital society, not just a community. We can, as software developers, really affect the lives and the spirits of others around us and we will.

So Java is back on the desktop with JavaFX leading the way. I have no doubt that Sun are extremely serious about solving the JRE issues, because the market is ready for it yesterday! I personally cannot wait for the Consumer JRE. I think that it will be crucially important for all departments who are thinking about deploying future Java Web 2.0 applications. Java is already great for the enterprise and the server-side, we know that. Java EE has been a world-wide hit for a lot of businesses. Modularisation of Java SE and the usability of the Java plug-in will bring major benefits, because Java already has a decent 2D and 3D APIs. We just need tools like JavaFX and much better media support for popular CODECS to get there. Did you know that Java is not the only language that runs on the JVM? Including Java and now JavaFX, there are at least 200 other programming languages such as JRuby and Groovy that also run on it. This year's conference recharged my faith that Java will become great on the front-end.

That is all for now, until the next time. Roll on 2008! ■

Visiting Files and Directories in C#

Paul Grenyer walks the file system.

Recently I have been doing a lot of work with continuous integration [CI] using CruiseControl.Net [CCNet]. It is currently an adequate product, but has the potential to be an excellent product. I'm looking forward to the next release, which is likely to be the first for .Net 2.

One particular requirement I had was to remove the source tree from the disk after a change has been identified and before the change is checked out. This effectively gives the user a completely clean check out each time. This feature is not supported by any of the current CruiseControl.Net source control tasks. It is possible to use a pre-build task to make a system call to remove the source, but this is a hack and not portable easily across platforms.

In my article 'Continuous Integration with CruiseControl.Net – Part 3' (available soon from http://www.marauder-consulting.co.uk), I wrote about the powerful plug-in mechanism supported by CruiseControl.Net. It can be used to modify existing task blocks to add, for example, the ability to remove a source tree prior to checkout. C# is the ideal language to use to write CruiseControl.Net plug-ins as they are simply .Net assemblies.

In this article I am going to look at how to use C# to remove a source tree and develop the code into a enumeration method [EnumMethod] and visitor [Visitor] compound that can be used for general purpose file and directory traversal.

Directory.Delete

The .Net directory class has a **`Delete`** static method that takes a path to the directory to be deleted. At first glance it appears to be the answer to the problem and could make this article very short indeed (in reality all it did was make my estimate of the time it would to write the plug-in wildly inaccurate). To test it I created a temporary folder and put an empty text file in it, then wrote the following code:

```
Directory.Delete(@"C:\temp\somedir");
```

I was surprised to find that an exception was thrown. When I added exception handling code the message was as follows:

```
The directory is not empty.
```

This seemed fair enough and the **`Directory.Delete`** method does have an overload that takes a second parameter called **`recursive`**, which claims to remove all files and directories in the path recursively. I modified the code and tried again:

```
Directory.Delete(@"C:\temp\somedir",true);
```

Success! The file and the directory were both deleted. For good measure (although now it just feels like proving black was white and getting killed on the next zebra crossing [HHGTTG]), I tried the same code on a source tree. It failed with the error message:

```
Access to the path 'all-wcprops' is denied.
```

Some files had been deleted, but others had not. So I was left wondering what was special about the files that hadn't been deleted.

I have only worked with a few source control systems, namely CVS [CVS], SubVersion [SVN] and Perforce [P4]. Perforce and at least one configuration of CVS that I have used checks all its files out as read-only. Files in the special `.SVN` folders used by SVN to keep track of changes in the source are also read-only. `all-wcprops` is one of these files. I started wondering if the problem could be caused by read-only files, so I recreated my directory an empty text file, this time making it read-only. I ran the code again and got the same error, this time relating to the file in my directory.

This suggested that the solution was to remove the read-only flag from files before deleting them. The **`FileAttributes`** enum and **`File`** class can be used to test and set the attributes of a file, including read-only:

```
string file = @"C:\temp\somedir\somefile.txt";
FileAttributes fileAtts = File.GetAttributes(file);
File.SetAttributes(file,
   fileAtts & ~FileAttributes.ReadOnly);
Directory.Delete(@"C:\temp\somedir",true);
```

Executing the above code removed the file and the directory. However, I could not test it on the source tree as that would require a list of all the files in the tree that were read-only.

Listing read-only files in a directory

Listing the files in a directory in .Net is simple. All you need is a **`DirectoryInfo`** object for the directory and then you can iterate through a collection returned by the **`GetFiles`** method. To test this I added some more files to the test directory, made three of them read-only and then ran the code in Listing 1.

As expected the output listed the read-only files. The next step was to remove the read-only attribute from the files. It also makes sense to remove the files at the same time as that is the ultimate goal. The **`File.Delete`** static method is used for this (Listing 2).

Listing 1

```
DirectoryInfo dirInfo = new
DirectoryInfo(@"C:\temp\somedir");

foreach (FileInfo fileInfo in dirInfo.GetFiles())
{
  FileAttributes fileAtts =
     File.GetAttributes(fileInfo.FullName);
  if ((fileAtts & FileAttributes.ReadOnly) != 0)
  {
    Console.WriteLine(fileInfo.FullName);
  }
}
```

Listing 2

```
DirectoryInfo dirInfo = new
DirectoryInfo(@"C:\temp\somedir");

foreach (FileInfo fileInfo in dirInfo.GetFiles())
{
  FileAttributes fileAtts =
     File.GetAttributes(fileInfo.FullName);
  if ((fileAtts & FileAttributes.ReadOnly) != 0)
  {
    File.SetAttributes(fileInfo.FullName,
       fileAtts & ~FileAttributes.ReadOnly);
  }
  File.Delete(fileInfo.FullName);
}
```

PAUL GRENYER

Paul has been a member of the ACCU since 2000 and founded the ACCU Mentored Developers. Paul now contracts at an investment bank in Canary Wharf. He can be contacted at paul.grenyer@gmail.com

Listing 3
```
DirectoryInfo dirInfo =
   new DirectoryInfo(@"C:\temp\somedir");
foreach (DirectoryInfo subDirInfo in
   dirInfo.GetDirectories())
{
  Console.WriteLine(subDirInfo.FullName);
}
```

This is still not a complete solution as a source tree will also contain subdirectories that have some read-only files.

Listing subdirectories

A directory's subdirectories can be listed in much the same way as its files, but instead of using the `GetFiles` method the `GetDirectories` method is used (Listing 3).

I tested this in the obvious way, by creating some subdirectories in my test directory. The above code listed them all.

Putting it all together – recursion

I now had a method of:

- Listing all files in a directory
- Detecting if a file is read-only
- Removing the read-only attribute from a file
- Deleting a file
- Listing all subdirectories in a directory

The final solution needs to:

- Visit every directory in the source tree
- Remove the directory's files and subdirectories
- Remove the directory itself.

The easiest way to do this is using recursion [Recursion]. In Listing 4, `DeleteDirectory` is called recursively for every directory in the supplied path until a directory is reached that doesn't have any subdirectories or all subdirectories have been visited. Then it works back up the tree of directories removing the read-only attribute from read-only files and deleting all files.

Listing 4
```
private static void DeleteDirectory(string path)
{
  DirectoryInfo dirInfo = new DirectoryInfo(path);

  foreach (DirectoryInfo subDirInfo in
     dirInfo.GetDirectories())
  {
    DeleteDirectory(subDirInfo.FullName);
  }

  foreach (FileInfo fileInfo in
     dirInfo.GetFiles())
  {
    FileAttributes fileAtts =
       File.GetAttributes(fileInfo.FullName);
    if ((fileAtts & FileAttributes.ReadOnly) != 0)
    {
      File.SetAttributes(fileInfo.FullName,
         fileAtts & ~FileAttributes.ReadOnly);
    }

    File.Delete(fileInfo.FullName);
  }
 Directory.Delete(path);
}
```

This of course is the solution to the problem. I ran it on the source tree and it deleted all files and directories. So this article is finished, right?

The patterns

Wrong! The `DeleteDirectory` method is a very specific solution to a very specific problem. It can not be used for doing anything other than deleting directories and their contents. However, wanting to *visit* and operate on every file and directory in a tree is a common requirement. Imagine, for example, you were writing an application similar to Windows Explorer or any other application that requires a list of files and directories.

The functionality provided by `DeleteDirectory` can be separated out and expressed as two patterns. Iterating through directories and files is an implementation of the ENUMERATION METHOD pattern and operating on each file and directory is a (well hidden at the moment) implementation of the VISITOR pattern.

DirectoryTraverser

Let's start by looking at the class that will traverse through the directories and files. `DeleteDirectory` works very well as simple static method, but with `DirectoryTraverser` we want to be able to plug-in different functionality and the power that a class provides makes life very much easier, especially in terms of interface.

The `DirectoryTraverser` class only needs a single, non-constructor, public method in its interface. This method is used to specify which directory to traverse and to start the traversal:

```
class DirectoryTraverser
{
  public void Traverse(string path)
  {
    …
  }
}
```

`DirectoryInfo` and `FileInfo` objects are used when the actual file and directory traversal is performed. When the `DeleteDirectory` method is called recursively the full path to the next directory is extracted from the `DirectoryInfo` object. This is an unnecessary property access. This can be avoided by providing a private overload of `Traverse` that takes a `DirectoryInfo` object and use the original method to create the `DirectoryInfo` object to be passed the first time the overload is called (Listing 5).

If we were writing an application that sends an indented list of directories and files to the console, we want to know about a directory before we know about its files. If we're writing an application that deletes directories and files we want to know about each file before the directory so that we can delete the files and then delete the directory. Order is very important. Therefore we need two directory related actions into which we can plug functionality. One invoked when entering a directory. The other invoked when leaving a directory. Only a single pluggable action is needed for files (Listing 6).

That completes *most* of the functionality of `DirectoryTraverser`. However, at the moment it is not very pluggable.

Listing 5
```
class DirectoryTraverser
{
  public void Traverse(string path)
  {
    Traverse(new DirectoryInfo(path));
  }

  private void Traverse(DirectoryInfo dirInfo)
  {
    …
  }
}
```

Listing 6

```
private void Traverse(DirectoryInfo dirInfo)
{
  EnterDirectory(dirInfo);

  foreach (DirectoryInfo subDir in
     dirInfo.GetDirectories())
  {
    Traverse(subDir);
  }

  foreach (FileInfo file in dirInfo.GetFiles())
  {
    VisitFile(file);
  }

  LeaveDirectory(dirInfo);
}

private void EnterDirectory(DirectoryInfo dirInfo)
{
}

private void VisitFile(FileInfo fileInfo)
{
}

private void LeaveDirectory(DirectoryInfo dirInfo)
{
}
```

IDirectoryVisitor

The visitor part of the implementation is the ability to plug-in behaviour when a directory is entered, when a file is visited and when a directory is left. The easiest way to do this is via a method call for each operation and C#'s delegate mechanism would seem to be the obvious implementation solution. In fact my early implementations used it. It is a good solution if you only have one method to delegate. As soon as you have more it becomes cumbersome. Do you set the delegates via the constructor or properties or both? What if you only want a delegate for entering directories and visiting files? Do you have a constructor that takes only two delegates or only one? If so which one? Which two? etc.

The simplest solution in this case is to have a callback interface, such as the one below:

```
interface IDirectoryVisitor
{
  void EnterDirectory(DirectoryInfo dirInfo);
  void VisitFile(FileInfo fileInfo);
  void LeaveDirectory(DirectoryInfo dirInfo);
}
```

Listing 7

```
class DirectoryWriter : IDirectoryVisitor
{
  public void EnterDirectory(
     DirectoryInfo dirInfo)
  {
    Console.WriteLine(dirInfo.FullName);
  }

  public void VisitFile(FileInfo fileInfo)
  {
    Console.WriteLine(fileInfo.FullName);
  }

  public void LeaveDirectory(
     DirectoryInfo dirInfo)
  {
  }
}
```

Then you can just implement the methods you want and leave the others as empty methods that do nothing. For example the following implementation of the interface lists all directories and files in the given path. (Listing 7)

Now all that is needed is a method of plugging an implementation of the `IDirectoryVisitor` into the `DirectoryTraverser`. This is easy and straightforward. Simply pass an interface reference into the constructor, store and call the appropriate callback, instead of the methods show previous (Listing 8).

The class in Listing 8 does, of course, have a, strictly speaking, extra level of unnecessary indirection: the calls to the `EnterDirectory`, `VisitFile` and `LeaveDirectory` methods could be replaced with calls directly to the visitor.

`DirectoryTraverser` and `DirectoryWriter` are used together as follows:

```
DirectoryTraverser dirTrav =
   new DirectoryTraverser( new DirectoryWriter());
dirTrav.Traverse(@"C:\temp");
```

Coming full circle

The `DirectoryWriter` visitor in Listing 8 is very simple, not especially useful and does not demonstrate how the `EnterDirectory` and `LeaveDirectory` callbacks can be used together. The more advanced version shown in Listing 9 uses the methods to control the indentation when sending the directory and file structure to the console.

Running it against the test project I created for this article gives the output in Listing 10.

An example of a visitor that deletes files and directories is shown in Listing 11 to bring this article full circle. ■

Listing 8

```
class DirectoryTraverser
{
  private IDirectoryVisitor visitor;

  public DirectoryTraverser(
     IDirectoryVisitor visitor)
  {
    this.visitor = visitor;
  }

  public void Traverse(string path)
  {
    Traverse(new DirectoryInfo(path));
  }

  private void Traverse(DirectoryInfo dirInfo)
  {
    visitor.EnterDirectory(dirInfo);

    foreach (DirectoryInfo subDir in
       dirInfo.GetDirectories())
    {
      Traverse(subDir);
    }

    foreach (FileInfo file in dirInfo.GetFiles())
    {
      visitor.VisitFile(fileInfo);
    }

    visitor.LeaveDirectory(dirInfo);
  }
}
```

Listing 9

```
class DirectoryWriter : IDirectoryVisitor
{
  private int indent = 0;

  public void EnterDirectory(
     DirectoryInfo dirInfo)
  {
    StringBuilder buffer = new StringBuilder();
    buffer.Append( new string( ' ', indent ) );
    buffer.Append(dirInfo.Name);
    Console.WriteLine(buffer.ToString());
    ++indent;
  }

  public void VisitFile(FileInfo fileInfo)
  {
    StringBuilder buffer = new StringBuilder();
    buffer.Append(new string(' ', indent+1));
    buffer.Append(fileInfo.Name);
    Console.WriteLine(buffer.ToString());
  }

  public void LeaveDirectory(
     DirectoryInfo dirInfo)
  {
    --indent;
  }
```

Listing 10

```
DirectoryTraverser2
 DirectoryTraverser2
  bin
   Debug
     DirectoryTraverser2.exe
     DirectoryTraverser2.pdb
     DirectoryTraverser2.vshost.exe
   Release
     DirectoryTraverser2.exe
     DirectoryTraverser2.pdb
  obj
   Debug
    TempPE
     DirectoryTraverser2.exe
     DirectoryTraverser2.pdb
   Release
    TempPE
     DirectoryTraverser2.exe
     DirectoryTraverser2.pdb
    DirectoryTraverser2.csproj.FileList.txt
  Properties
    AssemblyInfo.cs
   DirectoryTraverser.cs
   DirectoryTraverser2.csproj
   DirectoryWriter.cs
   EntryPoint.cs
   IDirectoryVisitor.cs
  DirectoryTraverser2.sln
```

Acknowledgments

Thank you to Kevlin Henney for reviews and guidance.

References

[CI] http://en.wikipedia.org/wiki/Continuous_integration
[CCNet] http://ccnet.thoughtworks.com/
[EnumMethod] http://www.two-sdg.demon.co.uk/curbralan/papers/ATaleOfThreePatterns.pdf
[Visitor] *Design patterns: elements of reusable object-oriented software* by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. ISBN-10: 0201633612 ISBN-13: 978-0201633610
[HHGTTG] http://www.bbc.co.uk/cult/hitchhikers/
[CVS] http://www.nongnu.org/cvs/
[SVN] http://subversion.tigris.org/
[P4] http://www.perforce.com/
[Recursion] http://en.wikipedia.org/wiki/Recursion
[GoF] *Design patterns : elements of reusable object-oriented software* by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. ISBN-10: 0201633612 ISBN-13: 978-0201633610

Listing 11

```
class DeleteDirectoryVisitor : IDirectoryVisitor
{
  public void EnterDirectory(
     DirectoryInfo dirInfo)
  {
  }

  public void VisitFile(FileInfo fileInfo)
  {
    FileAttributes fileAtts =
       File.GetAttributes(fileInfo.FullName);
    if ((fileAtts & FileAttributes.ReadOnly) != 0)
    {
      File.SetAttributes(fileInfo.FullName,
         fileAtts & ~FileAttributes.ReadOnly);
    }

    File.Delete(fileInfo.FullName);
  }

  public void LeaveDirectory(
     DirectoryInfo dirInfo)
  {
    Directory.Delete(dirInfo.FullName);
  }
}
```

Regional Meetings

A round-up of the latest ACCU regional events.

ACCU North East

Report from Ian Bruntlett (ianbruntlett@hotmail.com)

Surroundings: We were in Contacts building again. Yesterday was the official opening of Contacts new premises and there was some food left over from Friday's buffet, so we had free lunch including some really nice cake. Contact was running a flag day and there were a number of people rushing about, selling Contact stickers.

Presentation/Discussion: Snakes and ladders, AKA Python traps and pitfalls. I (Ian Bruntlett) gave a presentation about Python. I covered how far I'd got through the standard tutorials, highlighting the kinds of things a C or C++ programmer would note as unusual things of interest.

We discussed the language and Alex experimented with some example programs on his laptop, trying things out and generally working out what can and can't be done.

We covered list comprehensions and generator expressions and their similarities: list comprehensions returning a list of results in memory, generator expressions returning values one at a time.

We got a bit bogged down re: passing values to functions, namely:

1. Normal / positional parameters
2. Arbitrary argument lists
3. Keyword arguments

After the meeting, I dug out some examples and sent them to the ACCU NE mailing list for people to experiment with and hopefully understand some of the fancy things Python can do.

Unpacking argument lists

I deliberately avoided covering unpacking argument lists where the arguments for a function are already in a list or tuple but need to be unpacked for a function call. This can be done automatically. Here is an example:

```
print  range(3,6)  # produces [3,4,5]
args=[3,6]
print range(*args) # also produces [3,4,5]
```

In the same way, dictionaries can deliver keyword arguments with the `**` operator:

```
d = { "voltage" : "four million",
      "state" : "bleedin demised",
      "action" : "VOOM" }
parrot (**d)
```

Programming was discussed and Alex took the opportunity to make people aware of high level languages LISP, Erlang etc and he quoted Philip Greenspun's 'Tenth Rule of Programming', that any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.

Next month: Alex Kavanagh has agreed to present Erlang, a massively threaded language.

ACCU Cambridge

Report from Ric Parkin (ric.parkin@gmail.com)

Since the last CVu, the local ACCU Cambridge group has held two meetings, again kindly hosted by DisplayLink.

In August Ric Parkin gave an updated version of his talk 'Semantic Programming' given at this year's conference, looking at how to take advantage of strongly typed languages to encode usage rules into the program types, and what consequences this has on a program's style and idioms.

In September Peter Smith gave a talk 'The Missing Link' looking at the much overlooked Linker, giving a high level overview of their role and delving into some specific examples to illustrate some of their more quirky corners, and how they've come to influence languages such as C and C++ with their concepts of 'Linkage', 'One Definition Rule', and name conventions.

Despite the holiday season turnout has been good, and involved plenty of audience discussion, and even more in the pub afterwards.

Future meeting are already pencilled in for 4th October, with Russel Winder looking at Dynamic Languages, and on 1st November, with Allan Kelly on Agile development. Hope to see you there!

ACCU London

Report from Steve Love (steve@essennell.co.uk)

The September ACCU London meeting was special for me because it was my first time attending. The talk was by Pete Goodliffe, author of *Code Craft* (if you haven't got it, go get it!) and a hefty history of 'Professionalism in Programming' articles, who is well known for monkeys and bare feet. If you've seen Pete present at the ACCU conference, you'll know that he is a treat to watch. The talk he gave was 'Code Monkeys', a slightly satirical, and completely irreverent look at the inhabitants of what Pete calls 'The Software Factory'.

A Goodliffe presentation wouldn't be complete without some fun and games, as if watching Pete himself dancing and gesticulating wasn't enough. To give himself a chance for a short breather, Pete had us designing a 'simple' system from scratch requirements. The purpose of the exercise became clear when the meat of the talk began and Pete started talking monkey-business. Being an excellent programmer isn't really about writing amazing code, it's just as much about attitude – both to your work and to your colleagues.

It was an entertaining and enlightening talk, delivered with the expected gusto and enthusiasm, despite there being only 10 of us there.

If you weren't there, you missed a treat and it's nobody's fault but yours! ☺

7 City Learning (http://7city.com) again hosted the evening and after the talk we retired to a nearby hostelry to debate our true 'monkey' identities.

Code Critique Competition 48

Set and collated by Roger Orr.

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

Last Issue's Code

> I have this small program here and am wondering why, when compiled with full optimisation it does not produce any output at all? It works fine without optimisation.

Please try to help the programmer find the answer to this question (and future ones).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int tab[13][20]={
   {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   {0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0},
   {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0},
   {0,0,0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,1,0,0},
   {0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1},
   {1,0,0,1,0,0,0,1,0,1,0,0,0,1,1,0,0,1,0,0},
   {0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,1,0,0,1},
   {0,0,0,0,1,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0},
   {1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,2,0},
   {0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,0,0,1},
   {0,0,0,0,1,1,0,1,1,0,0,0,0,0,1,0,1,0,0,0},
   {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0}};

int tabx[13][20];
int sumy[20];
int licz(char *konfig)
{
  int i,n,l,el,ile=0;
  char c;

  memset(&sumy,0,sizeof(sumy));
  memset(&tabx,0,sizeof(tabx));

  for (i=0; i<=12; i++)
   for (n=0; n<=19; n++) {
    c=konfig[i];
    l=atoi(&c);
    el=l*tab[i][n];
    if (el>0) sumy[n]=sumy[n]+el;
    tabx[i][n]=el; }
  for (n=0; n<=19; n++)
     if (sumy[n]==3) ile++;
  return(ile);
}
```


ROGER ORR

Roger has been programming for over 20 years, most recently in C++ and Java for various investment banks in Canary Wharf and the City. He joined ACCU in 1999 and the BSI C++ panel in 2002.
He may be contacted at rogero@howzatt.demon.co.uk


```
int ile1(char *s)
{
  int i,n=0;
  for (i=0;i<=12;i++) if (s[i]=='1') n++;
  return(n);
}
void main()
{
  char komb[13];
  int a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad;
  int xx,w,ile;
for (xx=0;xx<=4;xx++)
  for (a1=0; a1<=1; a1++)
   for (a2=0; a2<=1; a2++)
    for (a3=0; a3<=1; a3++)
     for (a4=0; a4<=1; a4++)
      for (a5=0; a5<=1; a5++)
       for (a6=0; a6<=1; a6++)
        for (a7=0; a7<=1; a7++)
         for (a8=0; a8<=1; a8++)
          for (a9=0; a9<=1; a9++)
           for (aa=0; aa<=1; aa++)
            for (ab=0; ab<=1; ab++)
             for (ac=0; ac<=1; ac++)
              for (ad=0; ad<=1; ad++)
               {
sprintf(komb,"%d%d%d%d%d%d%d%d%d%d%d%d",
a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad);
                 w=licz(komb);
                 ile=ile1(komb);
                 if ((w==1) && (ile<=3))
                 {
                  printf("%s\t%d\t%d\n",
                  komb,w,ile);
                 }
               }
}
```

Critiques

From Nevin :-] Liber <nevin@eviloverlord.com>

The main problem with this code is in its use of C strings. C strings must always be `'\0'`-terminated, and there must be room in the underlying array to hold that null byte.

There is also the minor issue that `void main()` is not a valid declaration; it must return `int`.

Making minimal changes to the code:

```
/* ... */
int licz(char *konfig)
{
  int i,n,l,el,ile=0;
  char c[2] = {};
/* was: char c;          */

  memset(&sumy,0,sizeof(sumy));
  memset(&tabx,0,sizeof(tabx));
  for (i=0; i<=12; i++)
   for (n=0; n<=19; n++) {
```

```
    *c=konfig[i];
/* was: c=konfig[i];    */
    l=atoi(c);
/* was: l=atoi(&c);     */
    /* ... */
}

/* ... */

int main()
/* was: void main();    */
{
  char komb[14];
/* was: char komb[13];  */
  /* ... */
}
```

This is a decent solution. However, for purposes of this critique, let's explore another one. There are advantages and drawbacks to using a C string as the underlying data representation.

Pluses:

1. It is extensible. In order to add more bits, just increase the size of the corresponding arrays.
2. It is displayable. All it takes is a simple **`printf`**.

Minuses:

1. It is fragile. Because it is not a self-contained type, everywhere it is used one has to make sure that the array is big enough and that it is `'\0'`-terminated.

Looking at the code, ultimately it is doing calculations based on bits in **`komb`**. If we are willing to limit the maximum number of bits to those in a fundamental type (**`short`**, **`long`**, etc.), we can use a fundamental type to hold the bits, eliminating the fragileness of the original solution.

Given that we only have 13 bits, on most platforms an **`unsigned short`** will do quite nicely to hold the data (I prefer using unsigned types when doing bit manipulation). We'll need some helper functions to extract bits and to print. Here is the start of the code:

```
static const size_t K = 13;
static const size_t T = 20;
typedef unsigned short k_type;

int bit(k_type k, size_t b)
{ return !!((1U << (K - 1 - b)) & k); }

void print(k_type k, int w, int ile)
{
  size_t i;
  for (i = 0; K != i; ++i)
    printf("%c", "01"[bit(k, i)]);
  printf("\t%d\t%d\n", w, ile);
}
```

Normally I'd put in better names but I don't know this problem domain.

`bit(k_type k, size_t b)` is a new function that returns the value (0 or 1) of the bth bit of object **`k`**. Note: bit 0 is stored as the most significant bit in **`k_type`** so that the bit positions match those in the original solution.

```
int licz(k_type konfig)
{
  int sumy[T] = {};
  static const unsigned char tab[K][T] =
  {
    {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0},
    {0,0,0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,1,0,0},
    {0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1},
    {1,0,0,1,0,0,0,1,0,1,0,0,0,1,1,0,0,1,0,0},
    {0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,1,0,0,1},
    {0,0,0,0,1,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0},
    {1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,2,0},
    {0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,0,0,1},
    {0,0,0,0,1,1,0,1,1,0,0,0,0,0,1,0,1,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0},
  };

  int ile = 0;
  size_t i;
  size_t n;
  for (i = 0; K != i; ++i)
    if (bit(konfig, i))
      for (n = 0; T != n; ++n)
        sumy[n] += tab[i][n];

  for (n = 0; T != n; ++n)
    ile += (sumy[n] == 3);

  return ile;
}
```

`sumy` and **`tab`** were only used inside this function, so I made them local. **`tabx`** was set but never read, so I eliminated it entirely. To clear **`sumy`**, I prefer initialization syntax over **`memset`**. **`tab`** is now a static **`const unsigned char`** array as it never needs to be modified and an **`unsigned char`** is perfectly large enough to hold the values 0, 1 and 2. I also added an optimization in the first **`for`** loop: if the bit is 0, then the corresponding contribution to **`sumy`** is also 0, so there is no need to calculate it.

```
int ile1(k_type s)
{
  int n = 0;
  size_t i;
  for (i = 0; K != i; ++i)
    n += bit(s, i);
  return n;
}
```

`ile1(k_type s)` just counts the number of 1 bits in **`s`**.

Finally, here is **`main()`**, using **`k_type`** instead of C strings:

```
int main()
{
  size_t xx;
  for (xx = 0; 4 != xx; ++xx)
  {
    k_type komb;
    for (komb = 0; 1U << K != komb; ++komb)
    {
      int w = licz(komb);
      int ile = ile1(komb);
      if (w == 1 && ile <= 3)
        print(komb, w, ile);
    }
  }
}
```

So, which solution would I go with? If I knew that I'd never need more than the number of bits that can fit in a fundamental type, I'd go with the second solution. If it needed to be extensible to an arbitrary number of bits (and written in C), I'd stick with the author's version (fixed, of course).

From Paul Evans <paul.evans@reuters.com>

First let's fix the bug.

```
  int licz(char *konfig)
  {
```

```
  int i,n,l,el,ile=0;
```

this next line is the start of the problem – see below

```
  char c;

  memset(&sumy,0,sizeof(sumy));
  memset(&tabx,0,sizeof(tabx));

  for (i=0; i<=12; i++)
    for (n=0; n<=19; n++) {
      c=konfig[i];
```

this next line is only works because a zero happens to be in memory after the variable **c**, when optimisation is cranked up, this unplanned side-affect of the code layout disappears and invalidates the translation of the numeric character stored in **c** to the corresponding integer in **l**:

```
  l=atoi(&c);
```

by changing **c** in function **licz** from a **char** to a null-terminated **char*** and then passing the pointer to this string to **atoi**, int **l** always gets the correct value, regardless of optimisation levels.

Top Tip: don't program in C, program in C++!!!!

[Ed: Paul then successively refactored the code – for reasons of space I won't show the intermediate versions, just give his commentary and output of the last iteration]

In the first refactoring of the code we get rid of the obviously unnecessary code + vars + do a little reordering of a nested loop: and behold! – the original problem vanishes. But, it can still be vastly improved...

Next a lot of those **for** loops are going to have to go, but let's do an incremental refactoring and change all of those unnecessary post-increments to pre-increments. They are very annoying because they involve an unused temporary variable.

The final refactor

1. get rid of that messy **char*** string of ascii 1s + 0s to represent a bit pattern. Use the actual bit pattern of an **int** + change all code affected.
2. get rid of all magic numbers and use **#defines** instead (this is C not C++)
3. try to find more meaningful names for things
4. in the spirit of space optimisation, use **chars**, **shorts**, etc., where appropriate (assumes BITS to be <16)
5. put every **for**/**if** target into a **{}** block – easier to add/delete/comment-out statements
6. consistent/tidy indenting, spacing + make **main()** standard by returning an **int**
7. try to comment some things, but need context to do it properly.

It is still pretty scary, but not knowing the context for this weird beast, it's much more clearer than before.

```
#include <stdio.h>

#define BITS 13
#define COLUMNS 20
#define SUMY_TARGET 3
#define W_TARGET 1
#define MAX_BITS 3
#define REPS 5

char tab[BITS][COLUMNS]={
  {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0},
  {0,0,0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,1,0,0},
  {0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1},
  {1,0,0,1,0,0,0,1,0,1,0,0,0,1,1,0,0,1,0,0},
  {0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,1,0,0,1},
  {0,0,0,0,1,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0},
  {1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,2,0},
                          // notice the 2!!
  {0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,0,0,1},
  {0,0,0,0,1,1,0,1,1,0,0,0,0,0,1,0,1,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0}};

char calc_w(int bit_pattern)
{
  char bit_pos=0, n=0, w=0, sumy=0;
  for (n=0; n < COLUMNS; ++n) {
  for (bit_pos=0; bit_pos < BITS; ++bit_pos) {
    if (bit_pattern & (1<<(BITS-1-bit_pos))) {
      // the bit in bit_pattern bit_pos
      // positions from the left is set to 1
      // so do a weird sum
      sumy += tab[bit_pos][n];
      }
    }
    if (sumy==SUMY_TARGET) {
      ++w;
    }
    sumy = 0;
  }
  return(w);
}

char bit_count(int bit_pattern)
{
  char bit_pos=0, bc=0;
  for (bit_pos=0; bit_pos < BITS; ++bit_pos) {
    if (bit_pattern & (1<<bit_pos) ) {
      // the bit in bit_pattern,
      // bit_pos positions from the right
      // is set to 1 -- so bump the bit count
      ++bc;
    }
  }
  return(bc);
}

int main()
{
  char bit_pos=0, reps=0, no_bits=0, w=0;
  short bit_pattern=0;
  for (reps=0; reps < REPS; ++reps) {
    for(bit_pattern=0; bit_pattern <
          (1<<BITS); ++bit_pattern) {
      // generate all the bit patterns from
      // 000....0
      // to
      // 111....1
      // that are BITS bits wide with
      // the lesser significant bits looping
      // more rapidly than the greater
      // significant bits
      w=calc_w(bit_pattern);
      no_bits=bit_count(bit_pattern);
      if ((w==W_TARGET)&&(no_bits<=MAX_BITS)){
        for(bit_pos=BITS-1; bit_pos >= 0;
            --bit_pos) {
          // print out a '1' or '0'
          // corresponding to the
          // bit in bit_pattern, bit_pos
          // positions from the left
        printf("%c",
          ( bit_pattern & (1<<bit_pos) ) ?
            '1':'0');
        }
      printf("\t%d\t%d\n",w,no_bits);
      }
    }
```

```
  }
  return 0;
}
```

Commentary

It is relatively easy to find the answer to the question the writer of the code asked. However, few of us would want to leave it there because the code is crying out for improving in the ways Paul and Nevin described.

The original code is hard to critique because it isn't clear what it does. For example, calling the big table of numbers **tab** gives no idea about what it is a table of. One question neither entrant asked is whether the single **2** in the table is actually correct – should it in fact be a **1**? (I don't know either!)

Paul's first comment in **main** in his final solution (generate all the bit patterns from 000....0 to 111....1 that are BITS bits wide) is an attempt to fill in some of the missing meta-data in this program, but without communication with the actual writer of the code there's a lot that remains unclear.

Does it matter?

Well, yes it does. For example, both solutions make attempts to reduce the number of bits used to store the data being worked on, but without some context for the problem it is hard to tell whether this is cost effective.

My favourite computing guideline is separate the things that change from the things that stay the same. This program would be much easier to work on if it was clearer which were which.

However, even where we don't have a full understanding of the context in which the code is to be used it is still possible to make localised improvements to it. In this case, the existence of an output file makes it very easy to verify that the new output matches the old, which gives confidence that the refactorings are sound.

Note how hard it is to get this right – the refactoring of:

```
for (xx=0;xx<=4;xx++)
```

to

```
for (xx = 0; 4 != xx; ++xx)
```

looks right at first, but isn't. I too have introduced similar bugs when refactoring non-idiomatic code.

The Winner of CC 47

Both the entrants answered the original question and also provided additional changes to try and make the code better.

Both solutions changed the original character string for **komb** into a numeric value; Nevin's explanation of the pros and cons of doing this was good. I liked the places where Paul provided some rationale for his changes as this should help the original programmer understand *why* things were changed.

After re-reading both entries I've awarded this issue's prize to Paul.

Code Critique 48

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

> I'm reading in dates and printing them out sorted in 'human friendly' format. Sometimes the program crashes but I can't see a clear pattern – can you suggest what might be wrong and how the program could be improved?

The code is in Listing 1.

(Thanks are due to Hubert Matthews for supplying the original idea for this issue's critique.)

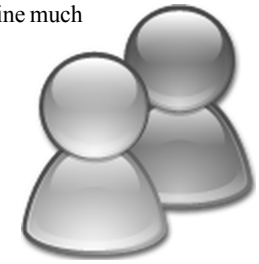
You can also get the current problem from the accu-general mail list (next entry is posted around the last issue's deadline) or from the ACCU website (http://www.accu.org/journals/). This particularly helps overseas members who typically get the magazine much later than members in the UK and Europe.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

Listing 1

```
// Read dates, and display as sorted text
#include <time.h>
#include <iostream>
using namespace std;
#define CLOCK_YEAR_OFFSET 1900
int main()
{
  tm *tm = new::tm[10];
  time_t time_t;
  char b[16];
  int i = -1;
  char ch;

  cout << "Enter up to ten dates (yyyy-mm-dd)"
          ", end with 'exit': ";
  while ( cin>>tm[++i].tm_year>>ch
            >>tm[i].tm_mon>>ch
            >>tm[i].tm_mday )
    cout << ": ";

  for ( int j = 0; j != i; j++ )
  {
    int to_year =
      tm[j].tm_year-CLOCK_YEAR_OFFSET;
    int to_month = tm[j].tm_mon-1;
    int to_day = tm[j].tm_mday;
    for ( int k = j; k != i; k++ )
    {
      if (((tm[k].tm_year-CLOCK_YEAR_OFFSET)
            > to_year)
          ||
          (((tm[k].tm_year-CLOCK_YEAR_OFFSET)
             == to_year) && (tm[k].tm_mon-1
             > to_month))
          ||
          (((tm[k].tm_year-CLOCK_YEAR_OFFSET)
             == to_year) && (tm[k].tm_mon-1
             == to_month) && (tm[k].tm_mday
             > to_day))
           )
      {
        swap(tm[j], tm[k]);
        to_year =
          tm[j].tm_year-CLOCK_YEAR_OFFSET;
        to_month = tm[j].tm_mon-1;
        to_day = tm[j].tm_mday;
      }
    }
    tm[j].tm_year= to_year;
    tm[j].tm_mon = to_month;
    tm[j].tm_mday = to_day;
  }
  while ( i-- )
  {
    strftime(b,sizeof(b),"%a %d-%b-%Y",
      localtime(&(time_t=mktime(tm+i))));
    cout << b << endl;
  }
}
```



Prizes provided by Blackwells Bookshops and Addison-Wesley

Bookcase

The latest roundup of book reviews.

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

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

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

Languages

Core Python Programming

by Wesley J. Chun, Prentice Hall, 2007, ISBN: 0-13-226993-7, Pages: 1077

Reviewed by Ivan Uemlianin

This is a fairly terrible book, but I hesitate to say 'not recommended' because I think inside it there is another book trying to get out which is not bad.

After some introductory material, the book works bottom-up through the language. 'Core Python' takes up the first 670 pages, and the next 350 take us through a selection of 'Advanced Topics'. The format is familiar: a chapter discussing some aspect of the language, followed by a set of exercises; gradually the reader develops a firm understanding.

The book's failings come in several categories:

- Errors, including the front cover (the cover promises 'Includes brand new chapters on ... Java/Jython and Microsoft Office.' – actually, the 13 pages on MS Office and four pages on Jython are contained in a chapter called 'Miscellaneous'); code that doesn't work; code that doesn't tally with the text describing it; code that doesn't produce its purported output.
- Inconsistencies: it gives two different definitions of floor division (a wrong one and a right one); recent additions to Python are simply appended onto sections describing earlier features, even when these additions break backwards-compatibility (e.g. exceptions on p.394). The effect becomes surreal when we read things like 'this feature [string-based exceptions] will be obsolete beginning with Python 1.6' (p. 386).
- Idiosyncrasies: the author makes some odd choices about what to include and when. Rarely used functions like `id()` (p. 113) are discussed early on, while for really useful features like regex flags the reader is referred to 'the documentation' (p.686).

On top of all this, the text is excessively verbose, leading to poor explanations precisely where clarity is needed.

The inner book would be a mildly interesting, if aimless, ramble through the lesser-travelled by-ways of Python – most, perhaps all, historical releases of Python are discussed here. Occasionally a topic is discussed well – for example class attributes, and static and class methods (p. 537f), but note that the code on p. 538 does not tally with the text – but such moments are rare.

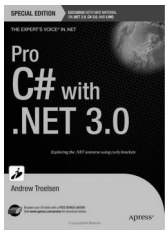
This is not a book for learning Python: for that I recommend *Learning Python* (Lutz & Ascher, 2003, O'Reilly). *Learning Python* covers Python 2.3 but the content is perfectly applicable to the current Python 2.5, and features new to Python since then can quickly be picked up. That a tutorial book covers the current version is not the most important factor.

If you have a lot of time and tolerance, this book might provide some light reading.

Pro C# with .Net 3.0, Special Edition

by Andrew Troelsen, Apress, 1186 pages, ISBN: 978-1590598238

Reviewed by Frank Antonsen



Conclusion: Recommended.

At first I was very excited about this book, then I was somewhat disappointed and then I got excited again. Sounds confusing? The initial excitement was due to the title. A whole book about C# 3.0, already! The disappointment was due to reading the table of contents where I spotted chapter titles such as 'Doing XYZ in C# 2.0'. Then I read the back cover again, and it was quite honest: this was a C# 2.0 book updated with 100+ pages about C# 3.0. Apparently somebody at the marketing department at Apress got carried away..... But when I started reading the book, I was again excited.

This is a very good introduction to C#, especially if you know some C-like language already, which should be true for all readers of an ACCU journal.

What I particularly like about the book is that it shows how to write console application before it discusses GUI programming or web services. Even more advanced topics as nullable types and writing your own iterators and indexers or using reflection are covered quite early on in a clear and natural fashion. I would have liked to see generics covered earlier as well, though, but the coverage is actually quite good.

The book furthermore contains a thorough discussion of IL (Microsoft's Intermediate Language, which all .Net languages compile to), together with dynamic modules (the Reflection.Emit namespace) threads, application domains and strong naming assemblies. After this it moves on to Windows forms, GDI+, web services and ASP.Net and ADO.Net and database access.

In general the coverage is clear and concise, and a nice touch is the emphasis (although only in the initial chapters) on how to use alternatives to Visual Studio such as SharpDevelop. There are

Bookshops

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

- **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

a few mistakes or omissions, though. For instance, claiming that the .Net attribute system gives full support for aspect oriented programming, when in fact it only provides a way to attach meta data to types. Another example is the way events and event handlers are defined. The recommended idiom is to have an event handler have the signature (object sender, **EventArgs**), where the user can subclass **EventArgs** to provide special information. None of the custom examples adheres to this. Furthermore, it was only in .Net 1.1 that you had to define first a delegate and then an event; in .Net 2.0 there is the short cut **event EventHandler<T>** where **T** is any subclass of **EventArgs**. Finally, in C# 2.0 there is no need to explicitly cast delegates, you can instead rely on inference by the compiler, so you can write code like this:

```
obj.MyEvent += MyHandlerMethod;
```

instead of the older style:

```
obj.MyEvent +=
   new MyEventHandlerDelegate(
   MyHandlerMethod);
```

But these are all minor issues.

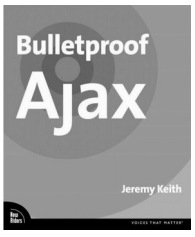
The C# 3.0 part is based on a beta release, but gives a nice overview of WPF (Windows Presentation Foundation – the declarative alternative to WinForms), WCF (Windows Communication Foundation – for programming web services etc.) and WF (Windows Workflow Foundation – for writing resumable code logic in a more declarative manner) and, of course, LINQ (the Language Integrate Query syntax for extracting data from collections, data bases, objects, XML files in a consistent style). This coverage is fairly good, although some of it is already obsolete. We now have C# 3.5 in beta

To summarize: if you are new to C# this is certainly a very good book. If you are already familiar with C# and wants to move on to version 3.0 (or 3.5), then you should probably invest in some of the dedicated books. There are a number of good books, published by Apress or by Addison-Wesley, covering each of the new technologies in more depth.

Bulletproof Ajax

by Jeremy Keith, published by New Riders, ISBN: 0321472667, 207 pages

Reviewed by Simon Sebright



I liked this book. It's a nice read-on-the-train size in softback, and a good smooth read in terms of content.

His basic idea is that a web application should have similar behaviour whether or not Ajax is available (perhaps because the browser setting does not allow Javascript). This leads to an architecture where server-side code is in charge of the html content, not client-side as is common in Ajax applications.

The book starts with an introduction to Ajax, Javascript and the XmlHttpRequest object. This was well-delivered. Compared to another book I recently read, it was to the point, and included more robust details for certain situations.

We then have a discussion of data formats including XML and JSON.

The chapter called 'Hijax' puts forward the concept of progressive development of Ajax applications. Start with a regular web page served up by some server. Then begin to intercept the links and button presses with Javascript when allowed to only refresh certain bits of the page as necessary. In the absence of this intervention, the regular web page refreshes will occur meaning that you don't lose functionality.

It notes that some applications may be so useless without Javascript that no other approach matters (consider a certain mapping application for example).

Next comes a consideration of problems with Ajax applications. For example, because whole page refreshes do not occur, extra effort needs to be put in to make the user feel that something is happening.

The final part is a 'put it together' chapter where various ideas are illustrated in a simple sell-some-stuff application. A real application will be a lot more, but the ideas in the book are clearly expressed.

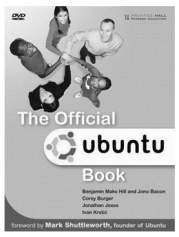
In summary, a refreshing approach for many web applications/sites.

Operating Systems

The Official Ubuntu Book

by Benjamin Mako Hill, Jono Bacon, Corey Burger, Jonathan Jesse and Ivan Krstic, published by Prentice Hall, Pearson Education, ISBN 0-13-243594-2

Reviewed by Omar Bashir



The striking issue with the text appears to be its organisation and probably some confusion with regards to the target audience. If it was meant for a Linux novice, some parts dived into unnecessary and at times incomplete details, which made it frustrating for even moderately experienced users. The book even attempts to make users understand fundamental windowing operations such as how to close a window. Most current Linux users have experienced other WIMP user interfaces and this level of detail was not only unnecessary but also inconsistent with other parts of the text. At times it appeared that the authors attempted to make the chapters self explanatory as there was a noticeable amount of repetition within the text and the text did not flow well between the chapters and at times also within the chapters.

The book explains various Ubuntu installation aspects quite well; however, it does not do quite as good a job towards performing system administration operations by an individual user (e.g. a desktop installation). Although Ubuntu is quite user friendly, it does require rather elaborate system administration. In my experience with Ubuntu, its administration can be mastered potentially by understanding a few fundamental operations and then building on that knowledge. The book should have focused on those operations (e.g., using Synaptic to install applications and various utilities and commands for system administration and diagnostics) rather than describing various applications. Most Ubuntu applications are the same as on other Linux-based operating systems and are fairly user-friendly even for novice Linux users with some Windows experience. Description of each one of them is, therefore, unnecessary for a book on the operating system.

Description of the evolution of Ubuntu, its various projects and the community effort, form an interesting reading. The support cases also provide insight into troubleshooting various common issues but at times do not go far enough.

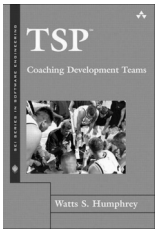
This book is recommended if the potential reader is not familiar with the Ubuntu project or Linux and intends to start using Ubuntu. However, experienced Linux users, especially Debian users and IT professionals may be familiar with many more resources that may provide more comprehensive information on Ubuntu installation, system administration and operation.

Miscellaneous

TSP - Coaching Development Teams

by Watts S. Humphrey, published by Addison Wesley, SEI Series in Software Engineering, 2006, ISBN: 0-201-73113-4, 416 pages

Reviewed by Allan Kelly



Coaching seems to be everywhere these days. I know I've been banging on about it for a couple of years but I am not alone; Phran Ryder wrote about it in CVu last year; several of the Agile methodologies suggest a team coach; and the general press seem to cover business or life coaching every month. Now the Software Engineering Institute at Carnegie Mellon University (yes, the people who brought you CMM and CMMI) have decided to jump on the bandwagon too.

I should perhaps state my bias up front. Good stuff does come out of the SEI but my first reaction to anything they produce is scepticism. I know this isn't good of me but I recognise the bias and try to compensate. In reviewing this book I'm trying to be even handed. And in truth, a lot of what does come form the SEI is well researched and thoughtful.

So to the book. This is one of many books from the SEI that covers the 'Team Software Process'

or TSP for short. Now I don't know much about the TSP so in reviewing this book I was hoping to learn a lot about coaching development teams and a little about the TSP. As it is, I've learned a little about coaching development teams and a little about the TSP.

The book is completely true to its title. Lets take it word by word:

- TSP: The book is TSP based, TSP runs through everything in the book. I felt at a disadvantage because I don't know the TSP and this limited what I could get from the book. Neither is it a book to learn TSP from.
- Coaching: it seems the TSP takes a very directive approach to coaching and assumes the coach is the TSP expert on the team. Manager and developers look to the coach for help running the process and resolving process issues.
- Development: its about software development teams (who else would use TSP?)
- Teams: between the TSP advice there is good advice for team coaches. Unfortunately the author tends to use more words then necessary. The book could probably have been written in half the number of pages.

The feel of the book is culturally biased towards large American corporations, maybe these are the only people who use TSP in which case it doesn't matter. But if you are European, Indian or a small corporation I wonder if you will find the book so applicable. Other than this the book is well written and the author knows his subject.

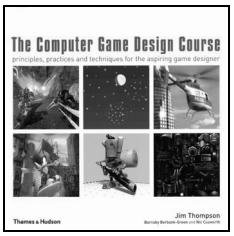
In summary this book is true to its title. It is about coaching TSP teams. If you are coaching another type of development team you will learn something here but there are other sources were you could learn more in fewer pages. If you are coaching a TSP team you might well find this book very valuable.

To the best of my knowledge nobody has yet written a good process-neutral book on coaching development teams. In the mean time if you need to coach a team, and they are not following TSP, I suggest you read some of the more general books on management coaching.

The Computer Game Design Course – principles, practices and techniques for the aspiring games designer

by Thompson, Berbank-Green and Cusworth, ISBN 978-0-500-28658-6.

Reviewed by Ian Bruntlett



This book is very much like a coffee table book in terms of build. It comes in at approximately 200 pages, filled with lush illustrations. It is split up into 3 parts – design theory, design process and design production.

The digital theory section explains the advantages of different types of game – shoot-'em-ups, platform, strategy, puzzle.

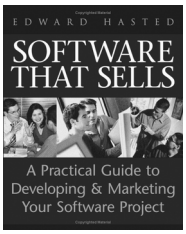
The final two sections show the use of a process to design a game. It discusses the design of character, environment and physics modelling. While it is biased towards studio-style development, the material presented, a bit like SSADM (Structured Systems Analysis and Design Methodology) can also be adapted by 'bedroom coders'.

Verdict: Recommended.

Software That Sells

By Edward Hasted, published by John Wiley & Sons Inc (30 May 2005), paperback - 379 pages, ISBN-10: 0764597833

Reviewed by Paul Grenyer



This book was not offered for review by the ACCU. I bought it as, like many software engineers, I have ideas for my own products and I would like to work for myself writing my software. Although I am a competent and experienced software engineer, I lack both the skills and experience of researching, funding, marketing and selling a product. I am writing this review for the ACCU as I suspect many other members are in a similar position and would therefore find it useful.

The first few chapters cover how to develop your idea, how to identify if it is likely to succeed, who to speak to and what questions to ask them, how to gather and interpret market research and how to plan the project. Hasted's approach is straight forward common sense explained in a clear pragmatic way. I was pleased to find I had already started to do many of the things he suggests, such as gathering a small 'inner sanctum' of colleagues with whom to discuss my ideas and gain valuable feedback, and writing a project plan. The chapter on market research confirmed much of what I suspected in terms of who to contact and what to ask, but also helps by documenting a structured approach.

The following chapter on making your company somewhere people want to work and how to find and utilize good people gives idealistic advice. I think a better approach would have been to state that the advice was idealistic and acknowledge that it's very difficult to get it to work in practice. I found some suggestions, like isolating developers in booths to work and preventing any contact with them other than via the project manager and that a webcam could be a way of monitoring remote workers as highly suspect. I can see what Hasted is trying to achieve, but experience tells me this is not the solution. Developers should be in an environment where they can both concentrate and interact with their team when necessary. The use of phones, an instant messenger system and email are all important tools for communication between team members both inside the office and for remote workers.

The chapter on raising cash and kind introduces the concept of venture capitalists and business angels. At least one of these was a term I had at least heard before and it is good to have them explained in a reasonable amount of detail.

The later chapters in the book covered topics that I had not even started to think about such as finding suppliers, shipping in bigger and bigger volumes, customer service, how to keep customers and what to do if you need to sell your company or if someone wants to buy it.

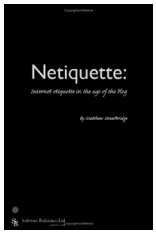
This book covers an extraordinary range of topics related to market researching software, writing software, creating and running a company and marketing, distributing and supporting software. It is not a big book, but does seem to include quite a lot of detail, mostly in the form the author's experience. The book gave me almost everything I was looking for and a fair bit more. My only disappointment was the description of development techniques, but this is the area I know the most about and have experience of to compare with Hasted's views. There are other books that cover all the topics in greater depth, but this is an excellent place to start and I'm looking forward to putting many of the suggestions into practice.

Recommended.

Netiquette: Internet etiquette in the age of the blog

by Matthew Strawbridge, ISBN: 0955461405, Software Reference

Reviewed by Giuseppe Vacanti



Anybody who can remember the net before it was called the Internet will be at home with Matthew Strawbridge's (MS) book: it contains known advice from the days of net yore, updated and expanded for the many new communication channels available on today's Internet. The book, however, will be useful to veterans and neophytes alike.

Netiquette, should you not be familiar with the term, is the informal framework of rules and conventions governing how people should behave online. The origin of the word, we learn, can be traced back to 1982: the issue of how to behave online is almost as old as the net itself.

Online communication must do without the many visual and audio cues that help people understand one another when they speak face to face. Add to that the fact that often members of online communities come from different countries and do not always master the language that is used, and you have all the ingredients that may lead to your misunderstanding others or being misunderstood. Proceed with care, think twice before you click on the send button, give your interlocutors the benefit of the doubt, and resist the temptation to immediately fire off a

[continued on back page]

ACCU Information

Membership news and committee reports

View From The Chair

Jez Higgins
chair@accu.org

In the View before last, I was still recovering from the conference (as if we ever truely recover), and this time round finds me chewing over my weekend at PyCon UK. I didn't go for the full sessions-all-day-carousing-all-night conference experience this time around, as the organisers had conveniently arranged a venue a 15-minute cycle ride from my front door, but my brain's still buzzing.

PyCon UK, held over the weekend of the 8th and 9th of September, was the first dedicated Python conference held in the UK. As you are probably aware, for the past several years the ACCU Conference has hosted the UK Python conference as a track. This year it took flight (slithered off?) as an event in its own right, with great success.

Running four tracks, there were over 35 sessions packed into the two days, with 200+ people attending. Sessions ranged from the introductory to the really rather hardcore, and every session I went to was well attended. The atmosphere was relaxed and well organized. There were even one or two ideas, like double-sided name badges, we should steal for the ACCU conference.

I'm not even anything like a full-time Python programmer, but the whole thing was great fun and I enjoyed myself immensely. The 2008 PyCon UK has already been announced, so if you fancy a trip to Birmingham next September you should seriously think about going.

If I can be so bold, I'd like to extend congratulations on behalf of ACCU to John Pinner and the organising committee for putting on such a splendid event.

Membership Report

Mick Brooks
accumembership@accu.org

As I'm writing this the flood of renewals is slowing to a trickle. I knew it was coming, but was still surprised by the size of the peak at the traditional renewal time of end of August. This was a real test of the new membership system, and things didn't run perfectly.

The first email reminder should have been sent three weeks before the end of August, but somehow went missing, so that people were only reminded one week before their expiry date. At a time of year when many people are on holiday, I know that this is not enough time. There was then lots of confusion about login details for the website (hopefully a one-off problem). I also managed to confuse our members who pay by standing order by sending them reminders after they'd paid. Finally, I've had some good feedback on website usability issues, with a number of members unable to find how to renew. I'm sorry if you were caught out by any of this; Tim Pushman and I are working to correct all of these problems.

Despite all of this, we managed to help the vast majority of members to renew. From a membership of close to 1000, I've been notified of about 5 resignations, and approximately 25 more people have (so far) let their membership lapse; suggesting that we will lose a maximum of 30 members.

With this year's renewal just out of the way, you're probably not thinking about next year's yet. However, now would be a good time to arrange a standing order. You'll save time and money – we offer a £5 discount for members who pay this way. If you're interested, I can give you details of who, when and how much to pay: just email me at accumembership@accu.org

Website Report

Tim Pushman
webmaster@accu.org

In the last issue of CVu (August 2007), Allan Kelly mentioned that he would be stepping back from his involvement in the ACCU website. Allan has been closely involved in creating the website and, if you've read his earlier reports, it's been quite an adventure. Allan has put tremendous energy into prodding and cajoling people to create, as he puts it, a 'shiny new website' and his input will be missed. Although he hasn't been involved in the 'grunt work' he has been instrumental in getting things moving. Now that the project is established he can step back and concentrate on other things (a few articles for CVu/Overload maybe???).

I'll be putting together a regular report on happenings on the website and putting out ideas for future plans and directions. Although a website relaunch is a small item for a large corporation, for the ACCU, an association made up of volunteers, it has been quite a major project. Most members have an hour now and then to spare and that is usually tied up with creating the bi-monthly magazines, planning the next conference, moderating the mailing lists, running a mentored project, handling membership enquiries, pulling in advertisers and a dozen other odds and ends. Some even find time to earn a living!

In this on-line world, the website serves as our public face and, with the mailing lists, as a focus for the association. Some of the main aims and aspirations of the site are: to increase the membership, provide more resources for programmers in general, provide a focus for the ACCU community and to supply backend services for running the association.

The website itself is handled by Tony Barrett-Powell, who is the editor dealing with site content, and me. As it is a CMS (Content Management System) there is the possibility for multiple editors, and there are others taking care of specialised areas such as the conferences and mentored projects. As you may gather, volunteers are always welcome to look after parts of the site and if you feel like you want to get involved, then contact Tony at webeditor@accu.org.

Logos

A couple of issues back our publicity officer, David Carter-Hitchin, mentioned that it would be useful if those of us with blogs or websites would put a link back to ACCU. I realised that my website had no such thing, and I also realised that the ACCU doesn't have a selection of ACCU logos that people can download or link to. So, I hope that by the time you read this, there will be some ACCU logos that you can link to from your blogs. I'm not sure where they'll be available yet, look for an announcement on the site.

Membership system

Some months back the whole ACCU membership system was put online. This wasn't simply a custom built application, it also had to integrate with the existing Xaraya framework and as such consists of three interacting modules. There were some bugs and unwanted features to iron out, but now it seems to work fine and the end of August was the time for a real test, as that is when the majority of the ACCU memberships are renewed. Whereas previously this would have meant taking data from the web, putting it into an Access database and exporting the data from there, it is now all handled on the web (with a little help from Worldpay), including generation of mailing list labels and reminder emails.

Mailing lists

August saw the move of the mailing lists to the new server. These have been one of the last things to move from brian.accu.org. And, thanks to Steve Dicks having converted the existing Majordomo lists to Mailman, the move went very smoothly. Information on the Mailman lists can be found at http://lists.accu.org/mailman/listinfo/. Having moved the mailing lists, I forgot to turn off the Mailman program on brian.accu.org and was suddenly reminded when, on Sept 1st, I received a bunch of subscription reminders from brian!

ACCU events

There have been a growing number of local ACCU meetings taking place and we now have a section of the website devoted to just that (see 'ACCU Near You'). Plus there is an 'Events

Calendar' on the site for an overview of what is happening (this includes other programmer related events). These pages also include the ACCU USA chapter, one of the most active of the regional ACCU groups.

Book reviews

The book reviews were not available for a couple of weeks at the end of August due to a mix of events, including a server hardware upgrade and a licence that was linked to the hardware... for now, things are back to normal, but the licensing situation will need to be worked out on a more long term basis.

Stats and such

The site continues to slowly grow in traffic. There is normally a peak in April/May during and after the annual conference, but the number of visitors has stayed high since then, in fact August saw nearly as many visitors as May. On average there are about 1,000 visitors per day (700 at weekends). The distribution of visits used to reflect the European membership bias but there have been an increase in visits at 4am, which suggests more programmers in Asia are dropping by. Interestingly, accu.org is the only website I manage that shows Microsoft Internet Explorer for less than half the visitors (46%). Most sites count about 70%.

So, I'll finish with a quick thank you to Allan Kelly for getting the website moving, and to Tony, Mick and Ewan for keeping it rolling.

Book Reviews (continued)

reply: these guidelines, implicit or explicit throughout this book, will serve you well when online.

The book comprises three parts. The first part covers the various forms of online communication. Electronic mail receives, and in my opinion deserves, the most attention: it must be true that everybody who is active online has an email account, and understanding how to communicate via email is the foundation of any other online communication form. Forums (newsgroups and mailing lists), real-time messaging, web sites, blogs, and wikis are also covered. MS draws from available guides and books, and his own experience, to list a number of rules that should be followed when communicating online through any of these means.

I must confess to being mostly an email-based person, and I was happy here to find some of my convictions (obsessions?) spelt out in writing: make a distinction between people in 'To' and 'CC' fields; four-line signatures; limit the use of attachments; use plain text instead of HTML. But there is plenty of sensible advice for each of the online channels, and if you are going to make use of any of them for the first time you'd be well advised to have a look at what this book has to say.

But the Internet is no longer a place where we only interact with other people by writing. More and more we make use of servers to download or share files, watch videos, listen to radio stations, or buy and sell stuff. Often we have only a vague notion that other people may be using the same service, and that our actions might affect them. Present day netiquette rules must address also these aspects, and this is what MS does in the second part of his book. The underlying principle here is: do not hog the service, and be honest and fair when operating online.

Finally, the third part of the book deals with some of the plagues of the present day Internet: spam, and the various security risks (viruses, identify theft, fraud in general). Of course, spammers and fraudsters will not be reading this book and stop their activities, but we ourselves can do something against them by behaving in certain ways. For instance, be sure you understand the economics of spam, and never reward spammers by buying any of the products they advertize. Here MS does a good job of explaining the dangers and what simple countermeasures we can take, in a manner accessible also to the less technically minded user.

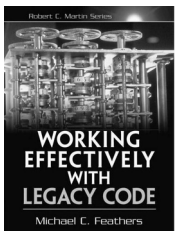
In summary, the book presents in a clear and concise manner the rules of engagement for online communication, from email to blogs and peer-to-peer networks. Some of the dangers of online communication are also discussed. Novices and seasoned users considering making use of one of the new online communication channels will find the book useful, and so will administrators of online communties and services who want to draft a code of conduct for their users. Finally, parents of cyber-teeangers might consider passing this book onto their offspring to help them learn the ethics of the net.

Recommended.

Working Effectively with Legacy Code

by Michael Feathers, published by Prentice Hall PTR (7 Oct 2004), ISBN-10: 0131177052, ISBN-13: 978-0131177055

Reviewed by Paul Grenyer



Everyone should read this book. Even people who are working exclusively on green field projects. It is a good book. It is very easy to read and, unlike many other large technical books, not boring. A lot of the time I couldn't put it down.

Most of the time Feathers was confirming that I was already doing the right thing and giving me names for the refactorings I was doing to get things under test. I also learnt some techniques that hadn't occurred to me such as Static Setter for testing code that makes use of singletons.

However, there were several things covered by the book that I wouldn't do. Feathers is obviously a big fan of object orientation and specifically inheritance. On more than one occasion I found myself thinking that I just wouldn't use inheritance there, I'd use an interface. Discussion with ACCU members revealed that a lot of people agreed with inheritance as a method of getting things 'safely' under test. However I feel it leaves you with worse and in some cases more dangerous code when methods that should be private are made protected or public for the sake of testing. I think Feathers could go a long way towards rectifying this situation by describing the use of inheritance as the first step to get the code under test prior to a further step of removing the inheritance and introducing another method, such as using an interface.

The majority of examples are in C# and Java, with a few in C++ and a small number in C (there is even one Ruby example). I think all examples should be in C# or Java, except where the refactoring is only applicable to C++ or C such as Link Substitution. The book has a number of examples that are not idiomatic to C++ and introduce more potential problems such as memory leaks under exception conditions. All the examples using pointers deference them incorrectly. Even if you are only using C++, you should still read this book.

There isn't a another book covering this subject so this is the book you should read. I would like to see a similar book aimed at a C++ audience from a C++ expert.

Again, this is a good book. You should read it.

Highly Recommended