The magazine of the ACCU

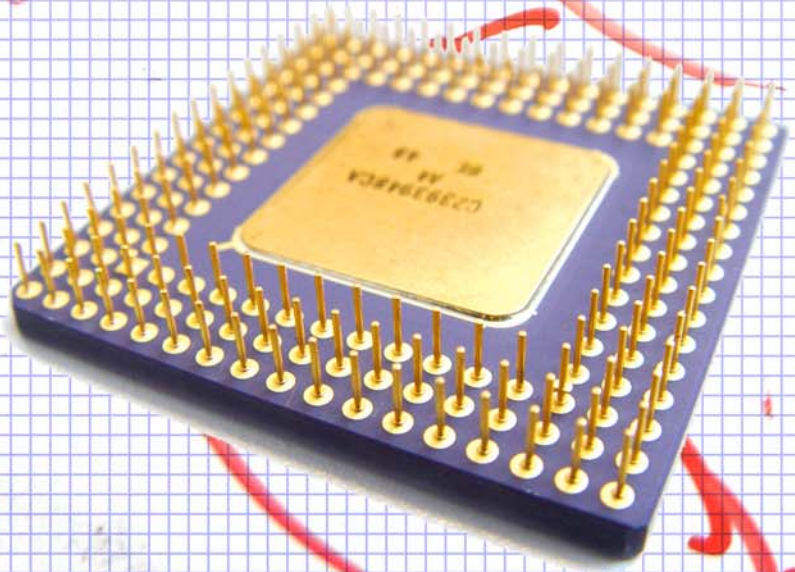
www.accu.org

{cvu}

Volume 19 Issue 6 December 2007 £3

Features

Regulars



Pete Goodliffe
Programming Posers

{cvu}
Volume 19 Issue 6
December 2007
ISSN 1354-3164
www.accu.org

It's summertime!

Well, at least in the southern hemisphere it is summertime, and in traditional New Zealand style, summer starts with a day of drizzly rain. However when the sun has been shining and the weather warm, I have been thinking of the UK Christmas break. Christmas in NZ mean barbeques and sunscreen, and pavlova (can't forget the pavlova) as opposed to the large heavy meals that make you want to sleep in the middle of the afternoon.

Winter in the UK means one thing however, and that is spring comes next. With spring comes the 2008 ACCU conference again held in Oxford. I'm expecting another fantastic conference with great speakers and keynotes, sensational evening discussions (sometimes with a drink in hand), and a terrific group of attendees to meet.

I'm having trouble sometimes realising that another whole year has passed. I look back and see a stack of C Vu mags that I have helped see the light of day. If there are other ACCU members who are wanting to help more with the production of the organisation's magazines, please email me. I must admit I'll be surprised if I get many, but that is the nature of a volunteer organisation.

I've found myself in an interesting position where I work now. I've gone from just being a software developer to being a manager. I'm still supposed to be writing code too, however I have come to realise that actually writing code is now somewhere around 50% of my expected work rather than somewhere around 90%. This has left me with a slight dilemma. I've never done any management before. I've just ordered a copy of a book from amazon.com which looks to be quite amusing – *Managing Humans* by Michael Lopp. I'll keep you posted on it.

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 20.1: 1st January 2008
C Vu 20.2: 1st March 2008

IN OVERLOAD

Kevlin Henney continues his series on 'The PfA Papers', Richard Harris investigates the travelling salesman in 'The Model Student' and much more.

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

Programming Posers

Pete Goodliffe helps us to adopt the position.

As modern software development project pressures increase, the demand placed on programmers grows, moving us from the once traditional 15-hour working day closer to the continental 26-hour working day. In such a climate, it's becoming increasingly important to ensure that you have a comfortable and ergonomically sound working environment.

This is perhaps as vital an issue to the 21st century programmer as good code design or any other software development practice. After all, you can't be an agile developer with a bad back, and you can't navigate a complex UML class diagram with failing eyesight. In order to improve the quality of your life spent in front of the computer, and perhaps to safeguard your physical well-being, we'll look in this column at how to tailor your working environment.

Pay close attention; if you don't get this stuff right you could end up with large medical bills! You'll thank me one day.

Basic computer posture

First, let's look at the most basic case of day-to-day computer use: sitting in front of a monitor (or as old-school Human Resources departments like to call it: a 'VDU' [1]). You probably do this for many, many hours a day, so it's vital to make sure that you sit correctly. Surprisingly, sitting down is a quite complicated task. It requires hard work and determination to master. As you work through this section, remember to take regular breaks – go for a run or something equally relaxing.

The way you sit at a computer has implications for not just your productivity (bad posture can have a surprisingly large effect on your concentration and, therefore, your productivity) but also your health. Poor posture can lead to: neck pain, back pain, headaches, digestive problems, breathing difficulties, eye strain... the list goes on. These are the ergonomic experts' recommendations (see figure 1):

1. Adjust your chair/monitor position so that your eyes are level with the top of your screen and your knees are slightly lower than your hips. Adjust the monitor so that it is a comfortable distance from you (say about 18 to 24 inches).
2. Your elbows should rest at an angle of about 90 degrees. You should not need to significantly move your shoulders when typing or to use the mouse. To achieve this your keyboard should be at about elbow height.
3. The angle of your hips should ideally be 90 degrees, or slightly more. (You're thinking about this as you read, aren't you?)
4. Your feet should rest flat on the floor; do not tuck them under your chair. Don't sit on them, either – you'll get terrible leg-ache and a footprint on your rump.
5. Your wrists should rest on the desk in front of you. (It's certainly very poor posture to let them rest on a desk behind you. Unless you are severely double-jointed.) Your wrists should remain straight when typing.
6. Adjust your chair to support the lower back.

To avoid problems:

- Shift your position throughout the day to keep your muscles loose and to ease tension in your body.
- Take plenty of breaks, and walk around the office. You may find it beneficial to talk to other people. After a little practice oral communication can become second nature, even to the seasoned programmer.
- Don't collapse your neck as you read the screen. Hold your head high, and be proud to be a programmer.
- Defocus your eyes occasionally. Try those auto-stereogram thingies that were popular in the 90s. Or look up from the screen and focus on a distant object (perhaps you could look longingly at the door, for example, or that nice temp in accounts).

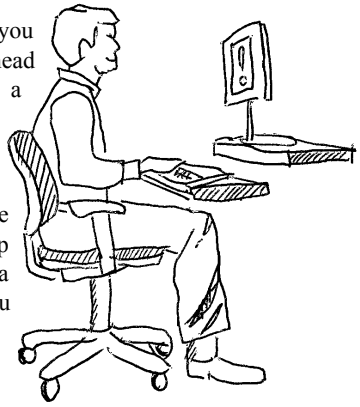


Figure 1

- In truly extreme cases of muscle fatigue you may find it necessary to take drastic action: step outside of the building (yes, it is perfectly safe to do this) and take an extended stroll. If the stroll gets too relaxing you'll find plenty of seats in the local park where you can practise sitting down momentarily.

Having determined a good posture for the basic case of computer use, let's now look at some of the less-considered postures required by the modern programmer. After all, it's important to ensure that we remain ergonomically sound during the entire day...

The debugging posture

Code got you down? Are the gremlins refusing to budge? Have you been concentrating for six hours flat, yet you still can't work out why there's an ugly brown rectangle on the screen when you should have an elegant turquoise octagon?

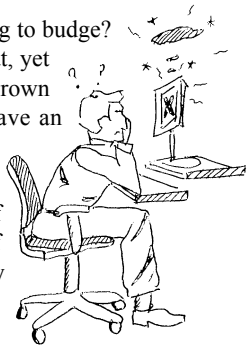


Figure 2

In this case your body needs a slightly different posture to accommodate the weight of the world on your shoulders and the shift of your cerebral cortex from the top of your body to somewhere inside your shoe. To adequately support your body and prevent further strain (unfortunately, the brain-strain is unavoidable) follow these steps:

- Lean forward slightly (a hip angle of 45–60 degrees is best)
- Place your elbows on the desk on front of you (ideally, they should rest at your wrists' position when typing)
- Extend your forearms vertically upwards
- Lean your head against your arms
- Sigh

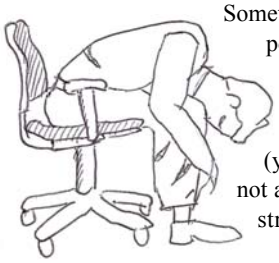
Figure 2 illustrates this. In these situations you may also find it more comfortable to move the monitor a little closer to the front of the desk than you would ordinarily have it. You'll find that this makes it much easier to repeatedly bang your head against it when you're feeling particularly frustrated.

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

When it's really going badly

Figure 3



Sometimes, despite adopting a careful debugging poise you won't be able to solve that thorny problem. The bugs just won't budge. You can look as nonchalant as you want, they just don't seem to respect your determined (yet comfortable) posture. Programming is not always plain sailing, and sometimes all that straight-back, slack-shoulder nonsense can take a running jump...

If you find that it is going really badly, then adopt the position shown in figure 3, and brace yourself for when it all crashes and burns around you.

The all-nighter

When deadlines loom you may find yourself working heroic hours to get everything finished in time. Of course, you know that no-one is going to thank you for it, but a sense of moral obligation and a pride in your work will compel you to stay up three nights in a row and to live off a diet of caffeine and stale donuts.

Figure 4

You will find the posture in figure 4 particularly useful after the fourth all-night stint. Like any ergonomic consideration, the really important thing here is to adjust your working environment to help you. If possible, shut the blinds and close the door to block out extraneous noise or anything that might distract you from your current task. If you work in a loud communal area with many people walking past all day, then arrange your desk and chair so as to offer maximum potential to not be seen.

Try not to snore too loudly. You may find it useful to insert the mouse firmly into your mouth to plug the airway. (Remember not to do this if you have a blocked nose, or you might asphyxiate yourself).

Intervention from above

Figure 5

Occasionally a boss feels compelled to prowl around to ensure that his minions are working as hard as pack mules. In order to ensure his maximum comfort and to prevent him from straining his delicate aggravation muscle, you should adopt the posture shown in figure 5. It's for his own good:

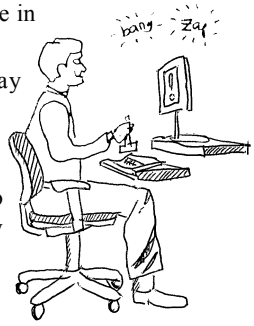
- Employ a taught, pained posture. Tighten all your muscles, and look like you are poised to chase after a burglar.
- Adopt a screwed-up facial expression (if this is not already your natural appearance after years of programming). Something along the lines of severe constipation gives an adequate appearance of extreme concentration.
- For the best effect, purchase some dry ice (this can be readily obtained from stage supplies store) and leave it under your desk. The boss will be impressed at the perspiration generated by your fervent work. Don't be tempted to over-do this though, or your colleagues may become concerned at your problems with flatulence, or security may call the fire department.

Ideally, orient your workspace so that your back is against a wall so that no one can walk up behind you unawares. Adopting the figure 5 posture at very short notice can lead to sprained muscles (especially if you have to rapidly remove your feet from the desk) and fused nerves.

The all-clear

Be careful when adopting the above posture to not screw your eyes up too much. It's important to be able to see your boss walk away so you know when it is safe to relax, and adopt the posture in figure 6.

Figure 6



You will find that using a joystick to play network games requires rather less wrist strain than a keyboard, and so it is preferable to use one, where available. Creative filing of expenses claim forms should enable you to justify the purchase of a very good quality gaming device. Do not consider Nintendo Wii controllers in the office – they are not especially subtle.

Design time

Figure 7

Our final programmer posture should be employed when designing new code, or working on extremely hard problems. At these times it is important to ensure maximum comfort and that you will not be distracted by your surroundings.

You should find that figure 7 is fairly self-descriptive.

Eye strain

Finally, it is valuable to spend a little time considering the health of your eyes. Make sure that as you peer at your monitor you don't strain your eyes. Take frequent breaks. Ensure your screen doesn't suffer excessive reflections from windows or lights; move the screen if this is a problem. Make sure that direct light sources (a window or a lamp) are not directly pointing at you, too.

If you have a CRT monitor, consider adjusting the refresh rate to make the image more stable. Can you change the contrast and brightness settings on the display to make it easier to read?

Regular eye tests are essential. Here's one simple test you can try in the comfort of your swivel chair, which doubles as a good regular eye exercise: Print a copy of figure 8 and hang it on the wall above your desk (you may need to experiment to work out the best distance you should be from the chart). From time to time during the day move your focus away from the computer screen and look at the chart. Start by reading the top letter, and work steadily downwards. Read as far as you can to the bottom.

Figure 8

Until next time...

I hope that you have found this information useful, and that it helps you to maintain a good posture as you continue to cut great code. See you next issue, if your eyesight doesn't pack up before then. ■

Endnotes

[1] Vision Destruction Unit

Pete's book, Code Craft, has been out for a while now. It's quite heavy, so you might need to adopt a careful posture when you read it. He wrote a lot of it using figure 7.

Check it out at www.nostarch.com

Distributed Version Control Systems

Ian Clatworthy describes the why and how of DVCS.

Abstract

The Version Control space is undergoing a renaissance right now thanks to the increasing popularity of Distributed Version Control Systems (DVCS) such as Arch [1], Bazaar [2], BitKeeper [3], darcs [4], Git [5], Mercurial [6], Monotone [7] and SVK [8]. This paper explains why this technology is useful today and will be important in the medium to long term for most software development teams, whether open source or commercial. Guidelines are also suggested for selecting a tool and recommendations are presented on how to use the technology effectively.

The challenges of software development

While there are important differences between open source and commercial development models, the core challenges are largely the same and timeless. As Fred Brooks observed many years ago, there will never be a silver bullet [9] to slay the software development beast. Most interesting software is inherently complex, users want it as soon as possible and typically have low tolerance for poor quality.

In the open source world and many commercial environments, a program is never finished while it has users with new ideas for what it can do, changing requirements, new environments or bugs. Software development is therefore not just about static design but very much about managing the evolution of design, communicating among the contributors and helping people understand what was done before.

To generalise further, Software Engineering is ultimately a communications challenge [10]. Version after version, the path from ideas to released code is often a long one involving many players . In commercial organisations for example, these include customers/users, sponsors, business analysts, architects, team leaders, software engineers, technical writers, quality engineers, support engineers, etc. The history of computer science is arguably one long procession of technologies designed to make that communication chain/cloud a more reliable one: higher level programming languages, OO, Use Cases, UML, design patterns, Agile development methodologies and beyond. The core challenge of software development is ultimately this:

> How do we collaborate more effectively, both instantaneously and over time?

The role of configuration management

Configuration management is a foundation best practice of every popular software engineering methodology regardless of the methodology type. Whether a waterfall, spiral, iterative, agile or open source approach is used, CM is fundamental because version control tools have proven themselves to be the most effective way, in terms of signal to noise ratio, of technical people communicating changes across the room to each other today, let alone across the globe over time. As a consequence, improvements to VCS technology have a direct impact on collaboration effectiveness. It is therefore no surprise that open source leaders such as Mark Shuttleworth and Linus Torvalds are among the driving forces behind this technology. Mark has stated that [11]:

> Merging is the key to software developer collaboration.

Linus has strong views on how VCS tools support development or otherwise. [12]

Changing the game

There are numerous pros and cons to using a distributed VCS tool. As with most technologies, these vary depending on one's perspective: Developer vs Release Manager vs Community. Collectively, the benefits do add up and most early adopters of the technology are using it today for one or more of the reasons presented. For the vast majority of teams though, important questions remain:

1. Could some of the benefits be realised by waiting for later versions of existing central VCS tools?
2. Are there compelling reasons why distributed VCS technology will become mainstream regardless?

The answer to both questions is 'yes'. I fully expect better central VCS tools (or the promise thereof!) to delay mass adoption of distributed VCS tools in the short term, particularly while the DVCS tools are maturing. In the medium to long term though, the compelling benefits of DVCS technology will shine though.

Ultimately, a tool is just a tool and the real benefit comes from the processes it enables. There are two reasons I believe distributed VCS will ultimately replace central VCS technology across the industry:

- Better adaptability
- Used wisely, it delivers higher quality software

These benefits are explored further below.

Distributed VCS in a nutshell

Wikipedia offers the following definition [13]:

> Distributed revision control takes a peer-to-peer approach, as opposed to the client-server approach of centralized systems. Rather than a single, central repository on which clients synchronize, each peer's working copy of the codebase is a bona-fide repository. Synchronization is conducted by exchanging patches (change-sets) from peer to peer. This results in some striking differences from a centralized system:
>
> - No canonical, reference copy of the codebase exists by default; only working copies.
> - Common operations such as commits, viewing history, and reverting changes are fast, because there is no need to communicate with a central server.
> - Each working copy is effectively a remoted backup of the codebase and change history, providing natural security against data loss.

While this definition is technically correct, practically there is always a single copy that is sanctioned as the main development branch in teams using distributed VCS tools. Over and above the potential for disconnected operation, the primary differences between distributed and central VCS are these:

- Developers can collaborate directly without needing central authority or incurring central administration overhead
- The acts of **snapshotting** changes and **publishing** changes can be decoupled.

IAN CLATWORTHY

Ian joined Canonical in early 2007 to work on Bazaar and other collaboration tools used to build Ubuntu. He can be contacted online via http://ianclatworthy.wordpress.com/ or ian.clatworthy@internode.net.

These differences mean that the integrity of the main trunk remains higher over time. This has a positive impact all round:

- developers are continuously working from a more stable base (so much less time is needed tracking down why their last update command has broken their sandbox)
- better quality is delivered when the time comes to ship reducing stress on the QA and release management roles/teams
- the VCS server set-up and managed by the system administration team does not need as much disk space, CPU, bandwidth, etc. to accommodate the concurrent load needs of the team/community. [14]

The developer view

One of the reasons why DVCS technology is gaining popularity is because developers seem to prefer them. Some notable benefits are:

1. **Disconnected operation** – developers can still be productive when the umbilical cord to their central VCS repository is broken, e.g. when travelling.
2. **Experimental branches** – creating and destroying branches are simple operations. This is particularly useful when experimenting with new ideas, e.g. a 'spike' when using eXtreme Programming.
3. **Easier ad-hoc collaboration with peers** – intelligent merge tracking means merging early and merging often is both possible and surprising painless. It is difficult to explain just how much of an impact this can make on how co-developers can work together more easily, e.g. when Pair Programming.
4. **Staying out of the way** – collectively, the above features mean that developers spend less time on mechanical chores and more time on tasks that add value.

New ways of building companies and teams are now a reality and distributed VCS is one of the important technologies required

The release manager view

Good release management is a complex art, balancing the numerous trade-offs implied by the classic management triangle: Scope/Resources/Time. A great deal of the success enjoyed by teams using Agile Development Methodologies comes from using smaller iterations more frequently and the adaptive planning [15] that occurs as a consequence. Even with the best adaptive planning and continuous integration practices though, centralised VCS trunk quality 'dips' during the course of each iteration and a scramble often occurs towards the end of the iteration to restore quality to the previous level or better.

In an ideal world, a Release Manager would built each new version by picking and choosing 'lego brick' size blocks of functionality – features, improvements and bug fixes. If a feature doesn't meet user requirements or quality standards during testing, it should be possible to truly drop it as an atomic change (not just leave the code largely in place and hide access to it). Following the Defer Commitment practice of Lean Software Development [16], the Release Manager could defer decisions to the last possible moment, not just the start of each iteration.

Given the inherent complexity of software, this nirvana will sadly never arrive. However, distributed VCS brings us closer to it than ever before provided developers work in feature branches [17] either as individuals or in small groups. While this is technically possible using massive amounts of branches in a centralised VCS tool, distributed VCS tools make it practical thanks to their intelligent merging and their support for pulling changes just as easily as pushing them.

Viewed from this perspective, distributed VCS tools become a natural progression of the state of the practice. Subversion improved on CVS by making tree-wide commits atomic while Bazaar (for example) improves on Subversion by making 'feature-wide commits' (i.e. feature branch merges) atomic.

The community view

Every innovation starts with a problem that existing technology doesn't successfully solve. Mark Shuttleworth has explained that his driving motivation [18] for developing DVCS technology is because of its positive impact on open source communities:

> Distributed version control is all about empowering your community, and the people who might join your community. You want newcomers to get stuck in and make the changes they think make sense. It's the difference between having blessed editors for an encyclopedia (in the source code sense we call them 'committers') and the wiki approach.

Beyond lowering the barrier to entry, there are other notable community benefits:

- an easier migration path from non-core to core contributor
- as the community grows from 1 to 10s to 100s to 1000s, different workflows can be adopted without needing to change the VCS toolset
- branch management scales better than patch tracking.

The last point is important for teams or people who want or need to keep a list of patches and reapply them when a new upstream version is released. While tools like Patchwork Quilt [19] are useful, it is typically much easier to do that particular dance using a DVCS and either:

- having your own branch with patches applied and merging the new work, or
- reapplying bundles, i.e. patches with the full intelligent metadata available.

A more detailed analysis of this issue has been provided by John Arbash Meinel. [20]

The communities we join, whether off-line or on-line, say a lot about the sort of people we are. The flip side to this is that the tools we offer, when starting a community, implicitly say a lot about the sort of people we want to sign up.

The senior management view

There are many complex issues and few easy answers facing senior IT managers in most large corporations. Many IT best practices (outsourcing, offshoring, agile) are inherently in conflict and the risks – wasting money solving the wrong problems, quality, unpredictable exchange rates, skill shortages in development centre locations, etc. – can be high. As the open source movement has shown with stunning results, effective distributed collaboration is possible given the right people, the Internet and the right toolset. New ways of building companies and teams are now a reality and distributed VCS is one of the important technologies required. To quote Brian Aker, Director of Architecture at MySQL AB [21]:

> We have been using a distributed source control system since 2000. Our development process, which allows us to span multiple countries since all developers work from home, wouldn't work without it.

Of course, employing the best people possible regardless of their location is nothing new. The change is that the areas where that is viewed as a best practice will expand from consulting to engineering and beyond.

Problems to be aware of

At this point in time (Q4 2007), assuming every feature you need is there and rock solid in a given DVCS tool would be unwise. The maturity of all DVCS tools has a way to go in comparison to industry standards such as Subversion. As a rule, quirks and rough edges remain in both the tools and their documentation. 3rd party support in terms of integration into other tools is often beta quality. Text books and training courses are generally not yet available.

It must be stressed though that the leading tools are clearly good enough for a huge number of projects to adopt today. Interestingly, many of the tools are indeed much better in many regards than more established tools

in ways users may not initially expect. For example, all the leading DVCS tools are incredibly good at efficient project history storage.

Maturity and 3rd party support will undoubtedly remain a tactical adoption issue for a fair number of projects in the short term. Given their rapid rate of development, I expect this issue to largely disappear over the course of 2008. The strategic debate will then most likely pick up steam.

The arguments against distributed VCS

Greg Hudson has argued that the distributed VCS approach and the pyramid development model used by the Linux kernel team has numerous limitations and that a central repository with multiple committers would work better. [22] Linus has rejected this arguing that the approach implements the practical reality of developers using a 'network of trust' to get things done. I agree that the centralised workflow model has far more merit than Linus gives it credit for but his network of trust argument is rock solid. Indeed, most businesses and all teams I've ever seen unofficially operate on trust, regardless of the official management and technical hierarchies.

most businesses and all teams I've ever seen unofficially operate on trust, regardless of the official management and technical hierarchies

Ian Bicking has written about the potential downside of the distributed approach. [23] To summarise his concern, sharing early and often (as the central approach effectively mandates) encourages a better dynamic, particularly in the open source world. A rebuttal of this has been given by Bryan O'Sullivan. [24] I would further argue that (team-wide) publishing of changes once complete actually increases the quality of communication within a team. For example, receiving a diff about a completed logical unit of work is far more valuable than lots of smaller diffs (that may not make sense in isolation and a higher percentage of which are likely to be going down an incorrect path). Free services such as Launchpad [25] also help by making it easy to publish new branches to a central registry.

Havoc Pennington [26] has argued against the broader applicability of the distributed approach:

> I think what I don't get yet is why you'd want to maintain a bunch of local changesets for very long. The Linux-kernel-style fork-fest seems just nuts for anything I'm working on.

He has also raised the issue of generally poor usability:

> The distributed systems seem pretty wild from a user experience standpoint. In the sense of 'jeez, I can tell a (kernel, Haskell, shell) programmer wrote this'. Subversion may be less flexible but it's also less confusing.

These concerns should not be dismissed lightly. For distributed VCS to gain industry wide adoption, it needs to embrace and extend the central approach the vast majority of teams are comfortable with today. As I have argued previously, the future of version control is neither central nor distributed – it's adaptive [27]. There are compelling advantages to distributed VCS technology but many users and teams will adopt it for incremental improvements initially, looking for flexibility down the track – not a whole new way of working on day zero.

Selecting a distributed VCS tool

Of the numerous tools available, a small number will reach a critical mass of acceptance. In my opinion, the most likely candidates are Bazaar, Git and Mercurial. A sample of the projects adopting these three tools are Ubuntu, the Linux kernel and OpenJDK respectively. Popularity aside, when selecting a tool, the criteria to consider ought to include reliability [28], adaptability, usability [29], extensibility [30], integration [31] and low administration. I have previously published a series of articles covering these criteria in more depth.

The case for Bazaar

While recognising Git and Mercurial for the amazing tools that they are, I believe Bazaar will be the right strategic choice for a majority of teams in the future, if not already. (Disclaimer: I work for Canonical, the company sponsoring Bazaar, and joined the development team in early 2007 accordingly.) Bazaar has a high focus on:

- **usability** – being a tool developers love to use (i.e. having an interface that exposes just enough detail, having a straightforward command set, trying hard not to unpleasantly surprise users)
- **flexibility** – enabling whatever workflow makes sense across the collaboration spectrum: from personal productivity tool to communities with thousands of users
- **correctness** – doing the right thing (e.g. comprehensive rename support so that merging just works as frequently as possible)
- **quality** – Bazaar has a test suite with around 8500 tests (and growing)
- **cross platform support** (though Mercurial is also good in this regard).

For these reasons among others, I believe Bazaar is, and will most likely remain, easier for many teams to migrate to than the alternatives. In the open source world, Bazaar has the added benefit of free hosting on Launchpad and strong integration with the collaboration tools it provides. For example, Launchpad makes it easy to register branches, find branches and associate branches with product releases and bug fixes. In the commercial world, Bazaar provides the easiest upgrade path for teams largely happy with centralised workflow and commercial support is available if required.

To date, the primary issue with Bazaar has been performance on large code bases, particularly across high latency networks. This issue has been largely addressed by the introduction in 0.92 of a new format known as packs. As well as greatly improving performance on some key use cases, packs are important to Bazaar's evolution because they enable some interesting new features under development such as history horizons. With a strong and growing community actively developing it using numerous best practices and a productive programming language (Python), it is common for each (monthly) release of Bazaar to include 50–100 enhancements. It's an exciting community to be a part of and we are always looking for new members. (If Bazaar sounds exciting to you, please consider helping us to change the world.)

Distributed VCS is a new field and, as a rule, best practices are still evolving

Using distributed VCS effectively

Distributed VCS is a new field and, as a rule, best practices are still evolving. Existing software engineering texts rarely mention DVCS and tool-independent books on the topic may well take another year or two to surface. With these facts in mind, the best source of information of how best to use the technology is arguably the open source communities: the IRC channels and mailing lists for the various tools.

Some general recommendations are:

- Make sure the location of the primary development trunk is well publicised
- Use a program such as Robert Collins' PQM [32] together with an automated regression suite to ensure its quality is sacred

- Use a program such as Aaron Bentley's BB [33] to track submitted patches and their review status.

Distribution increases flexibility but a sensible amount of central data management can greatly aid effective collaboration. For example, Launchpad provides a central nexus where people can find information about open source projects, distributed team members, find branches, browse the code and so on. Alternatively, tracking important branches can be done using whatever technology makes sense, e.g. Wiki pages, a CMS, database or shared drive in a commercial environment.

PQM (Patch Queue Manager) automates commits to a branch by enforcing a test suite on the result of automatic merges. This is important because, as a project grows, the time to run a thorough automatic test suite grows as well. Development either slows down or developers stop running the tests on every single commit. The latter option is the road back to poor quality where regressions are only found after they've been in the main branch for some time.

BB (Bundle Buggy) retains an action queue of patches still needing review. It is an excellent aid to the typical open source process of people posting patches to a mailing list. Too often these patches are dropped, or become an interruption to the reviewers' work if they must be handled immediately. Working with Bazaar, BB keeps track of patches that have been merged or superseded, and it accumulates comments and decisions from reviewers. To put the importance of this in perspective, here's a simplistic summary of the key difference between commercial and open source project management:

- Commercial: How fast can we plant?
- Open source: How fast can we harvest?

BB is Bazaar's harvesting dashboard of choice.

Conclusions

The new breed of distributed version control tools are exciting because they change the software engineering game: they enable new ways of collaborating and that in turn enables new ways of thinking about and executing software development. Given its inherent distributed and loosely coupled nature, the open source community has much to gain from DVCS technology. As distributed development and open source practices gain popularity in commercial teams, the technology will become increasingly valuable there as well.

Adoption of DVCS technology will be piecemeal in the short term as the tools and 3rd party support matures. Nevertheless, DVCS is here to stay as the process flexibility it offers and higher quality software development it enables are compelling advantages. In particular, DVCS makes development in feature branches practical, maximising the rate at which teams can deliver value to users – a key tenet of Lean Software Development. By 2011–2012, I predict this technology will be widely adopted and many teams will wonder how they once managed without it.

Of the large number of available DVCS tools, three stand out as likely to gain a critical mass of acceptance: Bazaar, Git and Mercurial. With a high focus on doing the right thing, usability, workflow flexibility and cross platform support, Bazaar is a safe choice for many teams looking to make the most of this technology. ■

Acknowledgements

Thanks to Elliot Murphy and Martin Pool for reviewing this paper.

References

1 Arch home page, http://www.gnu.org/software/gnu-arch/
2 Bazaar home page, http://bazaar-vcs.org/
3 BitKeeper home page, http://www.bitkeeper.com/
4 Darcs home page, http://darcs.net/
5 Git home page, http://git.or.cz/
6 Mercurial home page, http://www.selenic.com/mercurial/wiki/
7 Monotone home page, http://monotone.ca/
8 SVK home page, http://svk.bestpractical.com/
9 Fred Brooks, 'No Silver Bullet: Essence and Accidents of Software Engineering', *IEEE Computer*, Apr 87, http://en.wikipedia.org/wiki/No_Silver_Bullet
10 Ian Clatworthy, 'Collaboration Redefined', http://ianclatworthy.wordpress.com/2007/06/08/collaboration-redefined/
11 Mark Shuttleworth, 'Merging is the key to software developer collaboration', http://www.markshuttleworth.com/archives/126
12 Linus Torvalds, 'Google tech talk: Linus Torvalds on git', http://www.youtube.com/watch?v=4XpnKHJAok8
13 Wikipedia, 'Distributed revision control', http://en.wikipedia.org/wiki/Revision_control#Distributed_revision_control
14 Mark Reinhold, 'Source-code management for an open JDK', http://blogs.sun.com/mr/entry/openjdk_scm
15 Craig Larman, A*gile and Iterative Development: A Manager's Guide*, Chapter 2 – Iterative & Evolutionary, http://safari.oreilly.com/0131111558/ch02
16 Lean Software Development home page, http://www.poppendieck.com/
17 Steve Berczuk & Brad Appleton, *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, Chapter 19 – Task Branch, http://www.scmpatterns.com/book/
18 Mark Shuttleworth, 'Renaming is the killer app of distributed version control', http://www.markshuttleworth.com/archives/123
19 Patchwork Quilt home page, http://savannah.nongnu.org/projects/quilt/
20 John Arbash Meinel, 'Thoughts on branch tracking vs patch tracking', https://lists.ubuntu.com/archives/bazaar/2007q4/032519.html
21 Brian Akers, response to 'Distributed Source Code Management – Niche or Trend?: The Q&A' by Stephen O'Grady, http://redmonk.com/sograd y/2007/06/26/dscm/
22 Greg Hudson, 'Why BitKeeper Isn't Right For Free Software', http://web.mit.edu/ghudson/thoughts/bitkeeper.whynot
23 Ian Bicking, 'Distributed vs Centralized Version Control', http://blog.ianbicking.org/distributed-vs-centralized-scm.html
24 Bryan O'Sullivan, 'On distributed and centralised revision control', http://www.serpentine.com/blog/2005/08/10/on-distributed-and-centralised-revision-control/
25 Launchpad home page, https://launchpad.net/
26 Havoc Pennington, 'Source control', http://log.ometer.com/2006-10.html
27 Ian Clatworthy, 'Version Control: The Future is Adaptive', http://ianclatworthy.wordpress.com/2007/06/21/version-control-the-future-is-adaptive/
28 Ian Clatworthy, 'Wanted: Rock Solid Version Control', http://ianclatworthy.wordpress.com/2007/07/11/wanted-rock-solid-version-control/
29 Ian Clatworthy, 'It Takes a Community to Raise Great Software', http://ianclatworthy.wordpress.com/2007/07/02/it-takes-a-community-to-raise-great-software/
30 Ian Clatworthy, 'Version Control: Plugins vs Toolkits', http://ianclatworthy.wordpress.com/2007/07/18/version-control-plug-ins-vs-toolkits/
31 Ian Clatworthy, 'Version Control: Design for Integration', http://ianclatworthy.wordpress.com/2007/07/30/version-control-design-for-integration/
32 PQM home page, https://launchpad.net/pqm
33 Bundle Buggy home page, http://bundlebuggy.aaronbentley.com/

Continuous Integration with CruiseControl.Net

Paul Grenyer creates a handy plug-in.

One of the first rules of writing is to write about something you know about. With the exception of the user guide for Aeryn [1] and Elephant [2] I have never done this. I have always written about new techniques, operating systems and ideas that I have been exploring as the article develops. The article has been a key player in the learning process for me.

My articles about CruiseControl.Net [3][4] are no different. I'd only been using CruiseControl.Net on and off for about a month before I wrote the first article. I was no expert then and I'm still learning now.

So what's the main thing I have learnt? CruiseControl.Net [5] has the potential to be a great application, but it's early days. There are a number of missing features, such as support for makefiles, sufficient error reporting from NAnt [6] and, the feature I see asked for the most in the forums, a setting to remove and recreate the source code directory prior to version control checkout. This is all pretty basic stuff and I am at a loss as to why the writers of CruiseControl.Net have not added these features.

The CruiseControl.Net development team do have a patch submission procedure, but I need something that will work now and I don't want to use a non-standard build in the mean time. Luckily CruiseControl.Net does have a plug-in mechanism. The main missing feature, the ability to remove the source code directory prior to checkout, can be addressed by writing a plug-in.

In this article I'll describe how to create and install a plug-in for CruiseControl.Net, a couple of different ways to setup the appropriate Visual Studio project and write a new task block.

CruiseControl.Net plug-ins

The limited instructions provided on the CruiseControl.Net website states the following:

1. Create a Class Library project to build the assembly that will contain your custom builder plug-in. The assembly that it produces should be named: `ccnet.*.plugin.dll` (where the star represents the name you choose).
2. Add your new custom builder class.
3. The class (see Listing 1 for an example) must implement the ThoughtWorks.CruiseControl.Core.ITask interface (found in the ccnet.core assembly).
4. Mark your class with the NetReflector **`ReflectorType`** attribute. The name argument supplied to the attribute is the name of the element/attribute that will appear in the configuration file.
5. Add any configuration properties that you need, marking them with the NetReflector **`ReflectorProperty`** attributes accordingly. Note that the attribute names are case sensitive and must match exactly in the configuration.
6. Implement the **`Run`** method. The supplied **`IntegrationResult`** should provide you with everything that you need to know about the current build.
7. Compile the assembly.
8. Copy the assembly into the folder containing the CruiseControl.NET assemblies (or the current directory that you are running the ccnet server from).

Listing 1

```
using System;
using Exortech.NetReflector;
using ThoughtWorks.CruiseControl.Core;

namespace
ThoughtWorks.CruiseControl.Sample.Builder
{
  [ReflectorType("mybuilder")]
  public class NAntBuilder : ITask
  {
    public void Run(IntegrationResult result)
    {
      Console.WriteLine("Hello World!");
    }
  }
}
```

9. Modify your `ccnet.config` file in accordance with the sample config file shown in Listing 2.

Although the instructions are limited this is pretty much all there is to it. The only other details missing are how to setup a project and what the dependencies are. This is what I'll be covering next.

Setting up a Visual Studio project

This is an article about CruiseControl.Net and as Visual Studio is well understood by most Windows developers, I am not going to go into the details of creating the project. Instead I am going to concentrate more on the structure and dependencies of the project.

.Net versions

When I started this article in May 2007 the current version of CruiseControl.net was 1.2.1.7 and used .Net 1.1. In the middle of June 2007 CruiseControl.Net 1.3, the first version to use .Net 2.0, was released. Plug-ins written in .Net 1.1 can, of course, be used by both versions but plug-ins written in .Net 2.0 can only be used with version 1.3+. Equally, Microsoft Visual Studio 7.x can be used to write plug-ins for either version,

Listing 2

```
<cruisecontrol>
  <project name="myproject">
    <tasks>
      <mybuilder>
        <!-- include custom builder properties
          here -->
      </mybuilder>
    </tasks>
  </project>
</cruisecontrol>
```

PAUL GRENYER

Paul has been a member of the ACCU since 2000. He founded the ACCU Mentored Developers and serves on the committee. Paul now contracts at an investment bank in Canary Wharf.

but Microsoft Visual Studio 8.x can only be used to write plug-ins for version 1.3+.

Dependencies

CrusieControl.Net plug-ins are inherently dependent on the following assemblies:

- `ThoughtWorks.CruiseControl.Core.dll`

  This is the main CruiseControl.Net assembly and contains the classes and interfaces that make up the source control, task and publisher blocks.

- `NetReflector.dll`

  .Net reflector [7] provides the mechanism that CruiseControl.Net uses to take the XML parameters from ccnet.config and insert them into block class properties. It is a third party assembly, but supplied with CruiseControl.Net.

- `log4net.dll`

  Log4Net [8] is a well known third party logging library for .Net and this is exactly what CruiseControl.Net uses it for.

- `ThoughtWorks.CruiseControl.Remote.dll`

  The CruiseControl.Net remote assembly is not strictly a plug-in dependency. However, if the CruiseControl.Net core Visual Studio project is included in a solution the remote project must also be included as the core project references it.

Simple solution

The easiest way to create a solution for a CrusieControl.Net plug-in is as follows:

1. Create an appropriately named C# assembly project and solution with Visual Studio. The name of the assembly must be in the form `ccnet.*.plugin.dll` where the star represents the name of the plug-in.
2. Create a `lib` folder and copy into it the assemblies that CrusieControl.Net is dependent on. Make sure you get the right versions of the assemblies for the version of .Net you are using. Both the release and source code zips of CruiseControl.Net contain all the required assemblies, and they are also located in the subversion repository.
3. Add the dependencies as references to the project.
4. Check the solution, the `lib` folder and its contents into a source control system.

Full solution

CruiseControl.Net is an evolving project, so plug-ins should not be fixed to a completely static version and ideally space shouldn't be taken up in the source control system with pre-built assemblies.

CruiseControl.Net is one of the best organised projects I've seen. It uses a subversion [9] repository and releases are copied to the tags directory as recommended by the creators of subversion. Therefore if a plug-in project is also using subversion, a link can be created so that the appropriate CruiseControl.Net release tag is checked out as part of the plug-in working copy (see sidebar). This means that any patches can be picked up easily, space isn't taken up in the plug-in repository with the CruiseContol.Net code and when the next release comes out the tag link can be easily moved or added. If subversion is not being used the source from the appropriate CruiseControl.Net source code zip can be checked into the plug-in repository instead and used as a stable reference.

Assuming that either a link to a CruiseControl.Net release has been created or the source code checked into the plug-in repository the following steps should be taken to setup a CruiseControl.Net plug-in solution in Visual Studio:

1. Create an appropriately named C# assembly project and solution with Visual Studio. The name of the assembly must be in the form

Subversion Links

The recommended way to organise a subversion repository is into three directories:

```
branches
tags
trunk
```

The main development goes on in trunk, any specialist branches are developed in branches and releases are copied to tags.

CruiseControl.Net follows this recommendation. The final .Net 1.1 release is 1.2.1.7 and is located here:

http://ccnet.svn.sourceforge.net/svnroot/ccnet/tags/1.2.1.7

The current release (at time of writing) is 1.3.0.2918, uses .Net 2.0 and is located here:

http://ccnet.svn.sourceforge.net/svnroot/ccnet/tags/1.3.0.2918

Following the subversion recommendation a CruiseControl.Net plug-in should have the directory structure above and all development should be done in trunk. CruiseControl.Net plug-ins are dependent on the CrusieControl.Net assemblies and its third party assemblies. These can be committed to the repository or a link to the CrusieControl.Net release source can be created and the assemblies built from the source directory. The source code isn't actually copied into the plug-in repository, it is only checked out at the same time.

A good place to link the CruiseControl.Net source code is:

trunk/thirdparty/ccnet/

with a further subdirectory for .Net 1.1 version and the .Net 2.0 version:

trunk/thirdparty/ccnet/1.2.1.7

trunk/thirdparty/ccnet/1.3.0.2918

This enables you to build your plug-in for either .Net version just by pointing your build system at the appropriate sources.

There is no need to create the thirdparty directory or the ccnet directory. The version directories must not be created or a locked error will be given when the linked code is checked out. To create the link:

1 Open a command prompt and move to the trunk directory.

2 On Windows set the default text editor for Subversion:

```
set SVN_EDITOR=notepad
```

3 Execute the following command:

```
svn propedit svn:externals .
```

The full stop at the end is very important as it specifies that the property (see subversion manual for more details) is set on the current (trunk) directory. If the plug-in goes into continuous integration the chances are that only the trunk directory from the plug-in workspace will be checked out so the property must go there or the CruiseControl.Net source won't get checked out with it.

4 Enter the following into the default text editor instance that is launched and then save and close it:

```
thirdparty/ccnet/1.2.1.7 http://ccnet.svn.
sourceforge.net/svnroot/ccnet/tags/1.2.1.7
thirdparty/ccnet/1.3.0.2918 http://ccnet.svn.
sourceforge.net/svnroot/ccnet/tags/1.3.0.2918
```

5 Execute **`svn update`** to checkout the CruiseControl.Net source.

6 Execute **`svn ci -m"<suitable commit message>"`** to commit the property change to the repository.

The property can be checked by executing **`svn propget svn:externals`** and should give the following output:

```
thirdparty/ccnet/1.2.1.7 http://ccnet.svn.
sourceforge.net/svnroot/ccnet/tags/1.2.1.7
thirdparty/ccnet/1.3.0.2918 http://ccnet.svn.
sourceforge.net/svnroot/ccnet/tags/1.3.0.2918
```

If necessary the link can be removed by executing: **`svn propdel svn:externals`**.

Be very careful not to cancel a subversion command while setting a property or checking code out as this can cause repository locks that are very difficult to remove.

`ccnet.*.plugin.dll` where the star represents the name of the plug-in.

2. Create a second assembly project for unit tests called `ccnet.*.plugin.test.dll` where the star represents the name of the plug-in.
3. Add `core.csproj` from `<ccnet source>/project/core` and `Remote.csproj` from `<ccnet source>/project/Remote` to the solution.

   Make sure you choose the CruiseControl.Net source directory that goes with version of .Net and Visual Studio that you are using:

.Net	CruiseControl.Net	Visual Studio
1.1	1.2.1.7	7.x
2	1.3.0.2918	8.0

4. Add core as a project reference and `NetReflector.dll` from `<ccnet source>/lib` as an assembly reference to the `ccnet.*.plugin` project.
5. Add `ccnet.*.plugin` and core as a project reference and `NetReflector.dll` from `<ccnet source>/lib` as an assembly reference to the `ccnet.*.plugin.test` project.
6. Add `nunit.framework.dll` from `<ccnet source>/tools/nunit` as an assembly reference to the `ccnet.*.plugin.test` project.

   The nunit.framework assembly from the CruiseControl.Net source should be used to ensure you have an assembly that uses the correct version of .Net.
7. Create an empty directory called `xls` at the solution root.

   The core project has a post build step that copies some xls files from the root of the CruiseControl.Net solution to the output directory. It uses a Visual Studio environment variable to determine the root of the solution. Building the project from another solution confuses it. Without the xls directory the project will fail to build due to failure of its post build step. It's a hack, but it solves the problem.

The solution will now build. The CruiseControl.Net source code, unfortunately, generates a number of warnings. You might also want to create a NAnt script. This is useful if you're using, for example, Mono [10] or something other than Visual Studio. NAnt is also useful if you want to build your plug-in for multiple versions of .Net.

Getting the plug-in into Continuous Integration

This is an article series primarily about Continuous Integration, so naturally the first thing to do is to place the plug-in under Continuous Integration. In Part I of this series I described how to create a CruiseControl.Net configuration file with a project block, source control block, tasks blocks and an email publisher block. The project structure described above also requires an NUnit block.

NUnit task block

All software projects should be unit tested. CruiseControl.Net is one of the best and most completely unit tested projects I've seen. Any plug-in should also be unit tested. The solution creation described above includes the creation of a project for unit tests. Running unit tests as part of the build is an important part of Continuous Integration. CrusieControl.Net is intended primarily (but not exclusively) for .Net projects, therefore it has a block that supports the most well know and popular .Net testing framework: NUnit [11].

The only required NUnit task block parameter is a list of assemblies to run. However, just specifying assemblies assumes that the nunit-console.exe executable, which is used to run the tests and to generate the output xml file containing the results, is in the path. There are a few different versions of NUnit. To make matters more complicated there are separate .Net 1.1 and .Net 2.0 versions and you cannot have them both installed at the same time. To ensure the correct version of NUnit is used it should also be checked into the repository. CruiseControl.Net includes NUnit in its repository and it can be accessed as shown in Listing 3.

Listing 3

```
<nunit>
 <path>thirdparty\ccnet\1.3.0.2918\tools\nunit\
       nunit-console.exe</path>

<assemblies>
 <assembly>ccnet.marauder.plugin.tests\bin\Debug\
           ccnet.marauder.plugin.tests.dll
           </assembly>
 <assembly>ccnet.marauder.plugin.tests\bin\
          Release\ccnet.marauder.plugin.tests.dll
          </assembly>
 <assembly>build\net-1.1\ccnet.marauder.plugin.
           tests.dll</assembly>
 <assembly>build\net-2.0\ccnet.marauder.plugin.
           tests.dll</assembly>
</assemblies>
</nunit>
```

Bear in mind that assemblies created with both .Net 1.1 and .Net 2.0 can be run using the .Net 2 version of NUnit, but assemblies built with .Net 2.0 can only be run using the .Net 2.0 version.

There are also parameters for setting the name of the xml output file and the timeout. The result of running the tests can be viewed using the CruiseControl.Net Dashboard via the NUnit Details link in the Build view and are included in the email if an email publisher block is specified.

The complete CruiseControl.Net configuration for CCMarauder is shown in Listing 4 (continues overleaf).

Adding a source control task to the plug-in

Now that the infrastructure is in place, the development of the plug-in itself can begin.

The layout of a CruiseControl.Net plug-in ought to match the layout of the CruiseControl.Net project, for consistency if nothing else. Create a sourcecontrol folder in the `ccnet.marauder.plugin` project in Visual Studio (this will create a corresponding directory on the disk) and create

Listing 4

```
<cruisecontrol>
  <project name="CCMaruader">
    <workingDirectory>c:\temp\ccnet\ccmarauder\
        </workingDirectory>
    <artifactDirectory>c:\temp\ccnet\ccmarauder
        </artifactDirectory>

    <sourcecontrol type="svn">
      <trunkUrl>http://ccmarauder.tigris.org/svn/
         ccmarauder/trunk/</trunkUrl>
      <workingDirectory>c:\temp\ccnet\ccmarauder
         </workingDirectory>
      <executable>C:\Program Files\CollabNet
         Subversion\csvn.exe</executable>
    </sourcecontrol>

    <tasks>
      <devenv>
        <solutionfile>ccnet.marauder.plugin.sln
           </solutionfile>
        <configuration>Debug</configuration>
        <executable>C:\Program Files\Microsoft
           Visual Studio .NET 2003\Common7\IDE\
           devenv.com</executable>
        <buildtype>Rebuild</buildtype>
        <buildTimeoutSeconds>300
           </buildTimeoutSeconds>
      </devenv>
```

Listing 4 (cont'd)

```
      <devenv>
        <solutionfile>ccnet.marauder.plugin.sln
                      </solutionfile>
        <configuration>Release</configuration>
        <executable>C:\Program Files\Microsoft
           Visual Studio .NET 2003\Common7\IDE\
           devenv.com</executable>
        <buildtype>Rebuild</buildtype>
        <buildTimeoutSeconds>300
           </buildTimeoutSeconds>
      </devenv>
      <nunit>
        <path>C:\Program Files\NUnit 2.4.1\bin\
           nunit-console.exe</path>
        <assemblies>
          <assembly>ccnet.marauder.plugin.test\
             bin\Debug\ccnet.marauder.plugin.
             test.dll</assembly>
          <assembly>ccnet.marauder.plugin.test\
             bin\Release\ccnet.marauder.plugin.
             test.dll</assembly>
          <assembly>build\ccnet.marauder.plugin
             .test.dll</assembly>
        </assemblies>
      </nunit>
    </tasks>
    <publishers>
      <xmllogger logDir="buildlogs" />
      <email from="paul.grenyer@gmail.com"
         mailhost="mailhost.zen.co.uk"
         includeDetails="TRUE">
        <users>
          <user name="Paul Grenyer"
             group="buildmaster"
             address="paul.grenyer@gmail.com"/>
          <user name="ccmarauder Developers"
             group="developers"
             address="continuousintegration
             @ccmarauder.tigris.org"/>
        </users>
        <groups>
          <group name="developers"
              notification="change"/>
          <group name="buildmaster"
              notification="always"/>
        </groups>
      </email>
    </publishers>
  </project>
</cruisecontrol>
```

an `Svn.cs` file in it. Check the file and folder into your version control system.

CruiseControl.Net uses Net Reflector [7] to allow its blocks to be dynamically instantiated from the XML in its configuration file and have the task's properties initialised with the values specified in the XML. Therefore a Exortech.NetReflector using directive is required.

Usually, custom tasks implement the ITask interface. However, we want to add functionality to an existing task. The ability to remove the source directory is missing from all CruiseControl.Net source control tasks. I am going to describe how to add it to the Subversion task and therefore will inherit the new task from that. The task also needs a name to identify it in the CruiseControl.Net configuration file. **`marauder-svn`** is sufficiently descriptive and unique. The name of the task is specified using the **`ReflectorType`** attribute. The basic class declaration looks like Listing 5.

Listing 5

```
using Exortech.NetReflector;
namespace ccnet.marauder.plugin.sourcecontrol
{
  [ReflectorType("marauder-svn")]
  class Svn : ThoughtWorks.CruiseControl.Core.
     Sourcecontrol.Svn
    {
      // ...
    }
}
```

Listing 6

```
[ReflectorType("marauder-svn")]
class Svn :
ThoughtWorks.CruiseControl.Core.Sourcecontrol.Svn
{
  private bool removeWorkingDir = false;
  [ReflectorProperty("removeWorkingDirectory",
                     Required=false)]
  public bool RemoveWorkingDir
  {
    get { return removeWorkingDir; }
    set { removeWorkingDir = value; }
  }
  // …
}
```

Not everyone will want the new functionality so it needs to be easily enabled and disabled. The easiest way to do this is with a property based flag that is initialised from the configuration file. The **`ReflectorProperty`** attribute is used for this (Listing 6).

At this stage it's helpful and useful to add a test. Create a sourcecontrol folder in the `ccnet.marauder.plugin.tests` project in Visual Studio (this will create a corresponding directory on the disk) and create an `SvnTest.cs` file in it. Check the file and folder into your version control system.

All CruiseControl.Net blocks have tests that check that settings defined by the XML in the configuration file are correctly initialised in the blocks' properties. A number of using directives are needed to write the test:

- **`using ccnet.marauder.plugin.sourcecontrol`**
  Defines marauder-svn.
- **`using ThoughtWorks.CruiseControl.Core.Util`**
  Defines CruiseControl.Net utility classes such as Time.
- **`using Exortech.NetReflector`**
  Defines classes to take XML configuration and inserts it into block class properties.
- **`using NUnit.Framework`**
  Defines classes and attributres for the NUnit framework.

The test fixture and test function are defined as in Listing 7.

Listing 7

```
using ccnet.marauder.plugin.sourcecontrol;
  using ThoughtWorks.CruiseControl.Core.Util;
  using Exortech.NetReflector;
  using NUnit.Framework;
  namespace
     ccnet.marauder.plugin.tests.sourcecontrol
{
  [TestFixture]
  public class SvnTest
  {
    [Test]
    public void PopulateFromFullySpecifiedXml()
    {
      // ...
    }
}
```

Listing 8

```
[Test]
public void PopulateFromFullySpecifiedXml()
{
  string xml = @"
    <marauder-svn>
      <executable>c:\svn\svn.exe</executable>
      <trunkUrl>svn://myserver/mypath</trunkUrl>
      <timeout>5</timeout>
      <workingDirectory>c:\dev\src
          </workingDirectory>
      <username>user</username>
      <password>password</password>
      <tagOnSuccess>true</tagOnSuccess>
      <tagBaseUrl>svn://myserver/mypath/tags
          </tagBaseUrl>
      <autoGetSource>true</autoGetSource>
    </marauder-svn>";

  Svn svn = (Svn) NetReflector.Read(xml);
  Assert.AreEqual(@"c:\svn\svn.exe",
                  svn.Executable);
  Assert.AreEqual("svn://myserver/mypath",
                  svn.TrunkUrl);
  Assert.AreEqual(new Timeout(5), svn.Timeout);
  Assert.AreEqual(@"c:\dev\src",
                  svn.WorkingDirectory);
  Assert.AreEqual("user", svn.Username);
  Assert.AreEqual("password", svn.Password);
  Assert.AreEqual(true, svn.TagOnSuccess);
  Assert.AreEqual(true, svn.AutoGetSource);
  Assert.AreEqual("svn://myserver/mypath/tags",
                  svn.TagBaseUrl);
  Assert.AreEqual(true, svn.RemoveWorkingDir);
}
```

This is all that is required to create a test and, even though it does not actually test anything yet, if it is committed to source control, continuous integration will pick it up and run the test. Try it.

One thing worth testing is that the properties of the base class (`*.Core.Sourcecontrol.Svn`) are still populated correctly. The easiest way to do that is to copy the SVN CruiseControl.Net test into the plug-in test (Listing 8).

The code above is not an exact cut and paste from CruiseControl.Net test. The root XML tag has been modified so that it instantiates the Svn block from the plug-in, rather than the CruiseControl.Net block and a new test has been added to test the value of `RemoveWorkingDirectory`. The default value of `removeWorkingDirectory` is false, so currently the test will fail. Commit the file to source control to see this.

To fix this, a new element needs to be added to the XML. The xml string defined in the test function represents, although is not the same as, the XML used in the CruiseControl.Net configuration file. Listing 9 shows where the element must be added.

Listing 9

```
string xml = @"
  <marauder-svn>
    <executable>c:\svn\svn.exe</executable>
    <trunkUrl>svn://myserver/mypath</trunkUrl>
    <timeout>5</timeout>
    <workingDirectory>c:\dev\src
        </workingDirectory>
    <username>user</username>
    <password>password</password>
    <tagOnSuccess>true</tagOnSuccess>
    <tagBaseUrl>svn://myserver/mypath/tags
        </tagBaseUrl>
    <autoGetSource>true</autoGetSource>
    <removeWorkingDirectory>true
        </removeWorkingDirectory>
  </marauder-svn>";
```

Commit the modification to source control to see the tests pass.

CruiseControl.Net blocks also have tests for the minimum required XML:

```
[Test]
public void SpecifyFromMinimallySpecifiedXml()
{
  string xml = @"<marauder-svn/>";
  svn = (Svn) NetReflector.Read(xml);
  Assert.AreEqual("svn.exe", svn.Executable);
  Assert.AreEqual(false, svn.RemoveWorkingDir);
}
```

One test that is missing from CruiseControl.Net is the element present in the XML, but with a value of `false`. For completeness this should also be added to the plug-in tests:

```
[Test]
public void SpecifyFalseInXml()
{
  string xml = @"
    <marauder-svn>
      <removeWorkingDirectory>false
          </removeWorkingDirectory>
    </marauder-svn>";
  Svn svn = (Svn)NetReflector.Read(xml);
  Assert.AreEqual(false, svn.RemoveWorkingDir);
}
```

Implementing the task block

As shown in the sample builder class above, a task block is usually implemented by the ITask interface. The tasks that the task block performs go into the `Run` method. Our custom Subversion source control block does not implement ITask. It implements ISourceControl instead, which is the equivalent interface for source control blocks (Listing 10).

Listing 10

```
namespace ThoughtWorks.CruiseControl.Core
{
  [TypeConverter(
     typeof(ExpandableObjectConverter))]
  public interface ISourceControl
  {
    Modification[] GetModifications(
       IIntegrationResult from,
       IIntegrationResult to);
    void LabelSourceControl(
       IIntegrationResult result);
    void GetSource(IIntegrationResult result);
    void Initialize(IProject project);
    void Purge(IProject project);
  }
}
```

We need to modify the standard behaviour of the CruiseControl.Net Subversion source control block so that the working directory is removed before the source is checked out. Looking at the CruiseControl.net Subversion class reveals that the source is checked out in as shown in Listing 11.

Listing 11

```
public override void GetSource(
   IIntegrationResult result)
{
  if (! AutoGetSource) return;
  if (DoesSvnDirectoryExist(result)) {
    UpdateSource(result);}
  else {
    CheckoutSource(result);}
}
```

Listing 12

```
[ReflectorType("marauder-svn")]
public class Svn :
 ThoughtWorks.CruiseControl.Core.Sourcecontrol.Svn
{
  // ..
  public override void GetSource(
     IIntegrationResult result)
  {
    if (AutoGetSource)
    {
      // Add new functionality here.
      base.GetSource(result);
    }
  }
}
```

This method is virtual (this is indicated by the use of the **override** keyword and the lack of a compiler error), so all we need to do is override this method in our subclass, add our functionality and then call the above **GetSource** method. There is one further consideration. Looking at the above method closely reveals that if **AutoGetSource** is set to **false**, the method does nothing. Our overridden method needs to act in the same way (Listing 12).

Listing 13

```
using System.IO;
// …
public override void GetSource(
   IIntegrationResult result)
{
  if (AutoGetSource)
  {
    string workingDirectory =
       result.BaseFromWorkingDirectory(
       WorkingDirectory);
    if (RemoveWorkingDir)
    {
      Log.Info("Removing working directory.");
      DeleteDirectory(
                new DirectoryInfo(
                workingDirectory));
    }
    base.GetSource(result);
  }
}
private void DeleteDirectory(
   DirectoryInfo dirInfo)
{
  foreach (
     DirectoryInfo subDirInfo in
     dirInfo.GetDirectories())
  {
    DeleteDirectory(subDirInfo);
  }
  foreach (
     FileInfo fileInfo in dirInfo.GetFiles())
  {
    FileAttributes fileAttri =
       File.GetAttributes(
       fileInfo.FullName);
    if (
       (fileAttri & FileAttributes.ReadOnly) != 0)
    {
      File.SetAttributes(fileInfo.FullName,
         fileAttri & ~FileAttributes.ReadOnly);
    }
    Log.Debug(string.Format(
       "Removing: {0}",fileInfo.FullName));
    File.Delete(fileInfo.FullName);
  }
  Log.Debug(string.Format("Removing: {0}",
     dirInfo.FullName));
  Directory.Delete(dirInfo.FullName);
}
```

Before the working directory can be removed, we need to know what it is. Several CruiseControl.Net task blocks have a working directory property. Usually if this is an absolute path it is used as is. If it is a relative path then the project working directory is appended to the beginning. The CruiseControl.Net subversion source control block has an example of how to create the working directory:

```
string workingDirectory =
 result.BaseFromWorkingDirectory(WorkingDirectory);
```

WorkingDirectory is a property of the subclass. **result** is a reference to an object that implements the **IIntegrationResult** interface and gives access to all sorts of project settings, including the global working directory path, and is passed to all task blocks.

Once the working directory path has been ascertained, **RemoveWorkingDirectory** needs to be checked and if **true** the working directory removed (Listing 13).

The details of the DeleteDirectory method are beyond the scope of this article, but explained in detail in Visiting Files and Directories in C# [12]. Log is a log4net logger available in a base class.

As mentioned previously, CruiseControl.Net source control blocks do not create the source code directory if it does not exist. Adding this functionality is straight forward (Listing 14).

That completes the implementation of our custom Subversion source control block. The next step is to add more unit tests. However, this would involve:

- Applying an extract method to **GetSource** so that the new functionality can be called without calling the base class version of **GetSource**.
- Creating a class to call the **File** and **Directory** methods and extracting its interface so that a mock can be passed in when testing.

Listing 14

```
public override void GetSource(IIntegrationResult
result)
{
  if (AutoGetSource)
  {
    string workingDirectory =
       result.BaseFromWorkingDirectory(
       WorkingDirectory);
    if (RemoveWorkingDir)
    {
      Log.Info("Removing working directory.");
      DeleteDirectory(new DirectoryInfo(
         workingDirectory));
    }
    if (!DirectoryExists(workingDirectory))
    {
      Log.Info("Creating working directory.");
      CreateDirectory(workingDirectory);
    }
    base.GetSource(result);
  }
}
…
private bool DirectoryExists(string path)
{
  return Directory.Exists(path);
}
private void CreateDirectory(string path)
{
    Directory.CreateDirectory(path);
}
```

This is quite involved with a number of steps and outside the scope of this article.

Using the plug-in with CrusiseControl.Net

Finally the ultimate test: plugging the plug-in into a CruiseControl.Net server. To use, and test, the plug-in with CruiseControl.net:

1. Stop the service or command line running
2. Copy the plug-in assembly to the CruiseControl.Net service directory.
3. Edit the `ccnet.config` file (see below).
4. Restart the service or command line.

Adding a new task to the `ccnet.config` file is described at the beginning of this article. Using a source control task block is slightly different. The standard SVN block for the Aeryn project described in Part 1, is as follows:

```
<sourcecontrol type="svn">
  <trunkUrl>http://aeryn.tigris.org/svn/aeryn/
     trunk/</trunkUrl>
  <workingDirectory>c:\temp\ccnet\aeryn\working
     </workingDirectory>
  <executable>C:\Program Files\...\csvn.exe
     </executable>
</sourcecontrol>
```

It can be modified to use the plug-in like this:

```
<sourcecontrol type="marauder-svn">
  <trunkUrl>http://aeryn.tigris.org/svn/aeryn/
     trunk/</trunkUrl>
  <workingDirectory>c:\temp\ccnet\aeryn\working
     </workingDirectory>
  <executable>C:\Program Files\...\csvn.exe
     </executable>
  <removeWorkingDirectory>true
     </removeWorkingDirectory>
</sourcecontrol>
```

Changing the source control type to the attribute used to identify the plug-in SVN class tells CruiseControl.Net to load it instead of the standard SVN task. Adding the **`removeWorkingDirectory`** tags tells **`marauder-svn`** to invoke the new functionality. That's all there is to it.

There are a number of ways to test it. One easy one is to tail the CruiseControl.Net log (`ccnet.log` in the `service` directory), force a build and watch for the log messages:

```
[Aeryn:INFO] Building: Paul Grenyer triggered a
build (ForceBuild)
[Aeryn:INFO] Removing working directory.
[Aeryn:INFO] Creating working directory.
[Aeryn:INFO] Starting build: …
```

Another way to test is to open Windows Explorer (or equivalent) on the working directory and watch the source disappear and get rechecked out following a force a build.

Although the unit tests mean we should be confident that the new source control task works as expected, it is also worth checking what happens when **`removeWorkingDirectory`** is set to **`false`** and what happens when it is removed and the default value (**`false`**) is used. ■

Acknowledgments

Thank you to Jez Higgins for the pointers on svn propset and reviews. Thank you to Peter Hammond and Adrian Fagg for review and Caroline Hargreaves for proof reading.

References

[1] http://www.aeryn.co.uk/
[2] http://elephant.tigris.org/
[3] Integration with CruiseControl.Net – Part 1 http://www.marauder-consulting.co.uk/articles
[4] Integration with CruiseControl.Net – Part 2 http://www.marauder-consulting.co.uk/articles
[5] http://ccnet.thoughtworks.com/
[6] http://nant.sourceforge.net/
[7] http://sourceforge.net/projects/netreflector/
[8] http://logging.apache.org/log4net/
[9] http://subversion.tigris.org/
[10] http://www.mono-project.com/
[11] http://www.nunit.org/
[12] 'Visiting Files and Directories in C#' http://www.marauder-consulting.co.uk/articles

Embedding Lua into a C++ Application

Renato Forti demonstrates Lua with a tutorial.

Audience

This article was written for beginners. It introduces embedding and using Lua in C++ applications. You need have some proficiency in C++ and the Lua language to understand it.

If you are new to C++, you can read some introductory texts to become more familiar with the language. In the bibliography, I list some good books on C++. For Lua, see my last article or consult the bibliography for links and books.

Introduction

Lua is an embedded language, which means that your applications can incorporate Lua into them.

Your application needs some way to communicate with Lua, using what is known as the Lua C API. This is a set of functions that allow your C or C++ code to communicate with Lua and vice versa. The C API is divided into two parts:

- Lua core, which provides the primitive functions for all interactions between C and Lua,
- Lua library, which provides several convenient functions to interface C with Lua.

Stack

To pass values to the host program, Lua uses a virtual stack. Each element in this stack represents a Lua value: string, number and so on.

When Lua calls a C function, the called function gets a new independent stack. Each call creates a new stack. This stack initially contains any arguments to the C function and it is where the C function pushes its results to be returned to the caller. You will see this in action in the sample below.

Put Lua to work

Now we will start to construct a sample. You will need:

1. Lua – get it from http://www.lua.org/download.html
2. Build Lua in a library – see my previous article, or consult the bibliography.
3. A C++ compiler.

I use Visual Studio 2005 Command Prompt to compile and Notepad++ to edit source on Windows XP, but you can use your preferred compiler and OS. Lua runs on Windows, Unix, and others.

After you have all these, it is time to start.

Build Lua environment

The first thing is to include lua in your `main.cpp`, see Listing 1. This file is located in `<lua source>\etc`.

Lua needs to be initialized and, once you finish your work, closed down. The C API provides the following functions to do this:

- `lua_State* lua_open();`
- `void lua_close(lua_State* pLuaState);`

RENATO FORTI

Renato Forti is a C++ programmer who works at Vegas Card, a financial company located in Brazil. He can be contacted at re.tf@acm.org

Listing 1

```
// Lua support
#include <lua.hpp>
#include <iostream>
int main(int argc, char ** argv)
{
  return 0;
}
```

These functions are defined in `lua.h`, which is included, with others, in `lua.hpp`.

The **`lua_open()`** is actually a **`#define`** to **`luaL_newstate()`**, see:

```
#define lua_open() luaL_newstate()
```

The **`luaL_newstate()`** function create a new Lua environment, also known as state. This **`lua_State`** is a dynamic structure that you will need pass as an argument to all functions inside Lua.

An important question here is: What happens if I call **`lua_open`** more than once? A subsequent call of **`lua_open`** has no effect, so it is safe to call **`lua_open`** more than one time.

To keep things easy, I will use the RAII idiom here. (Listing 2 – `LuaEnvironment.hpp`)

Now is time to start, see Listing 3 (`main.cpp`).

Lua standard libraries

Lua provides many standard libraries. These libraries cover services like I/O, operating system facilities and so on:

- basic library;
- package library;
- string manipulation;

Listing 2

```
// Lua support
#include <lua.hpp>
#include <stdexcept>
class LuaEnvironment
{
  public:
    LuaEnvironment(): state_(lua_open())
    {
      if (!state_)
        throw std::runtime_error(
         "Lua environment failed to initialise");
    }

    ~LuaEnvironment()
    {
      lua_close(state_);
    }

    lua_State* getLuaState(void)
    {
      return state_;
    }

   private:
     lua_State* state_;
};
```

Listing 3

```
#include "LuaEnvironment.hpp"
#include <iostream>
int main(int argc, char ** argv)
{
  try
  {
    LuaEnvironment luaEnv;
    // Do things with lua here
  }
  catch(const std::runtime_error& runtimeError)
  {
    std::cout << runtimeError.what() <<
       std::endl;
    return 1;
  }
  return 0;
}
```

- table manipulation;
- mathematical functions (sin, log, etc.);
- input and output;
- operating system facilities;
- debug facilities.

luaopen_XXX is used to open a library, where **XXX** is the library name, for example consider:

```
 //for the mathematical library
luaopen_math(luaState);
  //for the I/O and the Operating
luaopen_io(luaState);
```

Or you can open all libraries with an single call to:

```
void luaL_openlibs(lua_State* L);
```

Call Lua functions

Now we will make one Lua function that returns the classic 'Hello world!' message to the caller. See Listing 4 – `sample.lua`.

Now some explanation of the Lua C API functions needed to call our **hw()**. The first is:

```
int luaL_dofile(lua_State* L,
                const char *filename);
```

luaL_dofile loads and compiles the given file. In our case **sample.lua** in second parameter, the first parameter is Lua State returned by **lua_open()**. If execute successfully it returns **0**.

To use a string rather than a file you can use:

```
int luaL_dostring(lua_State* L, const char *str);
```

Before we call a function, we need to push the function onto the stack. To do this we will use:

```
void lua_getglobal(lua_State* L,
                   const char *name);
```

Then we need to call our function:

```
int lua_pcall(lua_State* L, int nargs,
              int nresults, int errfunc);
```

Listing 4

```
-- very simple function

function hw()
  return "Hello world!"
end
```

And to get returned value:

```
const char *lua_tostring(lua_State* L,
                         int index);
```

Here we add all the functions to our wrapper class **LuaEnvironment** (Listing 5 – `LuaEnvironment.hpp`).

The sample code is shown in Listing 6 (`main.cpp`).

Listing 5

```
class LuaEnvironment
{
  public:
    LuaEnvironment()
      : state_(lua_open())
    {
      if (!state_)
        throw std::runtime_error(
         "Lua environment failed to initialise");
    }

    LuaEnvironment(const std::string& filename,
       bool openLibs = false)
      : state_(lua_open())
    {
      if (!state_)
        throw std::runtime_error(
         "Lua environment failed to initialise");
      if (openLibs)
        openStandardLibs();
      if (dofile(filename))
        throw std::runtime_error(
           "Load Lua File Error.");
    }

    ~LuaEnvironment()
    {
      lua_close(state_);
    }
    void openStandardLibs()
    {
      luaL_openlibs(state_);
    }
    lua_State* getLuaState(void)
    {
      return state_;
    }
    int dofile(const std::string& filename)
    {
      return luaL_dofile(
         state_, filename.c_str());
    }
    void setFuncName(
       const std::string& functionName)
    {
      lua_getglobal(state_, functionName.c_str());
    }
    int pcall(int nargs = 0, int nresults = 0,
       int errfunc = 0)
    {
      return lua_pcall(state_, nargs,
         nresults, errfunc);
    }
    std::string tostring(int index = -1)
    {
      return lua_tostring(state_, index);
    }

   private:
    lua_State* state_;
};
```

Listing 6

```
LuaEnvironment luaEnv("sample.lua", true); ❶
// Do things with lua here
luaEnv.setFuncName("hw"); ❷
luaEnv.pcall(0, 1); ❸
std::cout << luaEnv.tostring() << std::endl; ❹
```

Some explanation:

❶: Load `sample.lua` and open Lua standard libraries

❷: Define function to be called

❸: Call Lua function

❹: Get result as string.

A note about Stack Index:

Lua uses a LIFO (Last In, First Out) stack. In the code above you saw that we used **`-1`** in **`lua_tostring`** as the default index. What this means is that **`-1`** refers to the element at the top, the last element pushed, **`-2`** to the previous element, and so on: if we use **`1`** will we get the first element pushed, **`2`** to the next etc. Negative indices count down the stack from the top, positive indexes count up the stack from the bottom.

(top)

(3) **`lua_tostring(state_, 3);`** or **`lua_tostring(state_, -1);`**
(2) **`lua_tostring(state_, 2);`** or **`lua_tostring(state_, -2);`**
(1) **`lua_tostring(state_, 1);`** or **`lua_tostring(state_, -3);`**

Lua stack

Now we will improve our error handler with a class (Listing 7 – `LuaException.hpp`).

When an error occurs in Lua the message is always pushed on top of the stack, by **`luaL_error`**. Because of this we use **`lua_tostring(L, -1)`** to get the top message. Note that status must be greater than **`0`**. (See Listing 8 – `LuaEnvironment.hpp`, which uses **`LuaException`** and Listing 9 – `main.cpp`, which is the full code.

Note that I changed `sample.lua` to `@sample.lua` to generate an error. The output when I execute this code is:

```
C:\temp\lua-5.1.1\src>main
ERROR: cannot open @sample.lua: No such file or
directory
```

Now we will define a more complex function (`sample.lua`):

```
function more_complex(x, y)
  return x+y, x-y, x*y, x/y;
end
```

Listing 7

```
#include <lua.hpp>
#include <stdexcept>
#include <iostream>
#include <string>

class LuaException :
  public std::runtime_error
{
public:
  LuaException(const std::string& msg)
    : std::runtime_error(msg)
    {
    }
  // Lua error
  LuaException(lua_State*L, int status)
    : std::runtime_error(lua_tostring(L, -1))
    {
      lua_pop(L, 1);
    }
};
```

Listing 8

```
class LuaEnvironment
{
  public:
    LuaEnvironment()
      : state_(lua_open())
    {
      if (!state_)
        throw LuaException(
         "Lua environment failed to initialise");
    }
    LuaEnvironment(const std::string& filename,
                   bool openLibs = false)
      : state_(lua_open())
    {
      if (!state_)
        throw LuaException(
         "Lua environment failed to initialise");
      if (openLibs)
        openStandardLibs();
      dofile(filename);
    }
    ...
    void dofile(const std::string& filename)
    {
      int status = luaL_dofile(state_,
                               filename.c_str());
      if (status)
        throw LuaException(state_, status);
    }
    ...
    void pcall(int nargs = 0, int nresults = 0,
               int errfunc = 0)
    {
      int status = lua_pcall(state_, nargs,
                             nresults, errfunc);
      if (status)
        throw LuaException(state_, status);
    }
    ...
};
```

This function receives two parameters and has four results.

We will need some further Lua C API functions:

```
void lua_pushnumber(lua_State* L, lua_Number n);
```

This function pushes a number value on to the stack. This will be used as **`x, y`** argument to our function. The Lua API has **`push`** and **`lua_to`** functions for each C type that can be represented in Lua.

```
lua_Number lua_tonumber (lua_State*L, int index);
```

Listing 9

```
int main(int argc, char ** argv)
{
  try
  {
    LuaEnvironment luaEnv("@sample.lua", true);

    luaEnv.setFuncName("hw");
    luaEnv.pcall(0, 1);
    std::cout << luaEnv.tostring() << std::endl;

  }
  catch(const LuaException& luaException)
  {
    std::cout << luaException.what() <<
       std::endl;
    return 1;
  }
  return 0;
}
```

Listing 10

```
class LuaEnvironment
{
  public:
    ...
    std::string tostring(int index = -1,
                         int pop = 1)
    {
      std::string result = lua_tostring(state_,
                                        index);
      if (pop)
        lua_pop(state_, pop);
      return result;
    }
    lua_Number tonumber(int index = -1,
                        int pop = 1)
    {
      lua_Number result = lua_tonumber(state_,
                                       index);
      if (pop)
        lua_pop(state_, pop);
      return result;
    }
    void pushnumber(lua_Number number)
    {
      lua_pushnumber (state_, number);
    }
    ...
};
```

We use **lua_tonumber** to get value of stack:

```
void lua_pop (lua_State*L, int n);
```

We use **lua_pop** to pop one or more elements from the stack. See this reflected in our class (Listing 10 – `LuaEnvironment.hpp`).

Now I will show how to call it (Listing 11 – `main.cpp`). Some explanation:

❶: push first arg (x)

❷: push second arg (y)

❸: get 1st returned value. Recall that Lua has a LIFO stack, so the results are returned right to left. In this case the first returned value is x/y : 1.09091

❹: get 2nd returned value.

❺: get 3rd returned value.

❻: get 4th returned value.

Note: If you want to check if the returned value is fine, you can use the **lua_is**… function, that checks the type of value, see Listing 12 (`main.cpp`).

As well as **lua_isnumber**, you have:

```
lua_isstring
lua_istable
```

and so on…all with same prototype.

Listing 11

```
//...
LuaEnvironment luaEnv("sample.lua", true);
luaEnv.setFuncName("more_complex");
luaEnv.pushnumber(6); ❶
luaEnv.pushnumber(5.5); ❷
luaEnv.pcall(2, 4);

std::cout << luaEnv.tonumber() << std::endl; ❸
std::cout << luaEnv.tonumber() << std::endl; ❹
std::cout << luaEnv.tonumber() << std::endl; ❺
std::cout << luaEnv.tonumber() << std::endl; ❻
//...
```

Listing 12

```
//…
  if (!lua_isnumber(luaEnv.getLuaState(), -1))
    std::cout << "Error : value need be number."
       << std::endl;
//…
```

We have others stack operations, like **lua_gettop**, **lua_settop**, **lua_pushvalue**, **lua_remove**, **lua_insert** and **lua_replace**. See the API documentation for more detail about these functions.

Well, now is time to show how to call a C function from Lua.

Call C function from Lua

To call C functions from Lua you need to register the C functions with the Lua environment.

The first step is create a C function, see Listing 13 (`main.cpp`).

Listing 13

```
static int l_foo(lua_State* L)
{
  int value = abs(luaL_checknumber(L, 1)); ❶

  lua_pushnumber(L, value); ❷

  return 1;
}
```

You can change this for any action that you need. Well let's start with the explanations:

❶: Checks whether the function argument narg is a number and returns this number.

❷: here we push the result back to Lua, all will become clear …

```
void lua_register (lua_State*L,
                   const char *name,lua_CFunction f);
```

We use lua_register to register the new C function with Lua. In our class we add:

```
void registerFunc(
   const std::string& functionName,
   lua_CFunction func)
{
  lua_register(state_,
               functionName.c_str(), func);
}
```

In Listing 14, comment ❶ shows where we register the C function and the code in Listing 15 (`sample.lua`) calls it. ■

Acknowledgements

I would like to thank Tim Penhey and Jez Higgins for the various improvements they suggested for this article.

Bibliography

Lua: Roberto Ierusalimschy, Programming in Lua - 2nd ed. (http://www.inf.puc-rio.br/~roberto/pil2/) ISBN 85-903798-2-5

Lua.org: http://www.lua.org/

Mailing list: http://www.lua.org/lua-l.html

Community: http://www.lua.org/community.html

Using Lua to control your application: http://www.codeproject.com/samples/lua.asp

Calling Lua functions: using C++ language: http://www.codeproject.com/useritems/Calling_Lua_functions.asp

Wikipedia: http://en.wikipedia.org/wiki/Lua_%28programming_language%29

Reuse, Recycle, Refill?

Peter Hammond describes some pitfalls of reuse.

Introduction

We are all being encouraged to recycle and re-use these days, and in general I'm all for it. There is one area though where recycling is both unnecessary and harmful, and that is in naming variables when programming. Refilling a variable is often more like putting weed killer in a milk bottle than putting soup in a yoghurt pot. This paper shows some of the pitfalls that can arise through re-using a variable, how it can be avoided, and where it can be appropriate.

The cost of recycling

Many programmers start out with a procedural programming style. Those of us old enough to have starting programming on BASIC home computers would have had no choice. Many current C++ introductory texts (with some noble exceptions) perpetuate this style, even though it is neither necessary nor desirable in modern languages. With the older languages it was common to have to declare all the variables at the start of the scope, and then assign to them as they became usable. This lead some programmers to treat the local scope in much the same way that they might treat that box in the garage – to hold a bunch of stuff that might just 'come in handy' one day. Sometimes names were re-used because the programmer could see that the previous use was not needed any more and so could re-use the memory. In the old days of severely limited stack and compilers that did exactly what you told them to do, this may have been good practice, but for most of us those days are long gone. Often the re-use was simply a case of the programmer being too lazy to create a new name.

In more modern languages such as C++, it is no longer necessary to declare all the variables in advance: they can be created as needed. Furthermore the compiler is smart enough to work out when the variable is no longer in use, and recycle its memory automatically. This means that variables can be given sharp, clear names that help to document the code. If a variable is recycled by the programmer, then either the name has to be made fuzzy and weak, or at least one of the uses has the wrong name. Both happen in real life, and both are bad for readability and maintainability. Suppose we have an object that has a name, and buried in the middle of that name is a serial number that has to be found. This fairly typical approach demonstrates the first problem [1]:

```
const string digits ("0123456789")
string str = obj.get_name();
str = str.substr (str.find_first_of (digits),
                  str.find_last_of (digits));
```

`str` here is rather bland, but making the name more specific doesn't help:

```
string serial = obj.get_name();
serial = serial.substr (
  serial.find_first_of (digits),
  serial.find_last_of (digits));
```


PETER HAMMOND

Peter Hammond started out as a materials scientist, before changing career dirrection and joining BAE Systems in the late 1990s. Since then he has worked on a number of defence related systems, large and small, with particular interests in component architectures, real-time systems, and agile methods.

Embedding Lua into a C++ Application (continued)

Listing 14

```
#include "LuaEnvironment.hpp"
#include <iostream>
static int l_foo(lua_State* L)
{
  int value = abs(luaL_checknumber(L, 1));
  lua_pushnumber(L, value);
  return 1;
}
int main(int argc, char ** argv)
{
  try
  {
    LuaEnvironment luaEnv;
    luaEnv.openStandardLibs();
    luaEnv.registerFunc("foo", l_foo); ❶
    luaEnv.dofile("sample.lua");
  }
  catch(const std::runtime_error& runtimeError)
  {
    std::cout << runtimeError.what() <<
       std::endl;
    return 1;
  }
  return 0;
}
```


C++: Bjarne Stroustrup, The C++ Programming Language - 3rd ed. ISBN-13: 978-0201700732

C++ Books (full list): http://www.amazon.com/C%2B%2B-Books/lm/R1FWITBSD4E82F/ref=cm_lm_byauthor_title_full/103-7510116-1265448

Bjarne Stroustrup's homepage: http://www.research.att.com/~bs/homepage.html

Note: Web links listed here may not be valid in the future.

Listing 15

```
function myABS(p)
  return foo(p);
end

-- call foo
value = foo(-55);
print(myABS(-333));
print(value);
```

The Boost Assign Library

Paul Grenyer recently gave us a look at several ways to fill a container [2]. The Boost Assign library [3] gives us another approach. It allows standard containers to be filled with something closely approximating the aggregate syntax that is only available for arrays of PODs. Clearly this is only useful when the set of objects is known at compile time, and is reasonably short; it would be no use for creating a range between x and y in steps of z, where the values are not known until runtime, or between 0 and 10000 in steps of 2. It is however extremely convenient for filling containers with known objects, which can themselves be variables.

An example

Consider a test case where a method is expected to return a set of objects of some class type. Without Boost Assign, you have two choices. One approach is to fill a container using something like push_back:

```
vector<int> expected_results;
expected_results.push_back (0);
expected_results.push_back (1);
expected_results.push_back (1);
expected_results.push_back (2);
expected_results.push_back (3);
BOOST_CHECK (fib(5) == expected_results);
```

The other is to compare each member of the returned collection separately:

```
const vector<int> results = fib(5);
BOOST_CHECK (results.size() == 5);
BOOST_CHECK (results[0] == 0);
BOOST_CHECK (results[1] == 1);
BOOST_CHECK (results[2] == 1);
BOOST_CHECK (results[3] == 2);
BOOST_CHECK (results[4] == 3);
```

The Assign facility allows you to create an expected result collection easily, with minimal tedious repetition, and has the additional benefit that the result can be made const:

```
const vector<int> expected_results =
  list_of (0)(1)(1)(2)(3)
BOOST_CHECK (fib(5) == expected_results);
```

How it works

For the full details, the reader should of course consult the library's documentation and code. The design of the library is as simple as its use; it relies on no brain-melting template metaprogramming techniques, just operator overloading and some 'classic' templates. In brief, it overloads `operator ()` to permit objects to be constructed inline. It also overloads the assignment operators to return something that has operator, overloaded to push its argument onto the collection. This is not applicable when constructing a const object, but is illustrated by something like this example from the documentation:

```
my_vector<int> vec;
vec = 1,2,3,4,5,6,7,8,9,10;
```

since in the first of these two lines, it is not the serial part, it is the whole name, so the code is telling fibs. The answer is to name each concept correctly:

```
const string obj_name = obj.get_name ();
const string serial = obj_name.substr (
  obj_name.find_first_of (digits),
  obj_name.find_last_of (digits));
```

We will come back to the issue of efficiency and the additional string object later.

Good names are very important in expressing the design and intent of a piece of code, and they can help you to catch bugs; you are much more likely to spot the error (when it is buried in the middle of several lines of arithmetic) in

```
const float acceleration = position / speed;
```

than in

```
x = p/s;
```

A recycled variable makes code harder to read in another way too. When reading a moderate sized block of code, it is easy to miss a re-assignment between where you saw a variable being declared and where you are now. If, for example, the stream that was originally attached to a configuration file has been recycled as a data file, you may be left wondering for some time how the code was ever supposed to have worked in the first place. You will probably find it after a few minutes, but it slows you down and breaks concentration.

Reusing a variable will inevitably lead to variables whose scope exceeds their usefulness. The consequences are generally most obvious pointer with variables, as between outliving its last use and being pulled out of the 'come in handy box' it points at a dead object or some random piece of heap, and de-referencing it may lead to an access violation. Using a random integer may cause a less spectacular failure, but it is all the worse for that. Such a bug could lie dormant for years. A particularly virulent form of excessive scope is the member variable that is not actually part of the class's state; the original programmer put it in class scope to avoid passing a parameter to a private implementation function. Now, consider the hapless maintenance programmer who has just landed in one of that class's methods; how does he know what is in that member? Particularly if it is named something unhelpfully general like `m_file`. The only answer is to trawl the class's methods trying to pair up validations and invalidations of the object.

Prodigious use of consts

So, if we don't want to re-use variables, what do we do? Well, fairly obviously, declare a new name for every object. What's more, since we are not going to reuse it once it has done its particular job, we can make it immutable – `const` in C++, `final` in Java. This can save a whole set of errors related to accidentally changing it: for example by using `=` instead of `==`, by passing it to a mutating operation that we thought was read-only, or by a careless maintenance programmer thinking it was safe to reuse when it was not. Sometimes it is not possible to initialise an object directly, since it needs to be incrementally built. However, often this is a single stage and the object is `const` once completed. These cases can be taken care of by factoring out the construction code, or by using the Boost Assign library (see sidebar).

This leads to a style that becomes reminiscent of functional programming. In fact, once you get used to it, you might well find yourself dropping many of the names altogether, and chaining results into arguments directly. This may not be such good thing. As mentioned earlier, names are important for self-documenting code. It is probably easier, and certainly more reliable, to name an intermediate concept rather than writing a comment, so name those concepts. Also named objects have definite scope, and this can help with exception safety. For example, the Boost `scoped_ptr` library recommends assigning the result of every `new` to a named `scoped_ptr` to ensure the correct deallocation should an exception occur. C++ has many traps for the unwary; one of these is the order of evaluation of function arguments [4]. In the following example, it is implementation defined whether `bar` is called for `x` or `y` first:

```
void foo (int x, int y);
int bar();
foo (bar(), bar());
```

If `bar()` is stateful (perhaps reading the input from a stream, for example), this may result in strange errors. Declaring the intermediate constants explicitly removes this:

```
const int x = bar();
const int y = bar();
foo (x, y);
```

If you are adopting this style, soon you are going to come across the situation where the name you want to use is already in use. When this happens, ask yourself two questions. First, is it really the right name for both of them? If the name is `file`, probably it is just right for neither of

them. Ask yourself what each is. If you say one is the configuration file and one is the data, then there are your names. In other words, try to make both names more precise, and often that means longer. The second question is whether the scope is too large. Perhaps you need to refactor to bring out a small function which removes the ambiguity from the object name. Have you put a variable at class scope when a `const` at method scope would be more appropriate?

Efficiency

There are two forms of efficiency that are often cited as reasons to re-use a name: stack space and time. The stack space argument is generally spurious in modern compilers. Apart from the fact that a few extra bytes on the stack is very unlikely to be an issue these days, an optimising compiler will do away with the intermediate names anyway. Listing 1 illustrates this point clearly. It shows three versions of a simple function, with the salient parts of the corresponding object code, showing that there is no penalty for introducing meaningful names. It was compiled using Microsoft Visual Studio 7.1 with default optimisation for Release build settings. Comments and instructions not related to the comparison have been removed from the object listings for clarity.

The speed issue may be more significant. It is often true that a procedural style may be more efficient, since it can avoid making copies by working in-situ. However, remember the old maxim, 'Make it right, make it fast' [5]: it is more important that software be correct than fast. You should never appeal to efficiency without evidence from a profiler.

Re-assignment where appropriate

Still, you may well find that after all this you get back to a point where you want to re-assign to a variable. Unless you are working in a pure functional language, that is not a problem. The popular general purpose languages all support the procedural style, and it is appropriate for elegant solution of many problems. For example, the Fibonacci series is a classic example of a functional algorithm, but has exponential complexity. A more procedural style gives linear complexity with a compact algorithm [6]:

```
unsigned fib(unsigned count) {
  vector<unsigned> memo(2, 1u);
  for (unsigned i = 2u; i <= count; ++i)
     memo.push_back(memo[i - 1] + memo[i - 2]);
  return memo.back();
}
```

When the correct paradigm has been selected, re-assignment does not indicate that the name is being re-used, but merely continuing to be used for its original purpose. This should be the simple test for a piece of code; at any given point, does the object's name express its purpose clearly and correctly?

Conclusions

Clear, strong names can greatly improve reliability, readability and maintainability of software. Adopting a functional-like style, where each result is named in a constant, is a natural continuation of strong naming which can enhance these benefits. This is not to say that there is no place for variables and the procedural style of programming. However, once a programmer has developed the habit of not re-using names because they happen to be around, procedural style re-assignment is likely to be reserved for situations where it is appropriate. ■

Acknowledgements

Thanks to the many reviewers whose input helped to improve this article, particularly Simon Sebright, Paul Grenyer, Matt Reeve and the CVu review panel.

Notes and references

[1] In all code examples, namspaces `std::` and `boost::` have been omitted for brevity.

Listing 1

```
float getDistance();
float kmToMiles (float);
float alreadyGone();
void foo1 () {
  float distance = getDistance();
  distance = kmToMiles (distance);
  distance -= alreadyGone();
}
void foo2() {
  const float distance_km = getDistance();
  const float distance_miles = kmToMiles (
     distance_km);
  const float distance_left =
     distance_miles - alreadyGone();
}
void foo3() {
  const float distance_left =
    kmToMiles (getDistance()) - alreadyGone();
}
_distance$ = -4
?foo1@@YAXXZ PROC NEAR
  00000 51 push ecx
  00001 e8 00 00 00 00 call ?getDistance@@YAMXZ
  00006  d9 1c 24 fstp DWORD PTR
     _distance$[esp+4]
  00009  8b 04 24 mov eax,
     DWORD PTR _distance$[esp+4]
  0000c  50 push eax
  0000d  e8 00 00 00 00 call ?kmToMiles@@YAMM@Z
  00012  dd d8 fstp ST(0)
  00014  e8 00 00 00 00 call ?alreadyGone@@YAMXZ
  00019  dd d8 fstp ST(0)
_distance_km$ = -4
?foo2@@YAXXZ PROC NEAR
  00000  51 push ecx
  00001  e8 00 00 00 00 call ?getDistance@@YAMXZ
  00006  d9 1c 24 fstp DWORD PTR
     _distance_km$[esp+4]
  00009  8b 04 24   mov   eax, DWORD PTR
     _distance_km$[esp+4]
  0000c  50     push   eax
  0000d  e8 00 00 00 00 call ?kmToMiles@@YAMM@Z
  00012  dd d8 fstp ST(0)
  00014  e8 00 00 00 00 call ?alreadyGone@@YAMXZ
  00019  dd d8 fstp ST(0)
?foo3@@YAXXZ PROC NEAR 00000  e8 00 00 00 00
   call   ?getDistance@@YAMXZ
  00005  51 push ecx
  00006  d9 1c 24 fstp DWORD PTR [esp]
  00009  e8 00 00 00 00 call ?kmToMiles@@YAMM@Z
  0000e  83 c4 04 add esp, 4
  00011  dd d8 fstp ST(0)
  00013  e8 00 00 00 00 call ?alreadyGone@@YAMXZ
  00018  dd d8     fstp   ST(0)
```

[2] Paul Grenyer, 'Loading a Container with a Range', *CVu* 18-6, p 16, Dec 2006.

[3] Thorsten Ottosen, 'Boost.Assignment documentation', http://www.boost.org/libs/assign/doc/index.html

[4] Scott Meyers, *Effective C++*, 3rd ed, Addison-Wesley, NY, p 76.

[5] James Coplien and Kent Beck, 'After all, we can't ignore efficiency – part 2', C++ Report, p72, Jul 96; cited at http://home.earthlink.net/~huston2/dp/run_rite.html

[6] Based on Andrei Alexandrescu, 'The Appliance of Science: Things in Computer Science that Every Practitioner Should Know', ACCU Conference, Oxford. http://accu.org/content/conf2007/Alexandrescu-The_Appliance_of_Science.pdf, with small changes to emphasise the points of this paper rather than the original.

Code Critique Competition 49

Set and collated by Roger Orr

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;
  }
}
```

A book prize is awarded for the best entry. 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'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?

(See Listing 1 – thanks are due to Hubert Matthews for supplying the original idea for this critique.)

Critiques

From Ivan Uemlianin <ivan@llaisdy.com>

Comment on original code.

Crashes and bugs

The comment 'sometimes the program crashes' proved of no use to me. I ran the program a few times and it didn't crash. Rather than try to work out what might be causing any kind of hypothetical intermittent crash, I went straight on to analyse and improve the code. Far better would have been even a single example crash.

Although there were no crashes, there was some strange behaviour. I first looked at this program on a windows laptop – running g++ 3.4.4 under cygwin – and all seemed well. When I got home and looked at it with my main machine – running g++ 4.2.1 under debian – things suddenly seemed very unwell. Essentially the input was not being stored properly in the `tm[]`: the first date input was always output as 30th November year-1. For example:

Input	Output
1999-12-12	Mon 30-Nov-1998
2001-01-12	Thu 30-Nov-2000
1910-06-06	Tue 30-Nov-1909

I wondered whether the value returned from the increment in line 17 was interfering with the input stream. In any case, initialising the count to 0 instead of -1, and moving the increment from the condition to the execute block of the `while` loop, removed the bug.

As well as the above, there are several small things wrong with this program, which add up to make the program difficult to read and therefore difficult to maintain.

Range of use

It might be worth noting in passing that the `tm` data structure is only good for years since 1900. Years before that will be output as Wed 31-Dec-1969 or Thu 01-Jan-1970 or something similar depending on the platform.

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

Variable names

Variables are given meaningless names. This is fair enough with the count variables **i**, **j**, **k**, and the temporary **char** variable **ch**, but I would prefer meaningful names for variables that carry meaningful content. What counts as meaningful is of course open to debate, but I think many would agree that **b** is not.

Two variables are even given the same name as their type (i.e., **tm *tm** and **time_t time_t**). One could claim that these are meaningful names, in the same way that prefixes in Hungarian notation are meaningful. However, such a naming convention can only have limited use in small programs (i.e., with only one instance of the type) and might therefore count as a bad habit; and I must say I found it disconcerting to have type names popping up as variable names throughout the program. Better would have been something like **tm *dates** and **time_t timeType**).

Use of macros

The program uses a preprocessor macro to set **CLOCK_YEAR_OFFSET** to 1900 (line 5). Preprocessor macros insert arbitrary code segments into a program before compilation and are rather a powerful tool. The purpose of this line, however, is merely to define a constant integer. Far better to say so explicitly, both to the compiler and to any humans who might read the code, with something like:

```
const int CLOCK_YEAR_OFFSET = 1900;.
```

There are other constants in the program which have been left as magic numbers which would also benefit from the **const int** treatment.

Entangled operations

Between input and output, lines 20-51 perform two operations: they adjust the input for each date and they sort the dates. These two operations are logically independent and the simplest thing would be to do one before the other. The program decides to do both at the same time, resulting in gratuitously complicated and repetitious code.

Everything in main( )

Finally, everything is in **main()**. The **main()** function should tell you at a glance what the program is supposed to do, with the nitty-gritty elsewhere in the program. For this program, an ideal **main()** would contain three functions: **input_dates()**, **sort_dates()** and **output_dates()**. The three functions could then be tested and refined independently.

New code

The listing below shows a version of the program with the above changes. I have moved adjustment of input out of the sort routine and into **input_dates()**. I have not added unit testing, or used any more STL than that used in the original program. Indeed, the operation of the program is identical to that of the original. The new program is twenty lines longer, but I think is easier to read, and should be easier to maintain.

```
// Read dates, and display as sorted text
#include <time.h>
#include <iostream>
using namespace std;
const int CLOCK_YEAR_OFFSET = 1900;
const int LOCAL_TIME_LENGTH = 16;
const int MAX_SIZE = 10;
int input_dates(tm dates[]);
void output_dates(tm dates[], const int size);
void sort_dates(tm dates[], const int size);
int input_dates(tm dates[])
{
  int i = 0;
  char ch;
  cout << "Enter up to three dates (yyyy-mm-dd)"
    ", end with 'exit': ";
  while ( cin >> dates[i].tm_year >> ch
  >> dates[i].tm_mon >> ch
  >> dates[i].tm_mday )
  {
    dates[i].tm_year = dates[i].tm_year -
      CLOCK_YEAR_OFFSET;
    dates[i].tm_mon = dates[i].tm_mon - 1;

    cout << ": ";
    i++;
  }
  return i;
}
void sort_dates(tm dates[], const int size)
{
  for ( int j = 0; j != size; j++ )
  {
    for ( int k = j; k != size; k++ )
    {
      if ((dates[k].tm_year > dates[j].tm_year)
          ||
       ((dates[k].tm_year == dates[j].tm_year)
       && (dates[k].tm_mon > dates[j].tm_mon))
          ||
       ((dates[k].tm_year == dates[j].tm_year)
       && (dates[k].tm_mon == dates[j].tm_mon)
       && (dates[k].tm_mday > dates[j].tm_mday))
        )
      swap(dates[j], dates[k]);
    }
  }
}
void output_dates(tm dates[], const int size)
{
  char buffer[LOCAL_TIME_LENGTH];
  time_t time_type;
  int i = size;
  while ( i-- )
  {
    strftime(buffer, sizeof(buffer),
      "%a %d-%b-%Y",
      localtime(&(time_type= mktime(dates+i))));
    cout << buffer << endl;
  }
}

int main()
{
  tm *dates = new::tm[MAX_SIZE];
  int size;
  size = input_dates(dates);
  sort_dates(dates, size);
  output_dates(dates, size);
  return 0;
}
```

From John Penney <J.Penney@servicepower.com>

Bugs, like brown sticky stuff, happen. Bugs plague even the most experienced coder – mostly because even the most experienced coder sometimes has to work with other people's code! But one thing that makes a big difference is how you track down such problems.

I am firmly in the test-first camp, and I've chosen to take on this month's Code Critique with some test-first debugging. Not only will this find the bug, but we will end up with a good understanding of what the code is doing, and a good framework for moving on to refactoring to something a little more elegant!

An initial analysis

The code seems almost wilfully obscure. We have one (count 'em!) comment in the whole code. Most of the variables are one character long. The type name **tm** has been overloaded as a local variable name **tm**.

Fortunately that one comment turns out to be pretty helpful in deciding how to split the code up and apply unit tests. It tells us the program does three things:

1. Read some dates
2. Sort the dates
3. Display the sorted dates

So I took each of these 3 chunks of code in turn and placed it in a local function with the intention of adding unit tests for each, using `assert()` to confirm correct behaviour. Note that in this first refactoring we're necessarily operating without the aid of unit tests, so must change as little as possible. We can of course run the application manually to get a measure of reassurance that we haven't broken it.

Testing date input

The first operation – to read some dates – might look tricky to unit test, but with the help of ostringstream and istringstream it becomes much easier:

```
int getInputs(ostream& os, istream& is,
  tm* tm)
{  // cout and cin replaced with os and is
  os << "Enter up to ten dates "
        "(yyyy-mm-dd), end with 'exit': ";
  while ( is>>tm[++i].tm_year>>ch
            >>tm[i].tm_mon>>ch
            >>tm[i].tm_mday )
    os << ": ";
  return i;
}
```

So that our caller looks like this:

```
i = getInputs(cout, cin, tm);
```

And here are a couple of my tests:

```
{
  ostringstream os;
  istringstream is("exit"); // No dates
  const int numRecs= getInputs(os, is, tm);
  assert(numRecs == 0);
}
{
  ostringstream os;
  istringstream is("2007-10-22\n"
    "2007-10-23\n2001-01-01\nexit");
  const int numRecs= getInputs(os, is, tm);
  assert(numRecs == 3);
  assert(tm[0] == makeTM(2007, 10, 22));
  assert(tm[1] == makeTM(2007, 10, 23));
  assert(tm[2] == makeTM(2001, 1, 1));
}
```

As you can see, I've introduced a 'test fixture' called `makeTM()`. This is a trivial function that just creates a `tm` object. What might not be so obvious is that I've also added an `operator==` for a `tm` object.

```
tm makeTM(int year, int mon, int mday)
{
  tm localTM;
  localTM.tm_year = year;
  localTM.tm_mon  = mon;
  localTM.tm_mday = mday;
}

bool operator==(const tm& lhs, const tm& rhs)
{
  return lhs.tm_year == rhs.tm_year &&
         lhs.tm_mon  == rhs.tm_mon &&
         lhs.tm_mday == rhs.tm_mday;
}
```

These tests all pass (for normal input data). So the bug ain't here, but we've still achieved a lot. `makeTM()` and the `operator==` were added to make the tests easier to write and read, but we'll find that subsequent refactorings of the production code will make use of these. This is quite typical! Also, if it becomes a requirement to be more vigilant against stupid user input, we could add test cases for bad years (1889), bad months (13, 00), bad days (00, 32), bad formatting (17th Nov 2004) etc.

Testing date sorting

We're told our second operation is to sort the supplied dates. So I just moved the relevant chunk of code into a new local function called `sortTM()`:

```
void sortTM(tm* tm,
            int i)  // Number of tm records
{
  // Sorting code unchanged
    :
}
```

I added a test with `i=0`: no failures. Buoyed with success, I added a test with `i=1`… my test failed because in fact the function does not just sort the dates, but it modifies the `tm` objects so that they're suitable for input to `mktime()` later on:

```
{
  tm[0] = makeTM(2007, 10, 22);
  sortTM(tm, 1);
  assert(tm[0] == makeTM(107, 9, 22));
}
```

Note that my goal right now is to add tests and find the bug, not to change the code. So I left the functionality of `sortTM` unchanged and modified my expected results.

Adding my next test with `i=2` produced another test failure and another surprise. The dates are sorted with the most recent first:

```
{
  tm[0] = makeTM(2007, 10, 21);
  tm[1] = makeTM(2007, 10, 22);
  sortTM(tm, 2);
  assert(tm[0] == makeTM(107, 9, 22));
  assert(tm[1] == makeTM(107, 9, 21));
}
```

I added a few more tests and received no further surprises.

Testing date output

I moved onto the last operation of the code, to display the results. I thought it would be easier to work with one `tm` object at a time, so just introduced a function to format one `tm` object:

```
void formatOutput(char* buff, int buffSize,
  tm* theTM)
{
  time_t time_t;
  strftime(buff,buffSize,"%a %d-%b-%Y",
    localtime(&(time_t=mktime(theTM))));
}
```

I added my first test:

```
{
  tm[0] = makeTM(107, 9, 22);
  formatOutput(b, sizeof(b), tm);
  assert(strcmp(b, "Mon 22-Oct-2007") ==
    0);
}
```

The program crashed... hurrah!

The bug

I had my suspicions as to the cause, but confirmed the problem using the debugger. It turns out that the `mktime()` call is failing, returning -1. This in turn makes `localtime()` fail, leaving `strftime()` to go belly-up.

My IDE immediately shows me the problem: not just one an uninitialised variable but lots of uninitialised variables! The `tm` type has many data members but we've only set `tm_year`, `tm_mon` and `tm_mday`.

`mktime()` is trying to produce a `time_t` (the time represented as a number of seconds) and to do that it needs to consider the `tm` object's member data representing number of hours, minutes and seconds, which we've not initialised.

As is the way with uninitialised variables, these could be zero (in which case `mktime()` will work) or could be something else (in which case `mktime()` might work!).

The fix

Now we can easily make the test work by initialising all the members of the `tm` object in `makeTM()`:

```
tm makeTM(int year, int mon, int mday)
{
  tm localTM;
  localTM.tm_year = year;
  localTM.tm_mon  = mon;
  localTM.tm_mday = mday;
  localTM.tm_hour = 0;
  localTM.tm_isdst = -1;  // "unknown"
  localTM.tm_min = 0;
  localTM.tm_sec = 0;
  localTM.tm_wday = 0;
  localTM.tm_yday = 0;

  return localTM;
}
```

However, this doesn't fix the production code because `makeTM()` isn't used to create the `tm` objects that `getInputs()` returns. But why not reuse `makeTM()` in `getInputs()` – and make it a bit easier to read at the same time:

```
int getInputs(ostream& os, istream& is, tm* tm)
{
  os << "Enter up to ten dates "
        "(yyyy-mm-dd), end with 'exit': ";
  int i = 0;
  char ch;
  int year, mon, mday;

  while (is>>year>>ch>>mon>>ch>>mday)
  {
    tm[i++] = makeTM(year, mon, mday);
    os << ": ";
  }
  return i;
}
```

And we can re-run our unit tests to confirm that we've not broken `getInputs()`. (Actually I should confess that my first refactoring did break `getInputs()`: with `i` initialised to -1 the tests failed!)

Let someone else do the hard work #1

So now we have a fully working program with a good set of unit tests. What further refactorings can we do? Well, one important principle in C++ is 'don't attempt to do something that the STL can do for you'. And the STL can do sorting and loops pretty neatly. This knowledge led me to this rather startling simplication of our `sortTM()` function:

```
void sortTM(tm* tm,
            int i)  // Number of tm records
{
  std::sort<tm*>(&tm[0], &tm[i],
    compareTM);
  for_each<tm*>(&tm[0], &tm[i], mangleTM);
}
```

Note that we can pass pointers to the first and one-past-the-last `tm` objects, and they quite happily work as iterators in `sort()` and `for_each()`. You gotta love that!

The new function `compareTM()` neatly encapsulates the existing code to compare `tm` objects:

```
bool compareTM(const tm& lhs, const tm& rhs)
{
return (lhs.tm_year > rhs.tm_year) ||
       (lhs.tm_year == rhs.tm_year &&
        lhs.tm_mon > rhs.tm_mon) ||
       (lhs.tm_year == rhs.tm_year &&
        lhs.tm_mon == rhs.tm_mon &&
        lhs.tm_mday > rhs.tm_mday);
}
```

And `mangleTM()` isolates our code to prepare the `tm` struct for the `mktime()` call:

```
void mangleTM(tm& theTM)
{
  theTM.tm_year-=CLOCK_YEAR_OFFSET;
  --theTM.tm_mon;
}
```

Let someone else do the hard work #2

The STL is rather good at containers as well. They are fast, easy to use and extendable. If we used an STL container instead of the hard-coded array of 10 `tm` objects, we'd lose the limit on 10 dates and wouldn't have to carry the container length around with us (variable `i`). Which STL container to choose? Well, a good default is `std::vector()` and it suits us well enough here.

Another thing that was bugging me from the first was the overload of `tm`: it's a type name and a variable name. OK, it works, but it's not very readable, so while replacing the `tm` array with a vector, I also renamed it.

Our `main` is now more concise, readable and maintainable:

```
int main(int argc, char* argv[])
{
  std::vector<tm> theTMs;
  char b[16];

  getInputs(cout, cin, theTMs);

  sortTM(theTMs);

  for(vector<tm>::const_iterator iter
    = theTMs.begin() ;
      iter != theTMs.end() ;
      ++iter)
  {
    formatOutput(b, sizeof(b), *iter);
    cout << b << endl;
  }
}
```

The tests are run and pass.

Exercises for the reader

I stopped refactoring here as the Code Critique deadline was looming… but you don't have to stop!

If this code were performance-critical I'd consider replacing the `std::vector` with a `std::list` and use the member version of `sort` (`std::list::sort()`). Member versions of algorithms are always preferable to non-member versions because they can take advantage of knowing how the container is organised – and so can avoid creating unnecessary copies of objects for example.

You could also make `formatOutput` return a `std::string` object:

```
cout << formatOutput(*iter) << endl;
```

This refactoring means that the client code (`main`) no longer knows or cares how long the formatted output is – allowing us to change it if needs be.

The unit tests look like this:

```
cout << "Testing sortTM()..." << endl;
{
  theTMs.push_back( makeTM(2007, 10, 22) );
```

```
  sortTM(theTMs);
  assert(theTMs[0] == makeTM(107, 9, 22));
}
// more tests elided
cout << "... PASS" << endl;
cout << "Testing formatOutput()..." << endl;
{
  const tm theTM = makeTM(107, 9, 22);
  formatOutput(b, sizeof(b), theTM);
  assert(strcmp(b, "Mon 22-Oct-2007")== 0);
}
// more tests elided
cout << "... PASS" << endl;
```

[Ed: The complete code, including the unit tests, was listed in full. I have summarized it to save space]

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

Besides a memory leak (you `new::tm[10]` but never `delete[]` it – and there is no reason to `new` it in the first place; a local variable would work just fine), one possible problem is with the line:

```
while ( cin>>tm[++i].tm_year>>ch
          >>tm[i].tm_mon>>ch
          >>tm[i].tm_mday )
```

While `i` has incremented before evaluating `tm[++i].tm_year`, that increment may happen before or after evaluating `tm[i].tm_mon` and/or `tm[i].tm_mday`, leaving `i`, `tm_mon` or `tm_mday` in an unknown state.

Also, nothing enforces the limit of 10 items. If you accidentally enter more, the program will be in an unknown state as well.

The other fields of the `tm` data structure are not initialized. This may or may not work correctly when `mktime` is called. So how could the program be improved?

While the code is just one big `main()` program, there is structure to it:

1. Input the dates
2. Looping over the dates:
   a) Fix the year and month so that `tm_year`, `tm_mon` and `tm_mday` match local time
   b) Loop over the rest of the dates and effective perform a hand coded (insertion) sort, from latest date to earliest date
3. Loop over the now sorted dates in reverse order and display them

To improve the program, I'll separate out each piece of functionality into a separate function, and be a little smart on the names I pick for those functions. Also, instead of hand-coding a sort, I'll use `std::sort`. Finally, I'll get rid of that pesky 10 date limit.

1. Include various headers:

```
#include <algorithm>
#include <ctime>
#include <istream>
#include <iostream>
#include <iterator>
#include <ostream>
#include <vector>
```

2. Write a function whose purpose is to input a date from a stream and convert it to a `tm`:

```
std::istream& operator>>(std::istream& lhs,
                         tm& rhs)
{
  tm      tmtm = tm();
  char    c;
  if (lhs >> tmtm.tm_year >> c
      >> tmtm.tm_mon >> c >> tmtm.tm_mday)
  {
    tmtm.tm_year -= 1900;
    --tmtm.tm_mon;
    ::mktime(&tmtm);
    rhs = tmtm;
  }
  return lhs;
}
```

Because I used `operator>>` as the name of my function, I can directly use it with any istream in any part of my program, without having to remember a different, non-matching syntax. This is how streams are meant to be used.

All the unused fields of `tmtm` are initialized to 0. If I can't read it in, I don't modify the caller's variable. I immediately convert it to the unbroken time. I never know who is going to use it once it leaves this routine.

3. Write a function to output a `tm`:

```
std::ostream& operator<<(std::ostream& lhs,
  tm const& rhs)
{
  char    buf[sizeof "Wed 31-Oct-2007"];
  ::strftime(buf, sizeof buf,
    "%a %d-%b-%Y", &rhs);
  return lhs << buf;
}
```

I don't know or care how big `buf` is; `sizeof` calculates the right size to fit a C string of this form.

4. Write a function to output a `vector<tm>` (since I'll be storing them in a vector instead of a fixed size array):

```
std::ostream& operator<<(std::ostream& lhs,
  std::vector<tm> const& rhs)
{
  std::copy(rhs.begin(), rhs.end(),
    std::ostream_iterator<tm>(lhs, "\n"));
  return lhs;
}
```

5. Write a function to compare `tm`s:

```
bool operator<(tm const& lhs,
  tm const& rhs)
{
  tm l(lhs);
  tm r(rhs);
  return ::difftime(::mktime(&l),
    ::mktime(&r)) < 0.;
}
```

Instead of hand coding the difference calculation, use `difftime` (on local copies of the time passed in, since we don't want to modify the user parameters passed in).

By naming the function `operator<`, it can be used naturally by `std::sort`.

6. Finally, add the greatly simplified `main()`:

```
int main()
{
  std::vector<tm> theTms;
  tm              tmtm;

  std::cout << "Enter dates (yyyy-mm-dd), "
    "end with 'exit': ";
  while (std::cin >> tmtm)
  {
    theTms.push_back(tmtm);
    std::cout << ": ";
  }

  std::sort(theTms.begin(), theTms.end());

  std::cout << theTms;
}
```

By using a vector, the only limit on the number of dates is the amount of memory available to the program.

Listing 2

```
// cmap.h
#include <map>
using namespace std;

// map that has a const operator[]
template <typename K, typename V>
class cmap : public map<K,V>
{
  typedef map<K,V> base;

public:
  using base::operator[];

  const V& operator[](K const &k) const
  {
    if ( count(k) == 0 )
    {
      static V v;
      return v;
    }
    return ((base)(*this))[k];
  }
};

// test_cmap.cpp
#include "cmap.h"
#include <iostream>
#include <string>

void test( cmap<int, string> & m, int idx )
{
  cout << "m[" << idx << "]="
    << m[ idx ] << endl;
}
void ctest( const cmap<int, string> & m,
            int idx )
{
  cout << "const m[" << idx << "]="
    << m[ idx ] << endl;
}
int main()
{
  cmap<int, string> map;
  map[0] = "Zero";
  map[1] = "One";
  map[2] = "Two";

  test( map, 2 );
  test( map, 3 );

  ctest( map, 2 );
  ctest( map, 3 );
}

//Example output when it goes wrong
// (const m[2] should be Two):
//m[2]=Two
//m[3]=
//const m[2]=
//const m[3]=
```

Use **`std::sort`** instead of the hand coded insertion **`sort`**. It will be faster in most circumstances, and being in the standard library, it is code you can be reasonably sure is correct (and you get all that with just one line of user code!)

The vector is actually sorted from earliest to latest date. To change that order to match the original program, replace **`begin()`** and **`end()`** with **`rbegin()`** and **`rend()`** throughout the program.

Commentary

When I first saw the code that formed the basis of the code critique, I was mostly struck two things, leaving aside for a moment the problem of the crash. One was the amount of repetition in the code, and the other was lack of clarity in the code – it was all in a single block.

The main problems with repetition are that it is hard to read (the text that matches tends to mask the text that's different), it is expensive to change (as the same change must be applied multiple times multiple lines of code) and it is inefficient (as the same instruction sequence is executed more than once).

The problem with the structure is that the same piece of code is comparing the dates was converting the format and converting the representation. Separating these out into individual functions makes it possible to write re-usable code (or make use of existing code).

One brief aside on the line **`tm *tm = new::tm[10];`**. This uses the same name for the variable as for the data type, which then means that the scope resolution operator **`::`** must be used in the allocation. Note that the **`::`** is binding to the following **`tm`** and not to the preceding **`new`**.

The winner of CC 48

I liked all the entries – they all both fixed the initial problem and made the code clearer. Both John and Nevin made use of the standard library sort in their solutions, which is (almost always) the right thing to do.

However, I was particularly taken with Ivan's **`main()`** as it was so very short and clear, so I've awarded him this issue's prize.

Code Critique 49

(Submissions to scc@accu.org by 1st January 2008)

> I'm frustrated by the map class in the standard because the indexing operator isn't **`const`**. So I'm trying to made my own class – **`cmap`** – which has a **`const`** friendly **`operator[]`**. It almost works, but I sometimes get the wrong value output – can you help me?

As always, try to go beyond simply solving the initial problem. The code is in Listing 2.

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. ■

Prizes provided by Blackwells Bookshops and Addison-Wesley

Guidelines for Contributors

Thinking of writing for us? Follow these guidelines to help smooth the way.

These guidelines provide general instructions on the submission of articles for publication. For more detailed information, please contact the editor of the relevant publication (cvu@accu.org or overload@accu.org).

For examples of the elements described, please see recent issues of the journals.

With your article, you need to send:

- A short personal profile (see 'Profile')
- Any images used in your article as separate files (see 'Illustrations')
- An introductory line or sentence (see 'Structure')

Format

Articles can be accepted as Word or Open Office documents; alternatively, save your file as RTF. If your article is in any other format, please check with the production team that it can be opened and the text extracted.

If you are using a text editor, devise and explain a convention for your document. For example, 'Heading 1 in block capitals, heading 2 underlined, code snippets in text surrounded by <<c>>'. You can use a numbered system for headings – the numbers will be removed from the finished article. Clearly separate any such instructions from the body of the document.

Text for sidebars or panels can either be inserted in the approximate position in the text or provided at the end if its location is unimportant. Clearly mark the beginning and end of the text forming the sidebar.

Please do not apply complex formatting to your articles – this will almost certainly be changed during the typesetting process to comply with the journal standards. The formatting you apply is, however, used as a guide by the production editor when determining how to format particular elements.

A fixed width font is used to display code fragments and filenames (bold for code and non-bold for comments and filenames) – if possible, use a suitable font in your article to indicate this (Courier, for example).

The formatting of listings will almost certainly be changed due to space constraints. Unless there is good reason to do otherwise, code is placed in a single column, giving a maximum number of characters per line of 48. If the formatting of your code is important to you, you are welcome to ensure your listings fit within this number of characters and every attempt will be made to retain your personal style, although this cannot be guaranteed. If the formatting of the code is particularly relevant to a point you are making (for example, contrasting two different styles of writing), please say.

A Word template is available containing a Code paragraph style set to the correct width.

Profile

A short profile of the author is required for all articles. This should be brief – approximately 50 words – and may need to be shortened if space is tight.

Please ensure your name is as you would wish it to appear in the journal and include an email address or another means of contact (for example, via a website).

For C Vu only, a photograph of the author (head and shoulders) is also required. This should be high-contrast and at a minimum resolution of 200 dpi (ideally 300 dpi).

Structure

Try to keep titles short and relevant - they should ideally fit on a single line and definitely require no more than two lines. A short strapline (C Vu) or introductory sentence (Overload) is required – if these are not provided by the author, they will be created as part of the editorial process.

Both publications have provision for three levels of heading within the body of the article, although the third should be used rarely. Headings are not numbered, so please do not reference other sections by heading number in your article, even if you use them.

Illustrations

You must supply images as separate files, as well as placing them in your article to show approximate location. The journals are printed in black and white, so please make sure that your images do not rely on colour alone to differentiate between the components. Also check that any lettering can be seen when the image is converted to greyscale and it has been scaled to fit on a page. (Note: Colour images are useful, as a PDF version is produced in colour).

- Supply images in common formats such as JPG, TIFF, SVG, or PNG.
- Photographs and similar images must be a minimum of 200 dpi, ideally 300 dpi.
- Save diagrams and line drawings in vector format. Do not attempt to convert raster images to vector format.
- Screenshots should be saved in a non-lossy format. TIFF and PNG have been used successfully. GIF is also a suitable format, but check that the colour depth supported by your application has not adversely affected the image.

References and notes

CVu and Overload treat references and footnotes differently. Please ensure that references are as complete as possible.

- CVU: References and notes are marked by placing numbers in brackets in the text; for example: [1]. References and notes are both numbered in the same sequence and the corresponding information provided in a Notes and References section at the end of the article.
- Overload: Notes are treated as footnotes - number these and supply the corresponding text either as footnotes or at the end of the article. References are marked in the text using the author's name and final two digits of the year of publication (for example, [Griffiths06]) and the full reference is provided in a References section at the end of the article. Sometimes it is not possible to provide an author or year: in this case choose a suitable word. For example, if you are referencing the Boost libraries, you may use [Boost] as the reference marker. ■

Cambridge Meeting

by Ric Parkin

The Cambridge local group have held another two well attended meetings, hosted once again at DisplayLink's offices, followed by much discussion in a local pub.

In October Russel Winder presented 'Closing the Case for Groovy (and Ruby, and Python)', which looked at the idea of closures (and their subtle varients) in various languages and showed how they can radically change your programming from the common imperitive style. Some facinating examples that generated HTML were shown, which illustrated the points and really made people think of alternatives to their current approaches.

In November Allan Kelly talked about 'Agile software development – where to begin'. This concentrated on the developer's viewpoint, including why you might want to be more agile, which practices you might try to introduce, and how to get started without having to change everything at once. He also touched on the effects on other parts of an organisation that becoming agile will cause, and how to survive them.

Sponsored Conference Places

by Giovanni Asproni

The ACCU Conference is a fabulous, entertaining and stimulating place to be. It is, without doubt, the jewel in the ACCU's crown. Nowhere else can you meet so many interesting and notable people in our field.

It should, therefore, be on every techie's calendar. Not everyone, however, can attend. For some people it is too far away or they are too busy; for others there are financial barriers. To encourage this last group, four one-day tickets for the 2008 conference are on offer.

The conditions are:

1. The candidate must be proposed by a full ACCU member who is expected to accompany the candidate for that day.
2. The candidate must not have been to an ACCU conference before.
3. The candidate must be at least 18 years old.

Application process

Any ACCU member wishing to put someone forward for a place should email accuplaces@oxyware.com before Jan 31st 2008 with a short description (maximum 250 words) of why they believe the candidate would benefit from coming to the conference. After the closing date, the four successful candidates will be informed by the end of February and the remainder will be placed on a waiting list.

Conditions

1. The offer covers only the entrance to the conference. Travel, accommodation and subsistence are not covered.
2. In the event of a candidate not being able to take up a place, the place will be offered to those on the waiting list.
3. In the event that the proposer cannot accompany the candidate, they may either find a substitute ACCU member to do so or offer the place to the waiting list.

New ACCU Group on LinnkedIn

by David Carter-Hitchin

For those who don't know http://www.linkedin.com/ is a popular contact website for work colleagues to stay in touch with each other, to recommend colleagues for new appointments and to win the hearts and minds of potential recruiters; somewhat like Friends Reunited for old work mates. The basic idea is that you build up a network of friends and colleagues, and you can also see your friends' networks, and your friends' friends' networks and so on. There are a number of protocols in place to ensure that people don't get any unwanted communications and so forth – for example if you want to contact a colleague's old colleague, you need to pass the message through your immediate colleague. Also the service has its own messaging system, so you don't need to declare any of your own e-mail addresses. You get to choose which parts of your profile are public and so on and so forth. In short LinkedIn is a useful resource when it comes to the corporate and professional workplace. So why do we in the ACCU care about all this? Well, LinkedIn is useful in its own right, so it's worth signing up for anyway, but recently Allan Kelly has added the ACCU as a group to the service, so you can now show that you are a member of the ACCU on your profile. The advantage of doing this is that other members of the ACCU will be more readily identifiable to one another, and also, and most importantly, it will advertise the existence of the ACCU to the work community as a whole. Notice I didn't specify which work community, as in theory LinkedIn is global and spans all industries, but it is heavily populated by geeks! Therefore, the more people that get their details registered and signed to the online ACCU LinkedIn group, the more chance we will spread the good word of the ACCU to people who are likely to join. It is an extremely good place for us to advertise our association – for one it's free, for another it's generally to likeminded people.

So, what are you waiting for? Fire up your favourite browser, and visit http://www.linkedin.com/, if for nothing else but in the interest of promoting our wonderful organisation. In Allan Kelly's words, here's how you join:

Visit http://www.linkedin.com/e/gis/1908/6123922B178D

Provided you use the ***same e-mail address*** for LinkedIn and ACCU membership you should be automatically accepted. Once you have joined the ACCU badge should appear on your profile and you should be able to contact other LinkedIn ACCU members through the system. If you have a different e-mail address then you can just add it to 'account&settings' (top of LinkedIn page) and Personal Information/ Email Addresses. Finally, once you've joined the ACCU group, be sure to make it publically visible on your profile (providing you are happy with this, but why ever shouldn't you be?!)

We have chosen to set up this group for several reasons:

- Several people requested such a group
- The ACCU badge will raise the profile of the ACCU
- ACCU members will have another route to contact one another
- Both the ACCU and LinkedIn are increasingly being used by recruiters and many ACCU members find this useful.

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.

Moving to Free Software

By Marcel Gagne, published by Addison Wesley, ISBN 0-321-42343-7

Reviewed by Ian Bruntlett

I've been putting free software onto a mental health charity's (http://www.contactmorpeth.org.uk/) client's PCs for nearly two years now and shipped about a hundred free PCs to them. I've personally built up a small library of useful programs and so I looked forward to reviewing this book and discovering new F/OSS gems.

This book comes with a DVD of F/OSS programs and, in general, it dedicates a chapter to each program. Some exceptions are 1) all the games are bundled together into one chapter and 2) OpenOffice isn't completely covered (but it does give chapters to Writer, Calc, Impress and Base).

To cut a long story short, here are the subjects and relevant programs that come with the book.

- Internet: FireFox (web browser), Thunderbird (email client), Gaim (IM), Skype (VOIP)
- NVU (Web Site Design)
- OpenOffice.org: Writer (word processor), Calc (spreadsheet), Impress(similar to MS PowerPoint), and Base (similar to MS Access)
- Audio: CDex (CD Ripper and Audio Converter), Audacity (Podcasts), Juice (Podcasts)
- Graphics: GIMP (like MS Paint), Inkscape (vector graphics), Scribus (DTP)
- Utilities : 7-Zip (compressing files), SpyBot (anti-spyware), ClamWin (antivirus)
- Linux : Ubuntu Linux. One of the easier Linux distributions available.

To finish it off, the following games are provided: PlanetPenguin Racer, FreedroidRPG, Armagetron Advanced, Super Tux, BZFlag, Fish Fillets: Next Generation, Neverball and Neverputt, SolarWolf and Flightgear.

So that's what the book is good at. What are its weaknesses?

This book should mention:

- which versions of Windows the programs are compatible with
- what are the minimum hardware requirements
- what other programs do they rely on (GIMP relies on GTK)

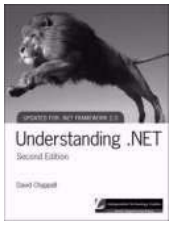
The author really should have some way of enabling readers/users of this book to contact the author to tell him about incompatibilities and share workarounds – possibly a web site with a forum. Don't expect to give this book to a non-technical friend and have all the programs work. You'll need to do some technical handholding and an internet connection is definitely needed to download apps that aren't fully on the disk (the DVD has GIMP on it but not the GTK toolkit that GIMP requires).

Verdict: Recommended.

Understanding .NET

Second edition, by David Chappell, published by Addison-Wesley

Reviewed by Albrecht Fritzsche



Are you one of those people always wondering what .NET actually means? What all these buzz words like ADO.NET, ASP.NET, etc. stand for? How to get a first understanding of the capabilities of Microsoft's framework?

Then this book might be the right choice for you – providing answers to all those questions and giving a very good overview in just 300 pages.

The very concise style, e.g. during the introduction you get an understanding why a VB program will exhibit roughly the same performance as a similiar C# program, makes this book a good choice for professional developers and managers wanting to gain an initial understanding of .NET, its fundamentals and capabilities.

In the following chapters you will be taken on a tour through the Common Language Runtime, some .NET languages like VB, C#, and C++/CLI, the framework's class library, ASP.NET, ADO.NET and distributed applications. The given examples are always complete and still rarely take up half of a page. The whole edition is updated for .NET framework 2.0.

The book is clearly structured, and the lucid style of the author is easy to read. An exceptionally good idea seemed to me that all subjective comments of the author are clearly separated from the rest of the book in greyed-out boxes, reaching from 'Are generics worth it?' and 'Is C# just a copy of Java?' upto 'The revenge of hierarchical data' and 'The short happy life of .NET My Services'.

All in all this book serves as a good and concise introduction into this huge framework and gives you within days a good first impression of what .NET actually means.

Recommended.

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

View From The Chair

Jez Higgins
chair@accu.org

A motion from the floor of this year's Annual General Meeting instructed 'the committee to produce a document detailing the financial management of the association to the 2008 AGM'. In the discussion around the motion, it was stated by the proposer that the document was not expected to exceed one side of A4 paper.

We are not yet at the 2008 AGM but the current draft stands at well under a side of A4, so it seems sensible to present it a little early. If you would like further details or clarification, please contact me so that the draft can be revised for the AGM.

> The Society instructs the committee to produce a document detailing the financial management of the association to the 2008 AGM.

The vast majority of the organisation's income is in the form of membership subscriptions. Aside from a few corporate members who pay by cheque, subscriptions are paid directly into the ACCU account by WorldPay or by members' individual standing order. There is a significantly smaller income from advertising in CVu and Overload.

The single biggest outgoing is the cost of producing CVu and Overload: production editing and pre-press, printing, and postage. These are, essentially, on going costs taken on and approved historically. There are a small number of other, much lower value, expenses that the Treasurer is instructed to pay until further notice, such as our annual data protection registration renewal.

All other spending decisions are approved by the committee as they arise. Any discussion usually occurs at a committee meeting, but can also take place on the committee mailing list.

The discussion and decisions arise either as the result of a request or suggestion from a member ('it would be a good idea if ...'), from a request within the committee ('can we have some money for ...'), or by some external event ('argh! we need a new production editor').

Most of these spending decisions are for fixed sums, for example a one-off room hire fee for a local group. Less often a committee member is given a budget within to negotiate, as when, for instance, a new production editor was hired.

Day to day management of the bank accounts rests with the Treasurer.

All payments are made by cheque, requiring two signatures. Like the Queen, ACCU never carries cash.

Membership Report

Mick Brooks
accumembership@accu.org

At the AGM earlier in the year, the membership approved the publication of the ACCU handbook as an online version, which will replace the traditional printed paper one. We are now preparing to produce this online edition, which in the first instance will take the form of a PDF file. This file will be made available for download on the ACCU website, exclusively to full members of the association.

A large fraction of the membership have previously given permission for details of their membership, including their contact information, to be displayed in the paper handbook which was distributed to all members. This makes the handbook a useful tool for members who'd like to communicate with other members, particularly those that live and work in their local area. We appreciate that members may have a different attitude to online publication of these details, and we'd like to give you plenty of time to remove your consent before the new handbook goes live.

I urge all members to login to the ACCU website at http://www.accu.org and review their contact preferences. They are accessed by clicking the Account link (which appears in the Login panel), and then, the ACCU Subscriptions tab.

There you can review and edit the all of the information recorded about your membership. The setting which determines whether you will appear online is labelled 'Name in Handbook?'.

The new handbook will not be produced until the 1st of February, giving you plenty of time to update the setting if you'd prefer not to appear. If, on that date, the setting for your account is ticked, then your details will appear in the next handbook. If you have any problems updating the setting, or have any questions about the new handbook arrangements, please contact me at accumembership@accu.org.

Publicity Officer Report

David Carter-Hitchin

It's been a busy few months for me since the conference for all the wrong reasons; I've finished a part-time degree and changed job. Not that this is a care of yours, but this means I haven't spent as much time as I would have liked to on publicity, but that is set to change. The first task has been to draft a letter introducing the ACCU which will be sent to every educational institution in the UK which offers some kind of software related (even vaguely related) award – be it a degree or diploma, computer science, engineering, physics or games programming or computer graphics. Actually thinking about it, it would be good to see more representation from the graphics people – I'd like to see the odd article in CVu about Phong shaders, radiosity and so on! If I'm wrong and you're already out there, then please write something! After the universities have been addressed, then next on the list will be software houses and any corporate and banks with a large computing requirement. After that, I'll be working on building up some mutual arrangements with other conferences. Allan Kelly has recently done a brilliant job of creating an ACCU group on LinkedIn (http://www.linkedin.com/), which will be superb free promotion of the ACCU – sign up for this now. For more information, see the article on this on page 33 of this edition. Hats off to Allan. Thanks must go to the various people who have sent me the odd suggestion by e-mail, notably Ian Bruntlett, who has rekindled an arrangement that we used to have with O'Reilly publishers. I'm not sure this will result in any significant promotion of the ACCU by O'Reilly, but it will at least mean that we can get all their books to review before they hit the shelves in Amazon. Even if this results in just one link to our reviews, then it will have been worth it. Keep those ideas coming. Finally, don't forget to mention the ACCU to all your friends and colleagues, and put a link to the ACCU website on your e-mail signatures.