The magazine of the ACCU

www.accu.org

{cvu}

Volume 22 Issue 1 March 2010 £3

{cvu}

Volume 22 Issue 1
March 2010
ISSN 1354-3164
www.accu.org

Features Editor
Steve Love
cvu@accu.org

Regulars Editor
Jez Higgins
jez@jezuk.co.uk

Contributors
Ian Bruntlett, Pete Goodliffe, Paul Grenyer, Richard Harris, Jex Higgins, Alan Lenton, Phil Nash, Roger Orr, Terje Slet Dáire Stockdale

ACCU Chair
Jez Higgins
chair@accu.org

ACCU Secretary
Alan Bellingham
secretary@accu.org

ACCU Membership
Mick Brooks
accumembership@accu.org

ACCU Treasurer
Stewart Brodie
treasurer@accu.org

Advertising
Seb Rose
ads@accu.org

Cover Art
Pete Goodliffe

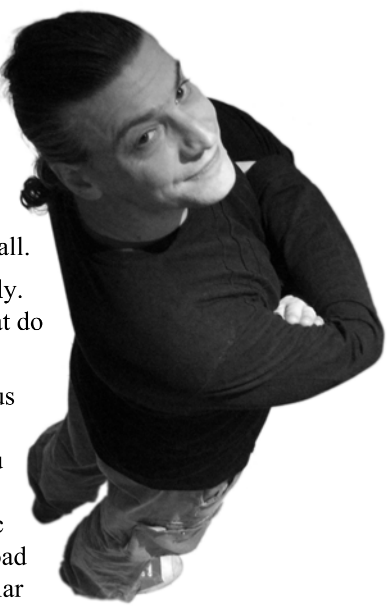
Repro/Print
Parchment (Oxford) Ltd

Distribution
Able Types (Oxford) Ltd

Design
Pete Goodliffe

accu

By Convention



So much about the way we interact with the world around us is driven by convention. Sometime those conventions have 'side-effects', for example driving on a particular side of the road. In a country where it is conventional to drive on the left, it's also – by implication – conventional to drive around traffic islands clockwise. Going against convention has consequences ranging from mild to severe, but you'd be very lucky indeed to escape without any consequences at all.

Some conventions allow us to operate almost automatically. Doors without handles are pushed. Doors with handles that do not ***pull*** often cause a moment of confusion.

Having to go against our familiar conventions also gives us pause. Having to drive on the right hand side of the road when you are accustomed to driving on the left means you have to concentrate a bit more, and pay attention to the implied things like going counter-clockwise around traffic islands. Even as a pedestrian, when crossing a two-way road you naturally and automatically expect traffic in a particular lane to be travelling in a particular direction, and when cars are on the 'wrong' side of the road, crossing that road takes a bit more thought.

So it is with programming. When we read code in a particular language, we naturally and automatically make assumptions about its behaviour according to the idioms of that language. For example, in Python member functions, it's conventional to use `self` to refer to the current object instance. It's not enforced, but it would cause most Python programmers significant confusion if you were to write `this` instead, even though it's entirely legal to do so.

Deliberately going against convention can have positive consequences, however. When it was introduced by Alexander Stepanov, the C++ STL went against a huge body of accepted wisdom for container and algorithm libraries that had gone before it.

Such paradigm-shifts are the result of ***questioning*** the accepted conventions, and challenging them. You can't just go out and drive on the wrong side of the road...or remove all handles from doors that only push, either!

Steve

STEVE LOVE
FEATURES EDITOR

The official magazine of ACCU

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

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

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

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

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

DIALOGUE

REGULARS

FEATURES

SUBMISSION DATES

C Vu 22.2:	1st April 2010
C Vu 22.3:	1st June 2010
Overload 97:	1st May 2010
Overload 98:	1st July 2010

WRITE FOR C VU

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

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

ADVERTISE WITH US

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

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

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

COPYRIGHTS AND TRADE MARKS

Live to Love to Learn (Part 2)

Pete Goodliffe continues his journey of self-improvement.

Learning is like rowing upstream: not to advance is to drop back.

~ Confucius

In the previous instalment of this column we began a journey of self-improvement. This is an unrepentant exploration of a 'soft skills' topic, one that is very often radically under-appreciated in our field.

In the last issue we considered the importance of learning to our craft. Software developers are in the knowledge business. We're not mechanical automatons, doing the same thing over and over. We're constantly solving new problems and facing new challenges. We're continually acquiring information, using information, and generating information. We must therefore be able to learn constantly, or it's Game Over. We must ensure that we know *how* to learn.

We also thought a little about *what* to learn. This list includes technology and technical skills, as well as softer skills (e.g. team-working). It also includes more 'left-field' non-programming topics; we need a broad world-view to inform our technical decision making processes. Learning basket-weaving might just give you valuable insights into your day job!

The facts of life

Here are a few facts that provide some useful context to the world of learning.

Learning is a basic human skill. We all do it. We just do it in different amounts. Small children are constantly learning and amassing new skills. Language, cause-and-effect, and social skills are picked up rapidly from birth. As are basic physiological capabilities like eating, walking, and continence. (Although continence might also be considered a social skill of sorts.)

When it arrives freshly formatted, the human brain rapidly absorbs information and develops skills across a wide range of experiences. It's a remarkable device, both intricate and powerful, and we need to learn how to best use it. Unfortunately, it doesn't come with a User's Manual. (Even if it did, most men wouldn't read it, anyway).

Our learning is often too narrowly focused. Infants absorb information in many disciplines rapidly. And simultaneously. They don't know that they're simply not supposed to. But all too soon, our training starts; we're introduced to the thought police.

Or, rather, at school we hop on board the academic train and our incredible generalised learning capability is slowly frittered away. At primary school we are taught a range of topics. At secondary school these topics are formalised and compartmentalised somewhat. As we progress through the academic system we cut out certain topics and concentrate on those that suit us best (because of interest or aptitude). At (UK) GCSE level, we narrow the focus to about 8 subjects. We progress to A-Level and narrow that focus to 3 or 4 subjects. Then we progress to university level where we focus on a single subject.

This increasing focus in learning is useful in that it allows us time to become highly proficient in one area, and to devote time to practise the specific skills required to master that subject. It also allows us to chose a single topic that excites and interests us: surely a prerequisite for effective learning.

But often such high specialisation is detrimental to your ability to learn in general. It ingrains a lack of curiosity for other disciplines or any appreciation of the wider human experience. However, it is when disciplines collide that we often see the the most exquisite insights (consider, for example, how Alexander's architectural work illuminated the field of software design [1]).

Learning is hard. When an infant learns, the world is new and exciting; as they coast through it they just absorb stuff. After a certain point, the things you have to learn become less interesting. Adults force you to learn stuff. It involves effort. Effort which you could expend on other tasks, like playing with your friends, or your toys. Then you're given homework and learning starts to take over your precious 'free time'. It feels like hard work. It is hard work.

Learning is hard.

That hard work will, though, be rewarded. But it's easy to resent learning as a tedious chore that stands between us and 'fun' or 'ability'. Many people's early school experience of learning still colours their adult life approach to learning, even if they don't realise it. How often have you balanced the reward of learning something against the effort involved? How often has the initial learning effort put you off learning something new? Are you now automatically programmed to avoid areas you know nothing about because it is uncomfortable to appear ignorant?

Have your past experiences coloured the way you approach learning?

We must learn how to learn. Many people were spoon-fed information in school, and told not to criticise. This instils a bad attitude to learning; where it is something someone gives you, or that a teacher does to you. Learning is actually a personal process that you have to take individual responsibility for.

Do you take personal responsibility for your own learning?

Don't blithely accept what you're told. Keep your brain engaged. It's dangerously easy to fall into the trap of believing what you're reading or being told is the gospel truth, rather than something that should be evaluated.

- How many people read things on the internet and presume that they are true? Wikipedia has a genuine air of authority, yet contains many remarkable mis-truths. Some of the tubes of the internet are rather grimy.
- How many people read things in their newspaper and unquestioningly presume that they are correct? True, there are lunatics out there on the web, but paid journalists wouldn't make things up, would they? *Wrong*: there are plenty of lunatics in the media, too. If you believed everything you read in the newspaper then you wouldn't eat or drink anything for fear that it would given you an incurable disease.
- If it's in a textbook, written by an academic, peer reviewed, and recommended by a prestigious university then it must be correct. But consider who paid to write the book, or who funded the research. What are their motives driven by? Many things presented as fact in the most prestigious journals are, in fact, opinion and conjecture backed up by slim arguments. For example, 70% of

PETE GOODLIFFE

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

everything I've written up to this point was made up. Including that statistic.

Be aware that you can learn the wrong thing and believe it's the right thing. This can be at best embarrassing, and at worst dangerous. This is illustrated by the *Four Stages of Competence* (a classification posited in the by 1940s by psychologist Abraham Maslow). You may have:

- **Conscious incompetence.** You don't know something. But you know that you're ignorant. This is a relatively safe position to be in. You probably don't care – it's not something you need to know. Or you know that you're ignorant and it is a source of frustration.
- **Conscious competence.** This is also a good state to be in. You know something. And you know that you know it. To use this skill you have to make a conscious effort, and concentrate.
- **Unconscious competence.** This is when your knowledge of a topic is so great that it has become second nature. You are no longer aware that you are using your expertise. Most adults, for example, can consider walking and balance as an unconscious competence – we just do it without a second thought.
- **Unconscious incompetence.** This is the dangerous place to be. You don't know that you don't know something. You are ignorant of your ignorance. Indeed, it is very possible that you think you understand the subject but don't appreciate how very wrong you are. It is a *blind spot* in your knowledge.

So be very careful when claiming competence in any subject, let alone any level of expertise.

Understanding the brain

Psychologists, biologists, and other similar evil geniuses have studied the human brain for years, and have formulated many models of brain behaviour and models of learning. There is an overwhelming quantity of research on the subject (some quite contradictory).

Understanding (at least at a basic level) how the brain works, and some of its characteristics will help us work out how to learn most effectively.

The left brain/right brain split. We hear a lot about the 'left-brain' and the 'right-brain'. These terms have entered pop culture, and you are as likely to hear them discussed by artists and CEOs as by psychologists. It comes from research (pioneered in the 1970s by Roger W Sperry who coined the left/right brain terminology) discovering that our mental activity is split between the two hemispheres of the brain; either side controlling a different mode of thinking [2]. The left hemisphere tends to govern analytic and logical activities (language, symbolic representation, rational deduction, the linear thought process) whilst the right hemisphere performs non-linear processes (characterised as non-rational, spatial, holistic, and identifying spatial relationships).

The common view is that people rely more, or tend towards using, one side of their brain rather than other. Artistic, creative types favour their right brain; academics and office-workers favour the left. Apart from reinforcing a social stereotype, this is not strictly accurate since the split of brain activity is not pure laterally biased. Several psychologists have since tried to come up with other terms, but popular culture has latched on to left- and right- and we're stuck with them.

The common view is that people rely more, or tend towards using, one side of their brain rather than other

Both parts of our brain and both modes of thought are essential. In order to think and learn effectively we must be able to bring both 'sides' of our brain into use. Since programmers tend to lean towards the left-brain mode of operation, we must learn to introduce more right-brain thinking into our regimen; this requires us to find ways to 'dial down' our left brain activity to give the other side a fighting chance.

We'll see some ways to do this later. There's a lot more that could be said on this subject; it is a huge topic, and well worth investigating more if you are interested.

Be aware that you can learn the wrong thing and believe it's the right thing. This can be at best embarrassing, and at worst dangerous.

Personality type affects learning style. If you favour the right-brain mode of thinking you will learn best when presented with patterns and an holistic view of a subject, rather than a serial stream of information. You'll prefer to make associations and understand overarching themes. If you favour the left-brain you want a linear rational presentation of the topic. You'll prefer to assimilate facts than to have a grand story told you.

Clearly your brain wiring has a radical effect on the most effective style of learning for you.

There are many models of personality type, perhaps the most famous being the Myers-Briggs Type Indicator (MBTI) [3]. MBTI classifies your personality along four axes. Your personality type will effect how you associate knowledge, and how you best learn. One MBTI axis is Introvert-Extrovert. Introverts would rather learn on their own – they need private mental space to work things through. Extroverts learn well in groups where they can discuss and feed off the ideas of others. Another interesting axis is Sensing-Intuition. The Sensing personality is the classic left-brainer who emphasises facts and details. They must complete one task before moving on. Intuitive people are right-brainers who use their imagination more. They can move on without completing a task or before they understand everything about a subject.

Understanding your particular personality type (there are many MBTI tests available on the web, for example) will reveal specific ways to make your learning routine maximally effective.

Understand how you learn best.

Memory fades. There was a time when psychologists believed that a memory, once made, was permanent. However this is not the case: a certain enzyme (PKMzeta) is required to keep synaptic connections valid. Lose the enzyme, you lose the memory! [4]

Our memory is also fallible in other, less chemical, ways. The brain isn't perfect. You'll notice that old memories can change and distort in your mind. (So it's probably not true that things aren't as good as they used to be.) Memories and opinions can easily warp as we recollect them, to fit our current preferences or preconceived notions. We can very easily implant our own false memories (unconsciously) or become subject to false suggestion from others.

In order to keep your memory active, it must be refreshed – constantly read and exercised. If you don't need and employ a skill then you will use it. [5]

Use your knowledge. Or you'll lose it.

Memory grows. It was originally thought that mental capacity decreased over time; that humans start off with a fixed number of brain cells and over the course of a lifetime this number decreases (due to the ageing process, damage caused by trauma, or other abuse – like excessive alcohol intake).

This was determined by scientists studying animals in laboratory cages; they saw no sign of *neurogenesis* – the growth of brain cells. But this was simply because a brain devoid of stimulus need make no new connections.

The test subjects' brains didn't expand because there was no need to; life in a cage stunts brain growth. That's a pretty damning indictment of modern cubicle working!

In the early 1990s, psychologist Elizabeth Gould discovered that in suitable conditions (an environment with stimulation and opportunities to interact and learn) the brain is perfectly capable of growing neurons and making new connections [6]. Other subsequent studies have drawn links between exercise and increased brain growth.

So that's good news – you have a practically limitless ability to expand your mind. And the scientists have proved that you need stimulation do to so! Break out of your cubicle, and feel free to drink a beer!

Mental state effects learning. Factors such as stress and a lack of sleep will clearly contribute to an inability to concentrate, and so will degrade your ability to learn. Your mental attitude also dictates how well you will learn. Psychologist Carol Dweck's research [7] shows that students who believed that they couldn't increase their knowledge were not able to do so. Those who believed they could increase their mental capability were easily able to.

Believe in your ability to learn.

People who enjoy learning naturally learn more. People who want to learn, learn more. Is this the power of mind over matter? Perhaps it is; and it's something we need to exploit. I've long argued that our attitude effects the quality of our work. This is the central theme of my book *Code Craft* [8].

Learning models There are a number of very illuminating models of learning that have been constructed by educational psychologists. The Dreyfus Model of Skill Acquisition is a particularly interesting example, postulated by brothers Stuart and Hubert Dreyfus in 1980 whilst they were working on artificial computer intelligence [9]. After examining highly skilled practitioners in fields such as airline pilots and chess grand-masters, they identified five specific levels of understanding:

a revealing framework to understand where you currently stand in the mastery of a topic, and help determine where you need to get to

- **Novice** A complete newbie. Novices want to get results fast, but have no experience to guide them in doing do. They look for rules they can follow by rote, and have no judgement to tell whether those rules are good or bad. Given good rules (or luck finding suitable resources on Google), novices can get quite far. Novices have no knowledge of a topic (yet).
- **Advanced beginner** At this level, some experience that has lead to learning; you can break free from rules a little and try tasks on your own. But perception is still limited, and you'll get stuck when things go wrong. At this level there is a better understanding of where to get answers (you know the best API references, for example) but you are still not at a level where you can comprehend the bigger picture. The beginner can't focus out irrelevant details, as far as they're concerned everything and anything could be important to the problem at hand. Beginners rapidly gain *explicit knowledge* – the kind of factual knowledge than can be easily written down and articulated.
- **Competent** This stage sees you with a mental model of the problem domain; you've mapped the knowledge base, and have begun to associate parts and understand the relative importance of different aspects. This big picture view allows you to approach unknown problems, and plan methodical routes into those problems rather than diving in and hoping rules will get you to a solution. This is a good place to be.
- **Proficient** people move beyond competency. They have a much better understanding of the big picture, and are frustrated with the simplifications that the novice needed. They can correct previous errors and reflect on their experiences to work better in the future. At this point you can also learn from other's experiences and assimilate them into your body of understanding. Proficient people can interpret *maxims* (as opposed to simplistic rules) and apply them to a problem (e.g. they know how and when to apply design patterns). Now it is easy to identify and focus only on the issues that really matter, confidently ignoring irrelevant details. Here we see the person has gained significant *tacit knowledge* – knowledge that's hard to transfer by exposition, that is only gained by experience and deep understanding.
- **Expert** This is the pinnacle of the learning tree. There are very few experts. They are authorities on a subject; they know it completely, and can use this skill interlinked with other skills. They can teach others (although they probably teach competents better than novices as there is less of a disconnect). Experts have intuition, so rather than needing rules they naturally see an answer, even if they can't articulate why it's the best solution.

Why is the Dreyfus Model interesting? It's a revealing framework to understand where you currently stand in the mastery of a topic, and help determine where you need to get to. Do you need to be an *expert*? Most people are *competent* and this is satisfactory (indeed, a team of experts would be far too top-heavy, and probably dysfunctional).

It also illustrates how you should expect to be solving problems at each stage of your learning. Are you looking for simple rules to apply, hungrily gathering maxims to draw experience from, or intuitively sensing answers? How much of a 'big picture' view do you have of the topic?

The Dreyfus model is also a very useful aid for teamwork. If you know where a colleague sits on the notice-expert spectrum you can better tailor your interaction with them. It will help you learn how to work with other people – should you give them some simple rules, explain some maxims, or leave them to weave new information into their broader understanding?

Note than the Dreyfus model applies *per skill*. You may be an expert in a particular topic, and a complete notice in another. This is natural. And should also be a source of humility – even if you know all there is to know about Behaviour Driven Design, you may no nothing about the Joe Bloggs Test Framework. It should excite you that there is something more to learn that may enhance your expertise in BDD, whilst keeping you humble that you aren't an infallible expert in every subject! No one likes a know-it-all.

Shu-Ha-Ri is another interesting model of learning that originates in Japanese Martial Arts. It is a set of three terms describing the stages a student goes through to to achieve mastery in their discipline, roughly translating to Learn, Detach, Transcend. In the *Shu* phase, the student studies under one teacher; he does not understand enough to mix concepts from different streams, but aims to imitate the teacher precisely. He must build up experience in the field, absorbing the teacher's explicit and tacit knowledge. Once he has a full understanding of the techniques, the student reflects on them in the *Ha* phase. The knowledge he has learned will not change, but he can interpret, reflect and explore the deeper meaning. Finally, the student progresses to *Ri*, where he moves beyond studentship; he has original thoughts, and becomes a master himself. He now has the skills and experience to innovate.

Heed this model. Don't try to transcend and innovate until you have completed the *Shu* phase of your learning (remember to be aware of unconscious incompetence) and have completed the *Ha* phase of understanding the knowledge.

Conclusion

These are key factors in working out how to learn most effectively. Let's review them again:

- **Learning is a basic human skill.** We can all do it. We do it all the time.

Of Lag, Throughput and Jitter

Alan Lenton gives a glossary of terms.

Over the last few years quite a number of people have asked me to explain the difference between latency, throughput and jitter. I was surprised at how many of them were programmers <cough, err software engineers>, but thinking things through, about that's probably not surprising because most programmers don't usually end up at the sharp end of networking. So, here for your edification is the Alan Lenton guide to latency, throughput and, as a special bonus, jitter!

Perhaps the best way to explain it is using an analogy. Suppose we have a wedding party, The bride and groom have exchanged their vows, the photographs have been taken and it's time to travel to the reception. There are 24 limos to take the party to the reception. Everyone gets in. The cars line up behind the happy couple's car and it's time to move off.

Now, as it happens there is a single lane highway that just happens to go from where the party are now, to the hall where the reception is being held. As luck would have it, the highway is empty and the cars are able to proceed single file at top legal speed (70 miles/hour where I come from) to the reception. The time it takes each car to get to the reception (let's say 16 minutes) is the latency.

Of course, getting the whole party there takes longer than that, because the cars are spaced out for safety, so let's say it takes another 12 minutes between the time the first car arrives and the last one discharges its passengers. This is the throughput – two cars/minute. So in this scenario it takes a total of 28 minutes from the time the first car departs to the time the last car pulls in, the passengers get out and we can toast the happy couple with Verve Clicquot NV champagne.

OK – now let's look at a scenario where the road-making machines have been busy during the wedding ceremony and have widened the highway to a two lane affair (fast machines and slow grooms). This time the cars line up two abreast and set off. It still takes the same time for the first cars to get there, 16 minutes, but since they are arriving two at a time it only takes half the time, six minutes, between the first and last car's arrival – giving us four cars/minute. That makes a total of only 22 minutes for the whole caboodle.

Of course, if we had a six lane highway we could get all the cars through in two minutes bringing the total time down to 18 minutes. That's probably as much increase in throughput as is worth doing at this stage since the time cost of the latency is now much greater than that of the throughput.

So how would we reduce the latency? Well, we could try and reduce the distance travelled – take out all the bends in the highway and have it made into a straight line between the start and the finish. That's probably a little excessive, so the most obvious way to reduce the latency is to speed up the cars. So let's assume that there are no highway patrols and no speed cameras so we can double the speed. That means that it

ALAN LENTON

Alan is a programmer, a sociologist, a games designer, a wargamer, writer of a weekly tech news and analysis column, and an ocassional writer of short stories (see http://www.ibgames.com/alan/crystalfalls/index.html if you like horror). None of these skills seem to be appreciated by putative employers...

Live to Love to Learn (Part 2) (continued)

- **Our learning is too narrowly focused.** Consider a wider sphere of reference. Draw inspiration from other fields.
- **Learning is hard work.** It doesn't come for free. But you don't get a reward without an investment.
- **Learn to learn.** That's why I'm writing this series! The most effective learning is done on purpose and planned to maximise effectiveness.
- **The left brain/right brain split.** We need to employ all of our available brainpower to make our learning maximally effective.
- **Personality type affects learning style.** Understand how you best learn and exploit that to your advantage.
- **Memory fades.** Unless you refresh your knowledge by using it, your memory of it will become unreliable.
- **Memory grows.** Constant stimulation is required to maintain growing mental capacity.
- **Mental state effects learning.** Cultivate a positive attitude towards learning. Otherwise you may as well not bother.
- **Learning models.** Certain models can help us classify levels of expertise and plan how to gain it.

When I set out to write this column I intended to talk about a number of quite specific techniques that will help us to learn better, and explain the above topics briefly as background material to justify the techniques. As you can see, I've hardly gone into any detail and yet I've filled many inches of magazine space.

So, in the next instalment, I promise we'll look at some very practical techniques to improve our learning capabilities.

References

[1] *Design Patterns: Elements of Reusable Object-Oriented Software*. Gamme, Helm, Johnson and Vlissides, Addison-Wesley, ISBN: 0201633612 and *A Pattern Language: Towns, Buildings, Construction*, Alexander, ISBN: 0195019199

[2] 'Lateral specialization in the surgically separated hemispheres' R.W. Sperry in *Neurosciences Third Study Program*. F. Schmitt and F. Worden (Eds.), Cambridge: MIT Press 3:5-19 (1974).

[3] http://en.wikipedia.org/wiki/Myers-Briggs_Type_Indicator

[4] 'Brain Researchers Open Door to Editing Memory' in *New York Times*. http://www.nytimes.com/2009/04/06/health/research/06brain.html?_r=1

[5] Use it or lose it

[6] 'Neurogenesis in the Neocortex of Adult Primates' Elizabeth Gould, Alison J. Reeves, Michael S. A. Graziano, Charles G. Gross in *Science* 15 October 1999:Vol. 286. no. 5439, pp. 548 – 52.

[7] *Mindset: The New Psychology of Success*, Carol S Dweck, Ballantine Books. ISBN: 0345472322

[8] *Code Craft: The Practice of Writing Excellent Code*, Pete Goodliffe, No Starch Press. ISBN: 1593271190

[9] *A Five-Stage Model of the Mental Activities Involved in Directed Skill Acquisition*, Stuart E. Dreyfus and Hubert L. Dreyfus, Storming Media. (Available from: http://handle.dtic.mil/100.2/ADA084551)

only takes eight minutes for the first cars to arrive – if you happen to have a six lane highway, the total time comes down to ten minutes from start to finish. That's not bad!

And jitter? Now this is interesting. In this case, imagine everyone has been sloppy about setting their cruise controls so that the speeds of the individual cars are slightly different. Not a lot different, just a tiny difference. This means that by the time the cars arrive they are just a little bit out of formation and the time between the arrival of each car is slightly different. We call this difference jitter. If the jitter is really bad, some cars might arrive out of order, and those of you who have attended weddings will know that this is serious, and can cause undying feuds...

So we now have latency, throughput and jitter. How do these apply to the Internet?

Well, modern networks divide the stuff you are sending (or receiving) into packets of data (the cars) and send them off to their destination. Network latency is considered to be the time it takes for you to send a packet and to receive an answer (a round trip, rather than the one way trip we used), throughput is considered to be the amount of data you send each second - usually measured in bits/second – and jitter is a measure of the difference in time between the arrival of packets.

And how does it affect you?

Well, if you are playing an online game the most important factor is latency (lag in the common game parlance). The longer it takes for you to get a response to your command from the game, the more difficult and frustrating it is to play. Of course, the acceptable lag/latency is different for different types of games. For a turn-based game a lag of several seconds will not make any appreciable difference, but for an on-line shoot 'em up, even a few tens of milliseconds can be too much. A lagged combat flight sim will have you shooting at opponents who are no longer actually in sight – even though most flight sims are slowed down by a factor of at least four!

The normal way to handle this is to use a probabilistic method of determining whether the bullet has hit. The easiest way to visualise it is to think of a cone of probability of bullets hitting, rather than a stream of single bullets. Unless of course you are lazy, in which case you offload the work to the client, instead of doing it at the server. You will then find yourself in an arms race with your players, who will repeatedly hack the client to give themselves an advantage.

for an on-line shoot 'em up, even a few tens of milliseconds can be too much

If you are downloading a large file, the important factor is throughput. Large files have hundreds of thousands of packets. A several second initial lag when you ask for the file is nothing compared to the tens of minutes it takes between the first packet to arrive and the last one. The more packets you can get through the pipe at the same time (the more lanes), the shorter the download.

Finally, jitter matters if you are receiving streaming video and audio. In video if the packets are arriving at slightly different intervals, and you are watching them in real time, then the picture will appear jerky. In audio the voice or music is distorted by the timing.

Partly this is caused by congestion, and partly it's in the nature of the internet, which you should think of as being like a fishing net where computers and routers are each knot, rather than two tin cans with a piece of string stretched between them. In the internet different packets can take different routes, some of which are longer than others, and as we all know from school physics, even light travels at a finite speed, so longer routes will take a longer time.

There are solutions, of course, and the most common is buffering – you don't start playing the video until you have a chunk of it already downloaded, so you are never in a state where the next packet isn't already there, and each packet can be displayed at the correct time. Unfortunately, you can't delay people's phone calls in this way, which is why Voice over Internet Protocol (VoIP) can still be a little on the ropey side.

So now you have more than you ever wanted to know about latency, throughput, and jitter. Think of it as a little something for you to use to demonstrate your superiority at both dinner parties and wedding receptions – especially if you are the best man!

Confession – this started life as a short piece in my weekly newsletter, 'Winding Down'. I've rewritten and extended it for this rag. Normally, I write fresh stuff for CVu, but this issue I plead a stinking cold striking just as the deadline came up, and exhaustion from having to actually work and commute – yes, surprise, surprise, I finally got a job! ■

Continuous Integration for One

Jez Higgins sees value in team tools for one person.

Like many of my programming chums, I believe in the importance of the build. Breaking the build is a bad thing. Knowingly breaking the build is, well I'm not a religious man, but it's a sin.

Sometimes the build goes wonky or tests start failing because of mishap or misfortune. Someone makes a change to one place in the code, somebody else makes a change in another part, something goes awry and oopsy. Of course, tools can help us here. If you've got something that regularly checks your source code repository for changes, rebuilds everything as soon as it does, and runs your tests, then you can find out if there's a problem sooner rather than later.

These tools have a name, *continuous integration servers*, and are so called because they automate the process of *continuous integration* as described by Martin Fowler [1]. You can put something together yourself pretty easily, but there are any number of them available. At work, we've been using CruiseControl, one of the more widely used CI servers, to reasonable effect over the past few months of pretty intense development.

We want to push our automated build a little bit further. For various reasons, our environment is awkward. Putting a release up into the test servers is complex and still involves a certain amount of time and a degree of manual intervention. Consequently, it isn't happening as often as it should.

We are in a quieter period at the moment and we're managing, but when things hot up again it's going to be a problem. What we would like to do is finish automating the process, and have our CI server build and deploy nightly into our test environment. With that complete, we're going to try and kick off the GUI testing scripts as well. Our testers can spend more time with their feet on the desk drinking tea, while the developers can be secure that yesterday's work does actually work.

It's fallen to me to make this happen. One of the things I needed to do first was spent a bit of time fiddling with the build server. CruiseControl, while a sound piece of kit, doesn't, however, lend itself to easy fiddling. It's driven by configuration files and to pick up changes you need to bounce the server. If you make a mistake in the configuration it's not necessarily obvious why, and that's just a faff and a time consuming one.

By coincidence as I started this work, several people on *accu-general* had been discussing the Hudson build server. One thing that stuck in my mind was that Hudson could be configured entirely though its GUI. I'm slightly ashamed to say that's the reason why I gave it a try, but I'm glad I did. Hudson is extremely easy to fiddle with – setting up builds, chaining different builds together, tool integration, and so on, are all straightforward and can be done via a web browser. I doubt there's anything it can do that can't be done with CruiseControl, but for exploratory work it's streets ahead.

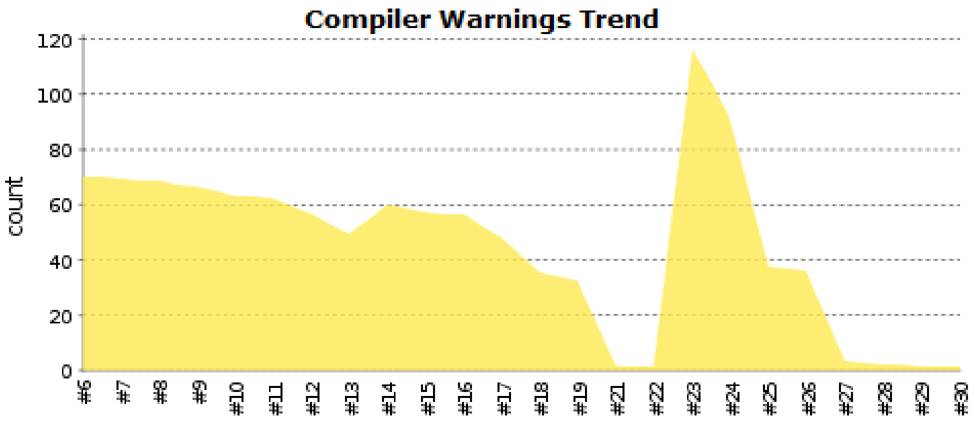


Figure 1

Most of the time I work remotely, accessing work machines over Citrix. It sufficient most of the time, but the Citrix connection isn't exactly snappy and that becomes a little frustrating when you need work across it for an extended period.

Because, at the time, I was more interesting in finding out about Hudson than anything, I installed it on my own server. The network latency between my desktop machine and the server by my left ankle is always going to beat Citrix. So I had Hudson and I needed some code. Fortunately, I had some of that too.. From a standing start to building my Mango library [2] of Java bits and bobs and running its tests, via installing a plugin to pull code from Bazaar repositories, took about five minutes.

When there's only you, there's no one to integrate with

I poked around Hudson a bit more and found a setting to draw a chart of test results. I can see why such a thing is useful in a team situation, but Mango is one man project. I build it and run the tests whenever I work on it. Poking some more, I found another setting that drew a chart of compiler warnings. That's when I had a little revelation.

Continuous integration servers are useful to a group because they'll let you know, and know quickly, when the build has broken. And that's true and it's a real benefit. But it doesn't apply for a one man project. When there's only you, there's no one to integrate with. If a test has failed, it must be you (by which I mean me) who broke it and that must have happened in the last few minutes. What CI servers can do for the one man project is remember. Remember your test results. Keeping track of build times. Remember your compiler warnings. Stuff you wouldn't yourself record unless you were an outrageously meticulous record keeper.

As it happened, Mango had no compiler warnings in the build. However I had another library that did. Written in Java several years ago, I hadn't had reason to work on it for quite some time. Building with a modern Java compiler produced a slew of warnings, which I'd not had the inclination to fix. Hooking up Hudson to build it, within a few minutes I had a chart telling me exactly how many warnings I had – seventy – together with breakdowns of warnings by type and class name, hyperlinked into the source.

As I explored the warnings report, some of them looked very simple to fix. So I fixed them and committed the changes. So Hudson rebuilt the code and sent me an email telling me it had. That prompted me to look again at the Hudson build console, where I saw the warnings graph had gone down a little. Which prompted me to fix another warning. You can probably fill in the rest. See Figure 1.

JEZ HIGGINS

Jez works in his attic, which was recently replastered, living the devil-may-care life of a journeyman programmer. He is currently learning to tumble turn without getting water up his nose. His website is http://www.jezuk.co.uk/

By remembering what had happened in previous builds and then telling me in an easy to understand way, Hudson prompted me to fix a codebase that had languished for years. The second spike on the chart is where, having sorted out everything in the main code, I turned on warnings in the test code.

The psychological effect of that graph was really remarkable. By having a record of the build warnings that didn't simply scroll off the top of the terminal and disappear, I shamed myself ito fixing them, while simultaneously providing a tool to help fix the problems and positive feedback as I did so.

I decided to make deliberate use of it. When building my Arabica XML toolkit [3], I've always used GCC's default warnings. Recently I've been exchanging emails with a chap called Ash, who's using Arabica to provide XML support for the Flusspferd Javascript project. Flusspferd builds at a higher warning level than I do and Ash had sent me a few patches to silence various things he was seeing. Setting up Hudson to build Arabica took, again, only a couple of minutes even though it's a C++ project using Autotools rather than a Java project using Ant. I cranked up the warning level. 600 hundred warnings! Blimey!

I suspect that Hudson's GCC warnings parser isn't quite as refined as the Java warnings parser, so I don't believe that was the true number. Nevertheless it was still a lot and rather more that I was expecting. They were gone in 15 builds (Figure 2). All of them, gone. I didn't have to make 600 individual fixes – a change in an include file can wipe out several warning messages at a stroke – but I think that's pretty quick. And I know I'll keep those warnings at zero too.

In the next few days, I hacked Arabica's test suite to generate JUnit compatible output. Now Hudson tracks those for me too, as shown in Figure 3. Another plugin prompted me to add in Cobertura test coverage analysis to the Mango build. Hudson charts those results too. If you look at Figure 4 carefully, you'll see the lower line starting to trend up.

I installed Hudson on my server so I could fiddle around with it a bit, expecting to throw it in a couple of days. I didn't expect that it would creep into my brain and change the way I worked on my individual projects. ■

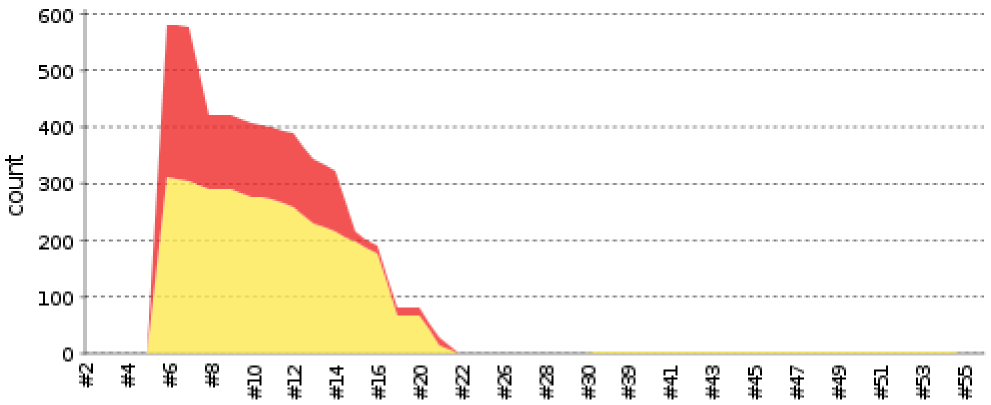


Figure 2

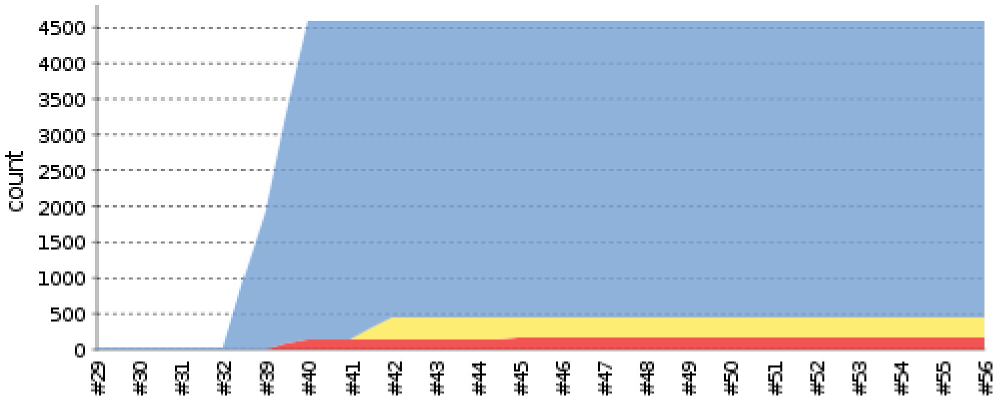


Figure 3

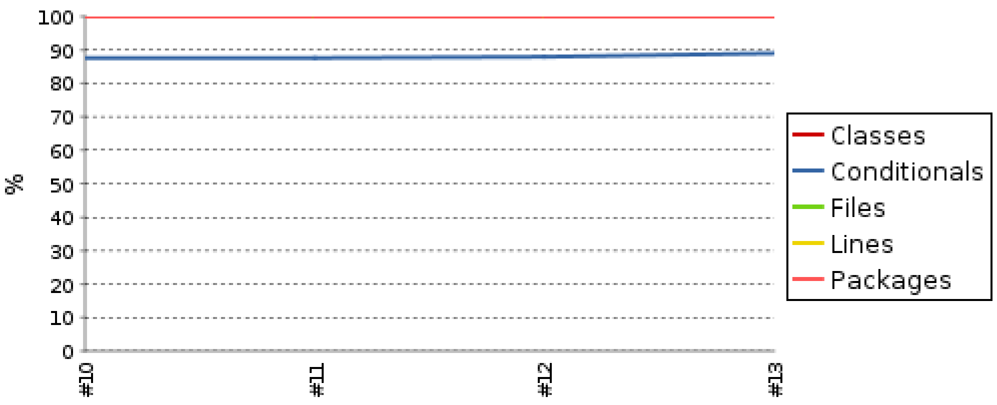


Figure 4

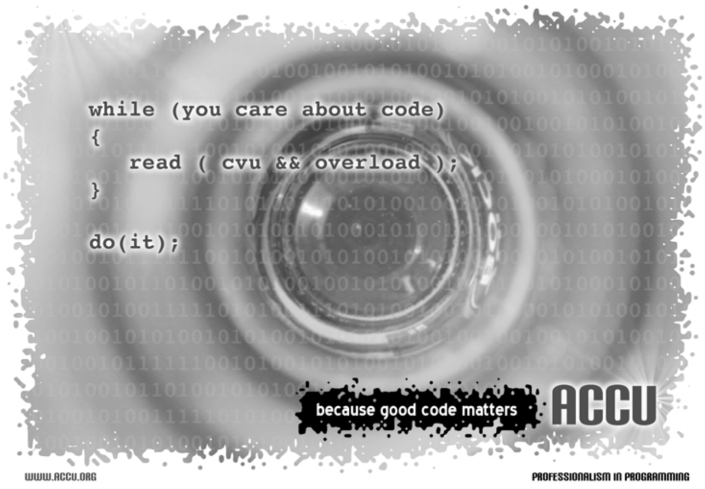
References

[1] Building a Feature with Continuous Integration, http://martinfowler.com/articles/continuousIntegration.html #BuildingAFeatureWithContinuousIntegration

[2] Mango, Java library of iterators, algorithms, and functions, http://www.jezuk.co.uk/mango

[3] Arabica, an XML and HTML toolkit written in C++, http://www.jezuk.co.uk/arabica

[4] Flusspferd, Javascript bindings for C++, http://flusspferd.org/


while (you care about code)
{
read (cvu && overload);
}
do(it);
because good code matters
ACCU
www.accu.org
PROFESSIONALISM IN PROGRAMMING

Code Formatting in C++

Phil Nash explains why format counts.

Code formatting or layout is one of the most religious areas of software development. With so many bloody battles fought and lost most developers learned long ago to avoid the matter altogether. They tend to do this by making consistency the only rule that matters. When in Rome do as the romans do. After all, everyone knows that it's all subjective and doesn't actually matter.

Or does it?

I'm going to present a couple of views that I hope will lead you looking at the matter once again. Some of them a little cliched. Some a little more novel.

First one of the cliches:

> code is written for humans to read – otherwise we'd all be writing assembler

Of course high level languages are not only about human-readability. They are also about portability and economy of expression, among other things. Nonetheless human-readability is certainly a large part of it. So if we use high level languages in order to make our code more readable, should the layout of that code be irrelevant? Looked at from this perspective we may say, 'it *should* be relevant, but code is also for *other* humans and it's the differences in styles that create the problems – it all balances out to zero'. In the context of software development is it worth fighting religious war over a zero-sum game?

What if there was a way to come to a consensus? Would there now be some advantage to looking at layout? If so, how much advantage? These are questions that I hope to answer shortly.

In the meantime, here's another cliche:

> code formatting is just personal preference. It has no intrinsic value

Again – is it worth fighting over something that has no value? But what if it did?

Sometimes it can be useful to look at the extremes. Consider listing 1. It's not too difficult spot the familiar 'first c++ program' example, even if it's in a less familiar layout. But are you sure it's correct? It shouldn't take too long to spot the bug (and even if you don't, the compiler will point it out to you), but now look at the same code expanded to a more canonical form, in listing 2.

I would bet that, for the majority of c/c++ developers spotting the missing `;` in the second example was faster, and perhaps even more 'automatic' than in the first. Even if you agree with that you may wonder, still, how much that matters. We're talking seconds, or perhaps milliseconds, difference in time. And that leads us to another cliche:

> The majority of development time is spent debugging code, rather than writing it

Listing 1

```
int main ( int argc , charargv[]){printf("hello world\n")}
```

This is often wheeled out when encouraging use of longer, more descriptive, identifier names, or the use of a more verbose, but explicit, way of doing something. Actually those areas are somewhat related to our topic, but here the point is: if actual coding time is a small part of overall development time, do a few milliseconds here, a second or two there, make any difference (and remember that was a fairly extreme case)?

These are all good questions. I'm now going to explore some possible answers for them. What I'm going to present here is my own view, based on my own experience and research, as well as the experience of a number of others that have tried my techniques. However the 'number of others' is not statistically relevant enough (nor the conditions controlled) to call it a study, so this remains merely a theory. I encourage you to consider this material and let me know how you get on with it.

How the eyes and brain read

A few years ago I studied (or, more accurately started studying) speed reading. A significant portion of what you learn is understanding how the eye moves across the page, takes information in, and works with the brain to turn this into the experience we know as reading. By understanding these principles we can adjust our reading style to take advantage of their strengths (and play down their weaknesses). As I learnt more I wondered whether the same ideas could be applied to writing as well as reading. It seemed logical that if you write in a way that more closely matches how the eyes and brain read best then reading will be easier, and potentially faster. As I continued studying I found that this is the case. One commonplace example is newspaper and magazine text. This text is arranged in columns as columnar material is easier to digest by the eyes and brain when reading. From here I began to wonder if this knowledge should affect the way we write code.

if actual coding time is a small part of overall development time, do a few milliseconds here, a second or two there, make any difference

Before we look at my conclusions I'll summarise the key speed reading insights I thought would be relevant:

Perhaps most important is that the eyes don't move across the page in a smooth, flowing, manner. Instead they jump in discrete fixations. At each fixation the eyes transmit a chunk of information to the brain. The size and content of each chunk varies and is one of the areas that a speed reader will exercise, attempting to take in more information at each fixation to avoid wasting too much 'seek time'. An average, untrained (in the speed reading sense), reader will take in about 2-4 words per fixation. For longer words this will decrease, and if the words are unfamiliar may drop below the one-word-per-fixation level.

Listing 2

```
int main ( int argc , char* argv [])
{
  printf("hello world\n")
}
```

PHIL NASH

Phil Nash has been coding for 29 years, and getting paid for much of the last 19 of them, working with C++ and related languages, including C#, Java and Objective-C. He runs a blog at http://www.levelofindirection.com and can be followed on Twitter as phil_nash.

Following from this is the insight that as well as several words along the same line being taken in in a single fixation, multiple *lines* may be taken in. But that's crazy talk! When you're reading you don't read the lines above and below, do you? (unless you're regressing, which is a bad habit that speed readers try to overcome as soon as possible). Well, speed readers will push the envelope here, but even for the rest of us we will still be taking in more information than we are consciously reading. The smooth, flowing, word-by-word reading experience that we perceive is an illusion. There is actually a disconnect between the information being captured by the eyes, transmitted to the brain, assembled and interpreted, and the perception of words flowing through our conscious minds.

We're in danger of getting too deep here. Let's stick with the knowledge that we can take several words horizontally *and* vertically in at each fixation. We can add to that another counter-intuitive nugget from the speed reading world. Good speed readers don't necessarily scan left-to-right then down the page (assuming a western reading context), but may scan in different directions according to different strategies – e.g., scanning down the page, then back to the top for the next column – even if they then have to mentally reassemble back into the original word order (I never got to this stage).

None of this addresses the question of whether this is even worth looking at. Are we trying to solve a problem that doesn't exist? We're constantly warned against premature optimisation but can that apply to our approach to code layout too?

Actually we have touched on one relevant principle already. I'll highlight it again here:

> The smooth, flowing, word-by-word reading experience that we perceive is an illusion

Why is this relevant? Well for one it tells us that what we think is happening is not necessarily what is actually happening. A lot of processing is occurring before the material we are reading *is even presented to us consciously*. With practice we can get better at reading unoptimised material – so much so that we are unaware of it. That doesn't mean the processing isn't happening. Processing has a cost associated with it. It tires us in particular it tires parts of the brain that we tend to use for other programming related activities too, such as solving certain types of problems – or extracting relevant details from a sea of information. It's almost like offloading some heavy number-crunching to a GPU then finding that your refresh rates are suffering.

When you're reading you don't read the lines above and below, do you?

Another factor is the concept of *flow*. When we are thinking about one problem and we are focused on it we are in a certain flow. The more we are then distracted by the mechanics of the task – consciously or subconsciously – the more it can knock us out of the the flow. Again, we may be so used to this that we hardly notice. Pay attention next time you are stuck in a problem and you find yourself losing track the more you have to hunt around through the code.

In summary, we need to get out of the trap of just looking at the numbers (a second here, a few milliseconds there). They may bear little relation to the real factors at play.

There's more we can learn from the world of speed reading and eye-brain coordination, but we now have some things to go on that can lead us to conclusions about code formatting.

Code fixation

First, the way eye fixations are able to take in multiple words both horizontally *and* vertically suggests that islands of related code should be readily assimilated in one *or* two fixations. Such islands can be created through logical grouping, and effective use of whitespace both to separate from other islands and for alignment purposes. Alignment touches on another eye–brain insight that I've not mentioned yet. Briefly, reading speeds can be enhanced by the use of a guide. This may be a moving object such as a finger or pencil), but just having a hard edge can be helpful too. However, too much of the same thing can make it harder to keep track of where we are, so if the hard edge is too long it loses some of its effectiveness. It follows, therefore, that small blocks with hard edges achieved through alignment should help the eye to more readily distinguish the associations between sections of code.

Listing 3
```
char* txt="hello" ;
int i = 7;
std::string txt2 = "world";
std::vector<std::string> v;
```

Listing 4
```
char*                       txt   = "hello";
int                         i     = 7;
std::string                 txt2  = "world";
std::vector<std::string>    v;
```

Listing 5
```
char*                       txt = "hello";
int                         i = 7;
std::string                 txt2 = "world";
std::vector<std::string>    v;
```

A lot of these are things we already do to some extent. Our use of code blocks and indentation help visually organise code to make it easier to take in – but can we take it further?

One good example is blocks of variable declarations. I'll be using C++ as an example here, as that has been my focus in this, but most of what I'm talking about applies to most, if not all, programming languages. I'd argue that you'll notice the difference in C++ more than most.

So, here is a typical stretch of variable declarations in listing 3.

Already this is organised into a little island. If the declarations were scattered around we would lose that aspect. Whether that makes sense for your application is another matter. I'm not suggesting you lose the benefit of declaring variables closer to where they are used. What I really want to illustrate is what happens if you add a bit of whitespace for alignment purposes. See listing 4.

Now most of us have probably seen code like this. Maybe we already prefer such a style. But quite a number of developers seem to be against it, either actively (they really dislike it), or at least have concerns over the extra overhead of writing and maintaining in this style.

Well, if you're one of that number, please bear with me. There is more to get out of this yet. Also, as we'll see, I believe the big wins are actually in other areas that we're building up to.

So let's analyse the properties of his format for a moment. Firstly we have three columns here. The first column contains the types. The second the names and the third the initially assigned values, if any. One of the potential problems here is that, as each column is as wide as the longest field in that column, the more columns we have the more horizontal space we'll end up using. In C++, between templates and namespaces, this can get out of control quickly. As we'll come onto a bit more later, if you have to scroll to take in a line you'll undermine any efforts to make things more readable.

Another problem is that it amplifies the objection over the writing and maintenance overhead of such a style. In this simple example we have seven points at which whitespace must be maintained for alignment purposes!

A compromise is to use just two columns, as in listing 5.

This still allows the identifier names to be easily scanned, but at some loss of clarity with respect to the initialised values. Arguably the identifier names are the most important element here (from the perspective of fast lookup), and are the most obfuscated in the original example, so this is still

Listing 6

```
void NamespaceQualifier :: SomeDescriptiveClassName ::LongishMethodName( int firstParameter ,
                                                                         const std : : string& secondPar
                                                                         const std : : vector& thirdPara
                                                                         Widget fourthParameter )
```

a big improvement. How big? We're getting to that. At this point I just wanted to present some options and examples, with a little rationale. We'll build on these shortly.

For sake of arguments

Blocks of variable declarations are common, but spreading them out through a function is perhaps even more common – making the above examples less relevant.

However there are a couple of places where we do still regularly see groups of related variable declarations in once place. One is in class definitions, where member variables are usually grouped in one place. Immediate readability of the names is probably even more important here as we tend to flip back to a class definition in a header file (in the case of C++) just for a moment to get the names.

But the other place, and where I'd like to focus further, is function parameter lists.

Parameter lists are obviously important. They define the interface between the function (or method – I'll use function here to mean both) and its caller. When looking at a function signature (usually at the prototype in a header file) a caller can see what arguments need to be passed in. However, when looking at the function *body* the parameter list shows you what has been passed in. If these two statements sound obvious then why do we so often neglect how these things are presented – as if they are second class citizens in our code?

How often have you seen (or written) a function that takes some number of parameters, where the parameter list is all on one line and spans more than a typical screen-width? – sometimes several screen-widths! Often an attempt is made to rectify the situation by splitting the list over one or more lines, sometimes even one parameter per line (but by no means always). Even then the tendency is to place the first parameter on the same line as the function name, then try to line up the subsequent parameters with the first. Something like listing 6.

Does this look familiar? I don't know about you but, even though it might look like an extreme example, I see this sort of code all the time. And remember this is where someone has made an attempt to split across lines and use alignment! Just today I saw an example on my current project of a function signature that was 239 characters wide – and that was by no means the worst case in the project.

And we haven't even added namespaces yet (except for **`std`**)!

I think you'll agree that this is a big readability issue. The issue is made worse by a lack of consistency which often accompanies such attempts. Sometimes multiple parameters are on one line, sometimes split across several. Sometimes aligned, sometimes not. Even if the developer is following other conventions consistently this one seems to slip through the cracks – probably because it is often not defined and it can be difficult to know exactly what to do in a consistent way.

I have a theory about this state of affairs. It's an important theory (even if it turns out not to hold in this case) because it touches on a principle of why people get so religious about code formatting in the first place. I'm going to expand on that a bit later, but for now the theory is this:

> Developers can't find a consistent style because the most obvious consistent style seems somehow 'wrong'

The underlying principle, which I'll come back to is:

> Regardless of what is objectively good, what is familiar always wins out

So what is the one true way when it comes to formatting function signatures? Well, apart from 'one true way' being an overstatement, I'm going to leave my specific recommendation for a subsequent article (how else could I get you to come back?)

End of scope

Before I finish for now I want to come back to the issue of how important this all is in the first place. We touched on three areas that I believe are worthy of consideration:

- The overhead of reading code 'unoptimised' for reading may be more significant than we realise due to the subconscious processes that are at play – not just in time, but in energy and focus.
- Even relatively small interruptions to our state of flow can have a noticeable impact on our productivity. Sometimes it can even become a serious bottleneck – think 'butterfly effect'.
- Having a consistent style may actually speed up code writing time, but optimising that style for the way our eye-brain connection works can lead to significant increases in code reading/ comprehension time.

In the next articles I will present my recommendation for function signature formatting and return once again to the question of how much difference it makes, including some anecdotal evidence. ■

A Game of Strategy

Baron Muncharris sets us another challenge.

Sir R-----! my fine fellow! I must say that it does me good to see you here this winter's night. Come take a glass of this warming spirit to see off Jack Frost and his mischievous cohorts.

As a Gentleman of sport, might I tempt you to pit your purse against mine in a game of strategy?

Splendid! It is to my shame that I entertained the remotest doubt that a man of your mettle would shy from a challenge.

I have in mind a game that I learnt from Feldmarschall von ?'?a§dr, commander of the dragon army of that greatest and most enlightened nation of Neptune; East Grinstead. It is employed there to train young officers in the art of placing defensive forces in the contested territory between them and West Grinstead; the two nations having been formally at war following the rape of Bramber some several hundred years ago.

The rules of the game are simple enough. I shall place a coin heads up upon a chess board and then you shall counter by placing a coin tails up upon any square other than that I have chosen. Each square on the board will be deemed to be controlled by the player whose coin is closest to it, as measured by the least number of moves required to move from the one to the other by a king in the noble game of chess.

We shall continue to take turns placing coins in unoccupied squares until we have both placed three coins on the board. If, after we have each played our three coins, I control more squares on the board than you then I shall have won the game and may take all six coins as my bounty. To counter the advantage I have in playing first, you shall win and take the coins if you control a greater or equal number of squares as I.

Now I feel it is only fair to warn you that I have something of a natural talent for strategy. Indeed, after having bested every officer in training I was immediately given the rank of Oberleutnant and dispatched to defend the border. This was, of course, during the well reported escalation of hostilities that arose after East Grinstead beat West Grinstead 13 knees to 3 in the Neptune International Cup and the subsequent accusations that the former had been wearing illegal performance enhancing lederhosen.

The account of how I diffused the situation and averted all out war with a well timed witticism regarding the peoples of Saturn is a tale for another evening.

I explained these rules to that witless student whom it is my very great misfortune to know and he started babbling on about the strangulation of some poor fellow called Delaney during a discreet duel with Lord Fauntleroy. Not a minute later he admitted that his story probably had little bearing on the playing of this game; a fact so transparently self evident that it should have been clear to all but the most feeble of minds.

Now, sit here and take another glass whilst I decide on my first move! ■

Figure 1 shows the distances of squares from O and Figure 2 shows coins and controlled squares after each round of a game (heads: O, tails: X)..

Figure 1

4	3	3	3	3	3	3	3
4	3	2	2	2	2	2	3
4	3	2	1	1	1	2	3
4	3	2	1	O	1	2	3
4	3	2	1	1	1	2	3
4	3	2	2	2	2	2	3
4	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4

Figure 2

				O			
		X					

O	O	O	O	O	O	O	O
	O	O	O	O	O	O	O
X		O	O	O	O	O	O
X	X		O	O	O	O	O
X	X	X		O	O	O	O
X	X	X	X		O	O	O
X	X	X	X	X		O	O
X	X	X	X	X	X		O

				O			
						X	
		X					
					O		

O	O	O	O	O	O	O	
	O	O	O	O	O	O	
X		O	O	O	O		X
X	X		O	O		X	X
X	X	X		O		X	X
X	X	X	X	O			X
X	X	X	X	O	O	O	
X	X	X		O	O	O	O

		X					
O							
				O		X	
		X					
					O		

	X	X	X	X			
O		X	X				X
O		X		O		X	X
O	O		O	O		X	X
	X	X		O		X	X
X	X	X	X	O	O	O	
X	X	X	X	O	O	O	O
X	X	X		O	O	O	O

Listing 1 is a struct representing a position; listing 2 is measuring the distance between two positions; listing 3 is calculating the scores.

Listing 1

```
struct position
{
  size_t rank;
  size_t file;
  position();
  position(size_t rank, size_t file);
  bool operator==(const position &x) const;
};
position::position()
{
}
position::position(size_t rank,
   size_t file) : rank(rank), file(file)
{
}
bool
position::operator==(const position &x) const
{
  return rank==x.rank && file==x.file;
}
```

On A Game of Skill

The Baron's student acquaintance plays the game.

Recall that the Baron's latest game involved taking turns to select cards from the set

A♥ 2♥ 3♥ A♠ 2♠ 3♠ A♣ 2♣ 3♣

and trying to construct tricks of three cards with either the same suit or face value. The Baron's opponent is the first to select a card and has the advantage that three cards all differing in both suit and face value are a winning trick provided they include the first card selected.

The Baron balances this advantage with the fact that he is also considered the winner if neither he nor his opponent can build a winning trick before the cards are exhausted.

Upon hearing the rules, I immediately exclaimed that the game is isomorphic to the children's game of tic-tac-toe, as can readily be seen by properly laying out the cards upon the table.

Specifically, the first card chosen by the Baron's opponent should be surrounded by the remaining cards such that each row is formed from one suit and each column from one value. For example, if the first card is 3♠ we could lay out the cards as follows.

A♣	3♣	2♣
A♠	3♠	2♠
A♥	3♥	2♥

Here, the Baron's winning tricks are the rows and columns and his opponent's the rows, columns and diagonals.

Now, when the Baron selects a card we shall place an X in the gap and for his opponent an O. Doing so, an example of play might be

A♣	3♣	2♣
A♠	O	2♠
A♥	3♥	2♥

A♣	3♣	X
A♠	O	2♠
A♥	3♥	2♥

A♣	3♣	X
A♠	O	O
A♥	3♥	2♥

A♣	3♣	X
X	O	O
A♥	3♥	2♥

A♣	O	X
X	O	O
A♥	3♥	2♥

A♣	O	X
X	O	O
A♥	X	2♥

A♣	O	X
X	O	O
A♥	X	O

X	O	X
X	O	O
A♥	X	O

X	O	X
X	O	O
O	X	O

We can thusly construct a game of tic-tac-toe for any first choice of card and, as any child might tell you, this game cannot be won by either player barring error.

As such I should have advised Sir R----- to have declined to play. ■

A Game of Strategy (continued)

Listing 2

```
size_t
abs_diff(size_t x, size_t y)
{
  return x>y ? x-y : y-x;
}
size_t
distance(const position &x, const position &y)
{
  const size_t rank_dist
     = abs_diff(x.rank, y.rank);
  const size_t file_dist
     = abs_diff(x.file, y.file);
  return std::max(rank_dist, file_dist);
}
```

Listing 3

```
size_t
shortest(const position &x,
   const std::vector<position> &moves)
{
  size_t best = 8;
  std::vector<position>::const_iterator first
     = moves.begin();
  std::vector<position>::const_iterator last
     = moves.end();
```

Listing 3 (cont'd)

```
  while(first!=last)
  {
    const size_t dist = distance(x, *first++);
    if(dist<best)  best = dist;
  }
  return best;
}
std::pair<size_t, size_t>
score(const std::vector<position> &first,
      const std::vector<position> &second)
{
  std::pair<size_t, size_t> score(0, 0);
  for(size_t rank=0;rank!=8;++rank)
  {
    for(size_t file=0;file!=8;++file)
    {
      const position x(rank, file);
      const size_t first_best
         = shortest(x, first);
      const size_t second_best
         = shortest(x, second);
      if(first_best < second_best)
         ++score.first;
      if(second_best < first_best)
         ++score.second;
    }
  }
  return score;
}
```

Assert Yourself

Richard Harris finds new respect for the humble assert.

When I started out as a, for want of a better description, professional programmer, I was advised by a senior colleague to avoid asserts and rely instead upon proper error handling. The reasoning was sound enough; since asserts compile to nothing during a release build any errors that they check for will pass by unnoticed.

Relatively recently, I came to the conclusion that this argument misses a crucial point; asserts are not for checking statements that *should* be true, but rather for checking statements that *must* be true.

The distinction is principally that the former are not within the control of your own code and the latter are. Any requirements that can be broken by a client using the public interface to your code, be that functions declared in a header file or public member functions of a class, or by interaction with external libraries and/or resources fall into the former and are rightly handled with a fully featured error mechanism such as exceptions. Any requirements that are logical consequences of the correct implementation of your code are rightly handled with asserts.

we as humans seem spectacularly ill equipped to reason about geometry in more than two dimensions

The most obvious example is the use of asserts to describe pre and post conditions of any non-public functions used to abstract away implementation details of functions in the public interface to your code. As such, they serve as a kind of compiler enforced documentation to assist future maintenance; not as powerful as that provided by languages like Eiffel, but useful nevertheless.

Now I am reasonably certain that I haven't raised any points that aren't already well known. However, I have recently been exposed to another use of assertions that I suspect is not so widely disseminated.

I am currently trying to generalise an algorithm for constructing Delaunay triangulations from the plane to an arbitrary numbers of dimensions. A Delaunay triangulation is a unique scheme for connecting points in a plane into triangles such that every vertex is one of the points. All of my sources claim that such a generalisation is possible, although irritatingly none of them provide any specific details on how one might go about it.

Neither a description of the algorithm nor my desire to generalise it are pertinent to this article, so I shall provide neither. I can almost hear the sighs of relief from readers sick to their stomachs of my relentless crusade to force feed mathematics to members of the ACCU.

I shall, however, attempt to impress upon you the tremendous difficulties that arise when trying to generalise an algorithm that operates in two dimensions to one that operates in an arbitrary number. The difficult step is, in fact, moving from two to three dimensions. Many geometrical properties that are blisteringly obvious in two dimensions turn out to be patently false in three.

A nice example is the fact that there are an infinite number of regular polygons; polygons with sides of equal length and internal angles of equal size. In contrast there are only five regular polyhedra; the Platonic solids. The simplest way to demonstrate that this is so is to transform the problem from one of geometry to one of algebra and to reason about numbers rather than shapes. In doing so, one is equipped to further show that there are six in four dimensions and just three in all other numbers of dimensions.

The fact that we as humans seem spectacularly ill equipped to reason about geometry in more than two dimensions is something I believe to be a consequence of the fact that we live in a two dimensional world. Our retinas are two dimensional; we see in two dimensions plus depth. Our movement is two dimensional; until very recently in our history we were more or less constrained to move only upon the surface of the Earth. Small wonder that we seem to have an intuitive grasp of two dimensional geometry, but are often flummoxed in three or more dimensions; try to imagine a rotating a four dimensional analogue of a triangle for a clear demonstration of our cognitive limitations.

The transformation of a problem from one of geometry to one of algebra is almost always the easiest way to generalise it to higher number of dimension and it is exactly this approach that I have taken in trying to construct my arbitrary dimensional Delaunay triangulations.

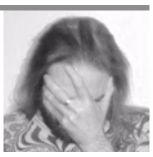
Specifically, I represent my multidimensional triangles with arrays of indices into a collection of points. I must now try and figure out the rules for manipulating those arrays of indices such that their geometrical interpretation represents a Delaunay triangulation. Unfortunately this is not an easy task since I am not capable of intuiting the geometric operations required to manipulate my triangles from which I might derive a system of rules for manipulating my arrays of indices. Indeed, I have upon my desk a number of models built out of toothpicks and ticky tack!

And it is here that asserts have come to my rescue. By using asserts to test things that I *believe* to be true I have been able to discover when I have held erroneous beliefs due to my pedestrian two dimensionalism. They have proven to be a supremely effective tool for whittling away at my assumptions and revealing the true rules for the geometric manipulation of multidimensional triangles.

I am convinced that this can be exploited in the development of *any* algorithm. Unit tests can tell us that our code fails to meet requirements, but not *why*. With appropriate use of asserts we can reveal the *why* and that is the reason behind my new-found respect for this humble and unfairly overlooked device. ■

RICHARD HARRIS

Richard Harris has been a professional programmer since 1996. He has a background in Artificial Intelligence and numerical computing and is currently employed writing software for financial regulation.



Developing Web Applications with the Google Web Toolkit

Paul Grenyer explores Web UI development in Java.

The Model View Controller pattern [1] is often used to build Java based enterprise web applications. It separates the user interface (view) from the application logic (controller) and data (domain). Traditionally the view is implemented using Java Server Pages (JSP) and there are a number of frameworks, such as Struts [2] that can help you do this. However, fundamentally you still have to write JSPs and they are cumbersome and difficult to test.

Writing a user interface for a Java application using JSP does not make much sense when you consider that the rest of the application is written in Java. If there is any functionality shared by the user interface and the application logic it must be implemented and maintained twice. If you have objects in the domain that need to have all or part of their state displayed, you have to decompose them and transfer the data to a page to be displayed.

The Google Web Toolkit [3] solves all of these problems and is much easier to write and test than JSPs. In this article I am going to look at GWT application project structure and how to write an application for creating and modifying Spring Security users as an example of how to develop applications with GWT.

Creating a GWT application

Creating an application with the GWT plugin could not be much simpler. Click the New Web Application Project button from the GWT plugin toolbar and fill in the New Web Application Project dialog (Figure 1):

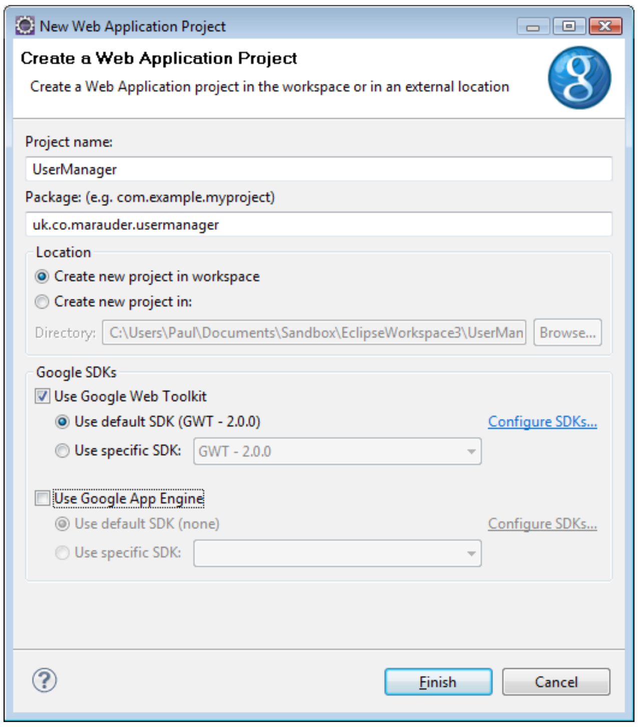
1. Fill in the project name (e.g: **`UserManager`**).
2. Fill in the package name (e.g: **`uk.co.marauder.usermanager`**). This is the top level package under which the Java parts of the project will be.
3. Untick the Use Google App Engine as it is not required for a GWT project.
4. Click Finish to create the project.

This will create a project called UserManager. If you are keen to see what a GWT web application looks like it can be run in the normal Eclipse way by selecting the project in Eclipse's Project Explorer and clicking the green run arrow. This will open a development mode window containing a URL similar to the following:

```
http://localhost:8888/UserManager.html?gwt.
codesvr=192.168.106.1:9997
```

The web application is running in an embedded webserver and can be accessed by pasting the URL into a browser. The first time you do this you will need to follow the instructions to install the GWT Developer plugin.

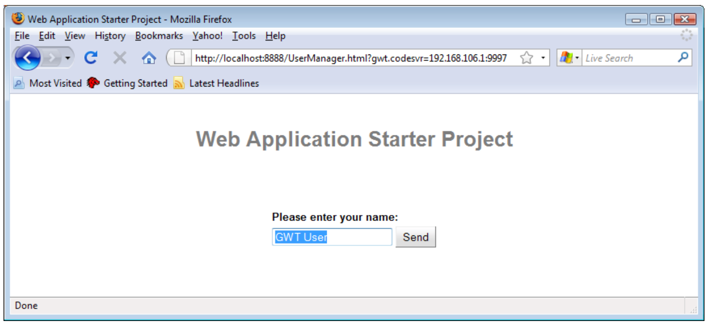
Once the plugin is installed and your browser restarted the Web Application Starter Project, which is the default GWT project, is displayed (see Figure 2). It consists of a title, and edit box with a label and a button. Clicking the button sends the contents of the edit box to the server, via Remote Procedure Call (RPC). The server returns some information which is displayed in a dialog box (see Figure 3). Although this seems very



Create a Web Application Project
Create a Web Application project in the workspace or in an external location

Project name: UserManager
Package: (e.g. com.example.myproject) uk.co.marauder.usermanager
Location
(● Create new project in workspace)
Create new project in:
Directory: C:\Users\Paul\Documents\Sandbox\EclipseWorkspace3\UserMan
Google SDKs
[✓] Use Google Web Toolkit
(● Use default SDK (GWT - 2.0.0))
Use specific SDK: GWT - 2.0.0
[ ] Use Google App Engine
Use default SDK (none)
Use specific SDK:

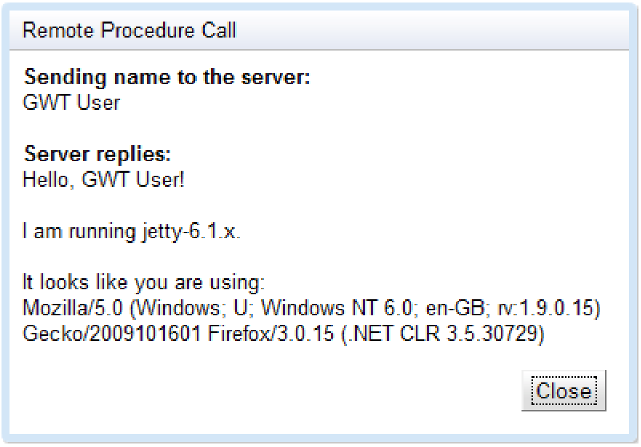
Figure 1


Web Application Starter Project - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
http://localhost:8888/UserManager.html?gwt.codesvr=192.168.106.1:9997
Live Search
Most Visited
Getting Started
Latest Headlines
Send
Done

Web Application Starter Project

Please enter your name:
GWT User

Figure 2



Remote Procedure Call

Sending name to the server:
GWT User

Server replies:
Hello, GWT User!

I am running jetty-6.1.x.

It looks like you are using:
Mozilla/5.0 (Windows; U; Windows NT 6.0; en-GB; rv:1.9.0.15) Gecko/2009101601 Firefox/3.0.15 (.NET CLR 3.5.30729)

Figure 3

PAUL GRENYER

An active ACCU member since 2000, Paul is the founder of the Mentored Developers. Having worked in industries as diverse as direct mail, mobile phones and finance, Paul now works for a small company in Norwich writing Java. He can be contacted at paul.grenyer@gmail.com

Listing 1

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='usermanager'>
  <inherits name='com.google.gwt.user.User'/>
  <inherits name='com.google.gwt.user.theme.standard.Standard'/>
  <entry-point class='uk.co.marauder.usermanager.client.UserManager'/>
  <source path='client'/>
</module>
```

simple, the application is demonstrating two of the most important features of GWT:

- The ability to write a web application front end in Java
- Remote procedure calls

Let's take a look at the project structure. As you can see from Figure 4, GWT projects have a fairly normal Eclipse project structure with a few extras. The extras are the GWT SDK, which consists of the two JARs needed to write and compile Java code and the war directory which provides a deployable structure for the web application. When the application is ready for deployment this directory can be zipped into a `.war` file and deployed to a servlet container such as Apache Tomcat [4].

GWT project structure is explained in detail in the Organising Projects development guide [5] on the GWT website. I have included some extracts below to give an overview:

Module XML files

Individual units of GWT configuration are called modules. A module bundles together all the configuration settings that your GWT project needs:

- Inherited modules
- An entry point application class name; these are optional, although any module referred to in HTML must have at least one entry-point class specified
- Source path entries
- Public path entries
- Deferred binding rules, including property providers and class generators

Modules are defined in XML and placed into your project's package hierarchy. Modules may appear in any package in your classpath, although it is strongly recommended that they appear in the root package of a standard project layout.

The XML Module file generated for the UserManager project, `UserManager.gwt.xml`, looks like Listing 1 and:

- Defines the name of the module for the project (**`usermanager`**)
- Inherits the required GWT modules:
  - **`com.google.gwt.user.User`**
  - **`com.google.gwt.user.theme.standard.Standard`**
- Specifies the entry point class:
  - **`uk.co.marauder.usermanager.client.UserManger`**
- Tells the GWT compiler where to find the source files it needs relative to the module XML file (**`client`**).

HTML host pages

GWT modules are stored on a web server as a set of JavaScript and related files. In order to run the module, it must be loaded from a web page of some sort. Any HTML page can include a GWT application via a **`SCRIPT`** tag. This HTML page is referred to as a host page from the GWT application's point of view. A typical HTML host page for an application written with GWT from scratch might not include any visible HTML body content at all.

The UserManager host page, `UserManager.html`, looks like Listing 2.

GWT was designed to make it easy to add GWT functionality to existing web applications with only minor changes. It is possible to allow the GWT module to selectively insert widgets into specific places in an HTML page. To accomplish this in the example above, the id attribute is used to specify a unique identifier that your GWT code will use to attach widgets to that HTML element.

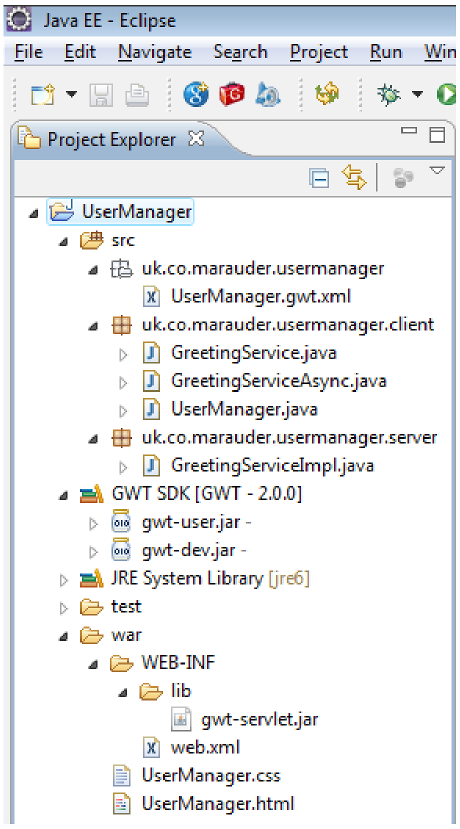


Figure 4

```
Java EE - Eclipse
File  Edit  Navigate  Search  Project  Run  Win
Project Explorer
UserManager
  src
    uk.co.marauder.usermanager
      UserManager.gwt.xml
    uk.co.marauder.usermanager.client
      GreetingService.java
      GreetingServiceAsync.java
      UserManager.java
    uk.co.marauder.usermanager.server
      GreetingServiceImpl.java
  GWT SDK [GWT - 2.0.0]
    gwt-user.jar -
    gwt-dev.jar -
  JRE System Library [jre6]
  test
  war
    WEB-INF
      lib
        gwt-servlet.jar
      web.xml
    UserManager.css
    UserManager.html
```

The UserManager host page:

- Specifies a style sheet called (**`UserManager.css`**), which is also generated as part of the project.
- Sets the page title
- Includes the JavaScript generated for the module by the GWT compiler (**`usermanager.nocache.js`**).

Listing 2

```
<html>
<head>
  <meta http-equiv="content-type"
        content="text/html; charset=UTF-8">
  <link type="text/css" rel="stylesheet"
        href="UserManager.css">

  <title>Web Application Starter Project</title>

  <script type="text/javascript"
        language="javascript"
        src="usermanager/usermanager.nocache.js">
  </script>
</head>

<body>
  <h1>Web Application Starter Project</h1>
  <table align="center">
    <tr>
      <td colspan="2" style="font-weight:bold;">
        Please enter your name:
      </td>
    </tr>
    <tr>
      <td id="nameFieldContainer"></td>
      <td id="sendButtonContainer"></td>
    </tr>
  </table>
</body>
</html>
```

Listing 3

```
public class UserManager implements EntryPoint
{
  ...
  public void onModuleLoad()
  {
    final Button sendButton = new Button("Send");
    final TextBox nameField = new TextBox();
    nameField.setText("GWT User");
    sendButton.addStyleName("sendButton");
    RootPanel.get(
       "nameFieldContainer").add(nameField);
    RootPanel.get(
       "sendButtonContainer").add(sendButton);
    ...
  }
}
```

- Specifies the body of the HTML tag, including two place holders for GWT functionality (**`nameFieldContainer`** and **`sendButtonContainer`**). The **`td`** tags include an **`id`** attribute associated with them. This attribute is accessible through the DOM class. You can easily attach widgets using the method **`RootPanel.get()`**.

Entry point classes

A module entry-point is any class that is assignable to **`EntryPoint`** and that can be constructed without parameters. When a module is loaded, every entry point class is instantiated and its **`EntryPoint.onModuleLoad()`** method is called.

Part of the UserManager entry point class, **`UserManager.java`**, is shown in Listing 3.

Here a button and a text box widget are created and attached to the HTML host page via **`RootPanel.get()`**. There is plenty more code in **`UserManager.java`** which I have not shown. It, along with the code in **`GreetingService.java`**, **`GreetingServiceAsync.java`** and **`GreetingServiceImpl.java`**, is mostly related to making a Remote Procedure Call (RPC) and displaying a dialog box, which I will describe in more detail later.

The WAR directory

The `war` directory is the deployment image of your web application. It is in the standard expanded `war` format recognised by a variety of Java web servers, including Tomcat, Jetty, and other J2EE servlet containers (Table 1). It contains a variety of resources:

- Static content you provide, such as the host HTML page
- GWT compiled output
- Java class files and jar files for server-side code
- A `web.xml` file that configures your web app and any servlets

Listing 4

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems,
     Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  ...
  <welcome-file-list>
    <welcome-file>UserManager.html</welcome-file>
  </welcome-file-list>
</web-app>
```

The `web.xml` file is worth looking at in a little more detail (Listing 4).

In its most basic form, `web.xml` specifies the default or welcome HTML page. For the UserManager project it also tells the servlet container how to handle RPC calls, but as I am going to look at that in more detail later I have omitted it here.

Writing web applications with GWT

To demonstrate how to write a web application in GWT I am going to work through an example and develop a front end for creating and editing Spring Security users, using a slightly modified version of the data access layer I developed as part of my Data Access Layer Design for Java Enterprise Applications [6] article. In the article I created a class to model a Spring Security user and wrote a data access layer capable of serialising it to and from a database. Listing 5 shows a slightly modified version of the class.

The modifications required are making all the setters public and the whole class serializable using the GWT **`IsSerializable`** marker interface. This is because, as we will see shortly, we need to be able to use Dozer, a Java Bean to Java Bean mapper that recursively copies data from one object to another, to make a copy of the class and have it serialised between the client and server.

The **`User`** class has fields for a user name, password, enabled flag and a list of authorities. Therefore the user interface for creating and editing Spring Security users needs to have an equivalent widget for each, plus a button for saving users, a button for finding users, buttons for adding and removing authorities and an edit box for entering authorities to add.

Making a basic project

To start off with we need a basic GWT application and the UserManager project is ideal. If you have not done so already, create it and remove the following files as these are an example of how to implement remote procedure calls and are not needed:

- `GreetingService.java`
- `GreetingServiceAsync.java`
- `GreetingServiceImpl.java`

Table 1

Directory	File	Purpose
war/	UserManager.html	A host HTML page that loads the UserManager application.
war/	UserManager.css	A static style sheet that styles the UserManager application.
war/usermanager/		The UserManager module directory where the GWT compiler writes output and files on the public path are copied.
war/usermanager/	usermanager.nocache.js	This is the script that must be loaded from the host HTMLto load the GWT module into the page.
war/WEB-INF		All non-public resources live here, see the servlet specification for more detail.
war/WEB-INF	web.xml	Configures the web application and any servlets.
war/WEB-INF/classes		Java compiled class files live here to implement server side functionality.
war/WEB-INF/lib		Any library dependencies the server code needs go here.
war/WEB-INF/lib	gwt-servlet.jar	This GWT JAR is needed for RPC support.

Listing 5

```
public class User implements IsSerializable
{
  private String username;
  private String password;
  private boolean enabled;
  private List<String> auths
     = new ArrayList<String>();

  @SuppressWarnings("unused")

  private User()
  {}

  public User(String username, String password,
     boolean enabled)
  {
    this.username = username;
    this.password = password;
    this.enabled = enabled;
  }

  public void addAuth(String auth)
  {
    auths.add(auth);
  }

  public String getUsername()
  {
    return username;
  }

  public void setUsername(String username)
  {
    this.username = username;
  }

  public String getPassword()
  {
    return password;
  }

  public void setPassword(String password)
  {
    this.password = password;
  }

  public boolean isEnabled()
  {
    return enabled;
  }

  public void setEnabled(boolean enabled)
  {
    this.enabled = enabled;
  }

  public List<String> getAuths()
  {
    return auths;
  }

  public void setAuths(List<String> auths)
  {
    this.auths = auths;
  }
}
```

Then remove the **`<servlet>`** and **`<servlet-mapping>`** elements from `web.xml` so that it looks like Listing 6.

Listing 6

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web
       Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- Default page to serve -->
  <welcome-file-list>
     <welcome-file>UserManager.html
     </welcome-file>
  </welcome-file-list>
</web-app>
```

Listing 7

```
<html>
  <head>
    <meta http-equiv="content-type"
       content="text/html; charset=UTF-8">

    <link type="text/css" rel="stylesheet"
       href="UserManager.css">

    <title>User Manager</title>

    <script type="text/javascript"
       language="javascript"
       src="usermanager/usermanager.nocache.js">
    </script>
  </head>

  <body>
    <h1>User Manager</h1>
    <table align="center">
      <tr>
        <td id="gridContainer"></td>
      </tr>
    </table>
  </body>
</html>
```

Again, the **`<servlet>`** and **`<servlet-mapping>`** elements are used by the example remote procedure call implementation and are therefore not needed. `UserManager.html` has a lot of HTML from the web application start project that we do not need, so modify it so that it looks like Listing 7.

This leaves us with a basic host page that displays an appropriate title and a place holder (**`gridContainer`**) for our application that will be horizontally centred in the browser. Finally remove all the web application starter project code from the **`UserManager`** class, so that we just have the basic class:

```
public class UserManager implements EntryPoint
{
  public void onModuleLoad()
  {
  }
}
```

Creating the interface

You now have a very basic GWT application with a place holder ready for the grid which will hold the widgets. Run the application to make sure all is well. Next, add the widgets, as shown in Listing 8.

GWT widgets are a lot like Swing [7] widgets. As you can see here there is a text box for the user name, a password text box for the password and a text box for entering new rules. The password text box is the same as a

Listing 8

```
public class UserManager implements EntryPoint
{
  private final TextBox usernameTextBox
     = new TextBox();
  private final Button findBtn
     = new Button("Find");
  private final Button saveBtn
     = new Button("Save");
  private final TextBox passwordTextBox
     = new PasswordTextBox();
  private final CheckBox enabledChkBox
     = new CheckBox("Enabled");
  private final ListBox authListBox
     = new ListBox();
  private final TextBox authTextBox
     = new TextBox();
  private final Button addAuthBtn
     = new Button("Add");
  private final Button removeAuthBtn
     = new Button("Remove");
  ...
}
```

regular text box, except it masks the characters that are entered. There is check box for marking a user as enabled or disabled, a list box to hold the authorities and buttons for finding and saving users and adding and removing authorities.

Creating the widgets is not enough. They need to be arranged and displayed on a panel. One way to do this is to use a **Grid**. To keep the application readable and expressive, create a **buildGrid** method:

```
private Grid buildGrid()
{
  final Grid grid = new Grid(5,5);
  // Add widgets to grid
  return grid;
}
```

The **Grid** constructor takes the number of rows and columns (in that order) that are required. The **setWidget** method is used to add widgets to the **Grid**. It takes a row and column and the widget to add. The widgets can be arranged any way you want, but I've decided to put the user name text box, the Find and the Save buttons on the first row:

```
grid.setWidget(0,0, new Label("Username:"));
grid.setWidget(0,1, usernameTextBox);
grid.setWidget(0,2, findBtn);
grid.setWidget(0,3, saveBtn);
```

You will notice that there is another type of widget here. The **Label** widget allows you to put text into the grid as a widget, so it's easier to customise with CSS. You could use the **setText** method instead if you wanted to. The first row is followed by the password text box on the next row:

```
grid.setWidget(1,0, new Label("Password:"));
grid.setWidget(1,1, passwordTextBox);
```

then the enabled check box on the row after that:

```
grid.setWidget(2,1, enabledChkBox);
```

and the authorities widgets on the final two rows:

```
grid.setWidget(3,0, new Label("Authorities:"));
grid.setWidget(3,1, authListBox);
grid.setWidget(3,2,removeAuthBtn);
grid.setWidget(4,1,authTextBox);
grid.setWidget(4,2,addAuthBtn);
```

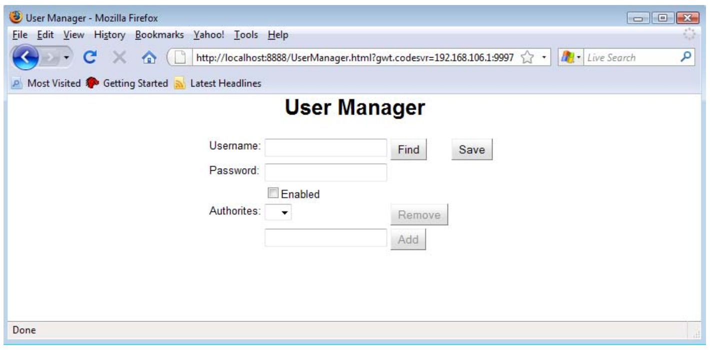

User Manager - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
http://localhost:8888/UserManager.html?gwt.codesvr=192.168.106.1:9997
Live Search
Most Visited
Getting Started
Latest Headlines
User Manager
Username:
Find
Save
Password:
Enabled
Authorites:
Remove
Add
Done

Figure 5

Place all of this code in the **buildGrid** method directly after the **Grid** instantiation. Then to display the grid it needs to be added to the host page:

```
public void onModuleLoad()
{
  RootPanel.get("gridContainer").add(buildGrid());
}
```

If you run the application now you will see, the not especially sexy, interface (Figure 5). I will take a look at how to sex-it-up once the rest of the functionality is in place.

As it stands you can type in the text boxes, click the check box and press the buttons, but they do not do anything. Next, let's look at the authorities widgets. The authorities list box currently looks like a combo box. To make it behave as a list box it needs to be told to display more than one item. The add button should only be enabled when there is text in the authorities text box and the remove button should only be enabled when an authority is selected in the authorities list box. Create another method called **setupAuthWidgets** and start by setting the number of visible items for the list box and disabling both the buttons:

```
private void setupAuthWidgets()
{
  authListBox.setVisibleItemCount(5);
  removeAuthBtn.setEnabled(false);
  addAuthBtn.setEnabled(false);
}
```

Call it after adding the grid to the panel:

```
public void onModuleLoad()
{
  RootPanel.get(
     "gridContainer").add(buildGrid());
  setupAuthWidgets();
}
```

To detect whether there is text or not in the authorities text box, we need to add a change handler:

```
authTextBox.addChangeHandler(new ChangeHandler()
{
  @Override
  public void onChange(ChangeEvent event)
  {
    final TextBox tb
       = (TextBox) event.getSource();
    addAuthBtn.setEnabled(
       tb.getText().length() > 0);
  }
});
```

Put this change handler in the **setupAuthWidgets** method. When the contents of the authorities text box changes the change handler's **onChange** method is called. Its **ChangeEvent** parameter includes a reference to the widget that has changed, which is accessible via the **getSource** method, which returns object so must be cast to the appropriate type. Getting the text from the text box and checking its length is then used to enable and disable the authorities add button via its **setEnabled** method.

We also need a change handler to enable and disable the authorities remove button when an item in the authorities list box is selected or no items are selected:

```
authListBox.addChangeHandler(new ChangeHandler()
{
  @Override
  public void onChange(ChangeEvent event)
  {
    final ListBox lb
       = (ListBox) event.getSource();
    removeAuthBtn.setEnabled(
       lb.getSelectedIndex() != -1);
  }
});
```

Put this change handler in the **setupAuthWidgets** method. It works in the same way as the change handler for the text box, except that the list box's **getSelectedIndex** method is used to determine if an item is selected. If no item is selected it returns **-1**.

To add authorities into the authorities list box you need to add a click handler to the authorities Add button:

```
addAuthBtn.addClickHandler(new ClickHandler()
{
  @Override
  public void onClick(ClickEvent event)
  {
    authListBox.insertItem(
       authTextBox.getText(), -1);
  }
});
```

When the button is clicked the **onClick** method is called. The contents of the authorities text box is read and inserted into the authorities list box at position **-1**, which means new authorities are always inserted at the top of the list. A click handler is also needed to remove items from the authorities list box:

```
removeAuthBtn.addClickHandler(new ClickHandler()
{
  @Override
  public void onClick(ClickEvent event)
  {
    authListBox.removeItem(
       authListBox.getSelectedIndex());
  }
});
```

In this case when the Remove button is clicked and the **onClick** method called, the index of the currently selected item is retrieved using the **getSelectedIndex** method and removed from the authorities list box using the **removeItem** method.

If you run the application now you will see that when you type an authority into the authorities text box, the Add button enables and clicking it inserts the authority into the authorities list box. Selecting the authority in the list box enables the Remove button and clicking it removes the authority from the list box. However, the Remove button remains enabled even when there are no selected authorities in the list box. This is due to the change handlers not getting called when items are inserted or removed from a list box. This is by design as GWT assumes that you know when you are inserting or removing items from a list box as you are explicitly calling **insertItem** or **removeItem**. This is not the behaviour that we want though!

To get the behaviour we want we need to inherit from **com.google.gwt.user.client.ui.ListBox** and override **insertItem** and **removeItem** and modify the implementations so that they cause the change handlers to be called. To do this, create a package called **uk.co.marauder.usermanager.client.widgets**. It is a sub package of **uk.co.marauder.usermanager.client**, so will be compiled by the GWT compiler. In the new package create a class called **ListBox** (Listing 8).

Listing 8

```
public class ListBox
   extends com.google.gwt.user.client.ui.ListBox
{
  @Override
  public void insertItem(String item,int index)
  {
    super.insertItem(item, index);
    ChangeEvent.fireNativeEvent(
       Document.get().createChangeEvent(),
       this);
  }
  @Override
  public void removeItem(int index)
  {
    super.removeItem(index);
    ChangeEvent.fireNativeEvent(
        Document.get().createChangeEvent(),
        this);
  }
}
```

One of the great things about Java is package scoping, which allows us to create a class called **ListBox** which inherits from another class also called **ListBox** in another package. As you can see, in the overridden **insertItem** and **removeItem** methods the first thing that happens is the original super class method is called. This is so that the insertion and removal behaviour is unchanged. Then the **fireNativeEvent** static method of the **ChangeEvent** class is used to fire a change event, which in turn causes the change handlers to be invoked. To use the modified list box, simply replace the import of **com.google.gwt.user.client.ui.ListBox** in `UserManager.java` with **uk.co.marauder.usermanager.client.widgets.ListBox**. If you run the application now it will behave as expected.

From a user interface point of view, all that is left to do is implement the Find and Save buttons. For these, create another setup method, called something like **setupButtons** and two more click handlers (Listing 9).

Call **setupButtons** right after **setupAuthWidgets**. The Find button click handler simply gets the user name from the user name text box and passes it to the **findUser** method, which we have not written yet. The Save button click handler creates a new User object and initialises it with the user name, password and enabled flag from the user name text box, the password text box and the enabled check box. It then iterates through all the authorities in the authorities list box and adds each one to the user. Finally it is passes the **User** object to the **saveUser** method, which we have not written yet.

Remote procedure calls

GWT only implements a restricted subset of the Java API, so any action that uses something outside of that subset, such as writing to a database, must be carried out on the server via an RPC call. Making an RPC call is the equivalent to making a method call to the server and is actually quite easy to do with GWT.

First we need a service interface which defines the methods we want to be able to call on the server. The interface must extend GWT's **RemoteService** interface:

```
@RemoteServiceRelativePath("user")
public interface UserService
   extends RemoteService
{
  User find(String username);
  void save(User user);
}
```

Listing 9

```
private void setupButtons()
{
  findBtn.addClickHandler(new ClickHandler()
  {
    @Override
    public void onClick(ClickEvent event)
    {
      final String username
         = usernameTextBox.getText();
      findUser(username);
    }

  });

  saveBtn.addClickHandler(new ClickHandler()
  {
    @Override
    public void onClick(ClickEvent event)
    {
      final User user
         = new User(  usernameTextBox.getText(),
         passwordTextBox.getText(),
                enabledChkBox.getValue());
      for(int i = 0;
         i < authListBox.getItemCount(); ++i)
      {
        user.addAuth(authListBox.getItemText(i));
      }
      saveUser(user);
    }
  });
}
```

Our service interface has two methods. One to find users, which takes a user name and returns a **`User`** object and one to save users which just takes a **`User`** object. The **`RemoteServiceRelativePath`** annotation is used in conjunction with the **`servlet-mapping`** element in `web.xml` to specify the URL for the service.

An asynchronous interface is also required to allow the client to make calls to the service interface, by specifying the callbacks used:

```
public interface UserServiceAsync
{
  void find(String username,
     AsyncCallback<User> callback);
  void save(User user,
     AsyncCallback<Void> callback);
}
```

The asynchronous interface must have the same name as the service interface, but appended with **`Async`** and must be in the same package. Put both interfaces into the **`uk.co.marauder.usermanager.client`** package as the service interface needs to be seen by both the client and the server.

The server side implementation, which usually goes in the server package (e.g. **`uk.co.marauder.usermanager.server`**), must both implement the service interface and extend GWT's **`RemoteServiceServlet`** (Listing 10).

To be callable, **`UserServiceImpl`**, must be configured in `web.xml` as a servlet. The **`<servlet>`** element is used to give the servlet a name and specify the class that implements it:

```
<servlet>
  <servlet-name>userManagerServlet</servlet-name>
  <servlet-class>
    uk.co.marauder.usermanager.server.
   UserServiceImpl
  </servlet-class>
</servlet>
```

Listing 10

```
public class UserServiceImpl
   extends RemoteServiceServlet
   implements UserService
{
  @Override
  public User find(String username)
  {
    return null;
  }

  @Override
  public void save(User user)
  {
  }
}
```

The url at which the servlet will be accessed must also be configured using the **`<servlet-mapping>`** element.

```
<servlet-mapping>
  <servlet-name>userManagerServlet</servlet-name>
  <url-pattern>/usermanager/user</url-pattern>
</servlet-mapping>
```

`<servlet-mapping>` uses the name of the servlet as defined by **`<servlet>`** and specifies the url pattern with **`<url-pattern>`**. The first part of the url pattern, before the slash, is the relative (to the server) path to the web application and the second part, after the slash, is the name of the remote service and must be the same as specified in the **`RemoteServiceRelativePath`** annotation. The remote service is now ready to call from the client.

To make a call to the remote service you first need to create an instance of the asynchronous interface. You only need to do it once, so make it a field of the **`UserManager`**:

```
public class UserManager implements EntryPoint
{
  private final UserServiceAsync userService
        = GWT.create(UserService.class);
  …
}
```

A call can be made on an interface implementation instance just like calling any method on any object, however you need to give it an implementation of the appropriate **`AsyncCallback`** interface (Listing 11).

Listing 11

```
private void findUser(final String username)
{
  userService.find(username,
     new AsyncCallback<User>()
  {
    @Override
    public void onFailure(Throwable caught)
    {
      Window.alert(caught.getMessage());
    }
    @Override
    public void onSuccess(User user)
    {
      if (user != null)
      {
        updateUser(user);
      }
      else
      {
        Window.alert(
           "Could not find: " + username);
      }
    }
  });
}
```

The **AyncCallback** interface has two methods you must implement, **onSuccess** and **onFailure**. The **onFailure** method is called if there is a failure on the server side, such as an exception being thrown and passes the failure back as a **Throwable** object. This object can be queried and the error message displayed to the user, in this case by way of an alert message box. The **onSuccess** method takes a parameter of the type specified as **AsynCallback**'s generic parameter. This is the object returned from the RPC call. All objects passed to or returned from RPC calls must implement the GWT **IsSerializable** marker interface, have a default constructor, which can be private, and be visible to the client and server side. RPC calls are entirely asynchronous, so calls return immediately and the **AsynCallback** methods are called sometime later. In this case, if the user has been found, a non-**null User** object is passed back and then passed on to the **updateUser** method for display in the interface. If it is not found **null** is passed back and the user is informed via a message box that the user could not be found. The **updateUser** method looks like this:

```
private void updateUser(User newUser)
{
  usernameTextBox.setText(newUser.getUsername());
  passwordTextBox.setText(newUser.getPassword());
  enabledChkBox.setValue(newUser.isEnabled());
  authTextBox.setText("");
  authListBox.clear();
  for(String s : newUser.getAuths())
  {
    authListBox.insertItem(s,-1);
  }
}
```

As you can see it takes the user name, password and enabled flag from the **User** object and uses them to set the text or value in the appropriate widget. It then clears the authorities list box and the inserts the authorities from the **User** object.

The **saveUser** method is similar (Listing 12).

Again the **onFailure** method reports failure to the user. The remote service save method return type is **void**, there is nothing returned. However, the **onSuccess** method must still be implemented to indicate when the method returns. Its single parameter is therefore of type **Void**. If the **User** object is successfully saved, the **clear** method is called to clear all the widgets and a message box is displayed to the user indicating success. The **clear** method is implemented as follows:

```
private void clear()
{
  usernameTextBox.setText("");
  passwordTextBox.setText("");
  enabledChkBox.setValue(false);
  authTextBox.setText("");
  authListBox.clear();
}
```

It clears the contents of all text boxes, sets the check box to unchecked and clears the authorities list.

Integrating the data access layer

If you run the application now and try to find and save users you will get messages telling you that you have saved users successfully and messages that none of the users you have saved can be found. This is because the **find** and **save** methods of **UserServiceImpl** are yet to be implemented. The required implementation is described in my 'Data Access Layer Design for Java Enterprise Applications' article (and can be downloaded here [9]). First drop the classes and interfaces from the data access layer into a newly created **uk.co.marauder.usermanager.dao** package. The required classes and interfaces are:

- **AbstractDMRegistry**
- **DMRegistry**
- **HibernateTemplateDMRegistry**
- **UserDataMapper**
- **UserHibernateDataMapper**
- **UserManager** (not to be confused with the client side **UserManager**).

Listing 12

```
private void saveUser(final User user)
{
  userService.save(user,
     new AsyncCallback<Void>()
  {
    @Override
    public void onFailure(Throwable caught)
    {
      Window.alert(caught.getMessage());
    }
    @Override
    public void onSuccess(Void result)
    {
      clear();
      Window.alert("Saved " + user.getUsername()
         + " successfully");
    }
  });
}
```

You will also need the Hibernate JAR and the JARs on which it depends. Put these in `WEB-INF\lib` in the `war` directory and add them to the User Manager project. The `User.hbm.xml` file is also required so that Hibernate knows how to persist the **User** object. Put the file in the **uk.co.marauder.usermanager.model** package alongside the `User.java` file. The Hibernate **SessionFactory**, which we will see how to create in a minute, also requires the `hibernate.properties` file. Put it in the `src` folder at the root package level.

The data access layer **UserManager** class will handle the persisting and retrieving of **User** objects. It is used by both the **find** and **save** methods of the **UserServiceImpl** class, but, unlike the **SessionFactory** class, is not thread safe so a **SessionFactory** object should be created in the constructor and stored as a field and the **UserManager** created on demand (Listing 13).

After creating the **UserManager** object, finding and saving **User** objects is trivial (Listing 14).

All that is required to find a user is to pass the user name to the **load** method of the **UserManager** and return the **User** object. **null** is

Listing 13

```
public class UserServiceImpl
   extends RemoteServiceServlet
   implements UserService
{
  final SessionFactory sessions;

  public UserServiceImpl()
  {
    sessions = new Configuration()
       .addClass(User.class)
       .buildSessionFactory();
  }
  private UserManager getUserMgr()
  {
    final HibernateTemplate hibernateTemplate
       = new HibernateTemplate(sessions);
    return new UserManager(
       new HibernateTemplateDMRegistry(
       hibernateTemplate));
  }
  ...
}
```

Listing 14

```
public class UserServiceImpl extends
RemoteServiceServlet implements UserService
{
  private final UserManager mgr;
  ...
  @Override
  public User find(String username)
  {
    return getUserMgr().load(username);
  }
  @Override
  public void save(User user)
  {
    getUserMgr().save(user);
  }
}
```

returned and handled by the client if the user cannot be found. To save a **User** object, just pass it to the **save** method.

If you run the application and try to save a user you will see that it is serialised to the database successfully. However, if you try to find that user again you will receive an error:

```
The call failed on the server; see server log for
details.
```

If you look at the output in the Eclipse console window (or equivalent) you will see the following exception message:

```
com.google.gwt.user.client.rpc.SerializationException:
  Type 'org.hibernate.collection.PersistentBag' was
  not included in the set of types which can be
  serialized by this SerializationPolicy or its Class
  object could not be loaded. For security purposes,
  this type will not be serialized.
```

This is due to the way Hibernate handles retrieving collections. Instead of putting the items into an **ArrayList<String>**, which GWT knows how to serialise via RPC, it puts them into its own class type **org.hibernate.collection.PersistentBag** which GWT does not know how to serialise, even though **PersistentBag** implements the **List** interface. Unlike Java, which does not need to know at compile time how an interface is implemented, GWT needs to know about the complete hierarchy if it is going to be able to use all the classes in it.

The Google Web Toolkit is a far more powerful and elegant tool for creating web application front ends than any of the JSP based libraries used by most

The easiest way to get around this problem is to copy the **User** object returned by Hibernate into a fresh **User** object that uses an **ArrayList<String>** rather than a **PersistentBag**. This could be done manually using a copy-constructor, but that would get tedious as domain objects get more complex. Helpfully there is a library, called Dozer [9] that recursively copies data from one object to another. The Dozer JAR can be downloaded from the Dozer website and should be placed in the `WEB-INF\lib` directory along with its dependant JARs `commons-beanutils` and `commons-lang` and added to the User Manager project.

Dozer's use is very simple. Just create a **DozerBeanMapper** and call the **map** method with the object to copy and the type of the object to copy it to and the copy is returned. The only draw back is that, unlike Hibernate, Dozer can only use public accessors. This is why all the **set** methods of the **User** class, as well as the **get** methods, must be public:

```
@Override
public User find(String username)
{
  User user = getUserMgr().load(username);
  if (user != null)
  {
    user = new DozerBeanMapper().map(
       user,User.class);
  }
  return user;
}
```

If the object to copy is **null**, **map** throws an exception therefore the object to be copied needs to be checked for **null** first. If you run the application now you will be able to both save and find users.

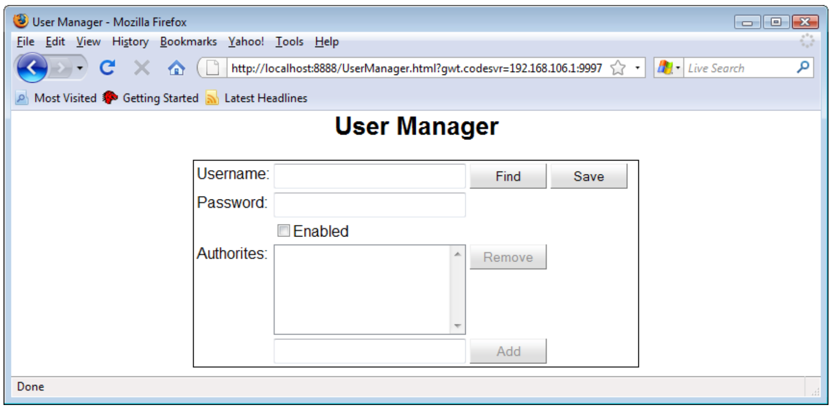
Sexing-up the interface

The basic functionality of the application is now complete. If you look back at Figure 5 you will see that the main problems with the layout are the buttons being different widths and the authorities list box not being the same width as the text boxes. This can all be fixed with CSS. Replace the contents of the existing `UserManager.css` file in the `war`'s `WEB-INF` directory with the Listing 15, and refresh the application in the browser. There is no need to restart the application to apply CSS changes. You will see that all the text boxes and the list box are the same width and all the buttons are the same width. Most GWT widgets have a default style name, such as **gwt-TextBox** or **gwt-Button**. These are used in the CSS file to specify things like font size and widget width. The GWT Grid widget does not have a default style name, so one must be added manually using the **setStyleName** method. It also does not have a way to specify the cell alignments in CSS so this too must be done manually by subclassing **Grid** in a similar way to **ListBox** (Listing 16).

The constructor, as well as assigning a style name, iterates through each cell in the grid and sets the vertical and horizontal alignment.

Figure 6 shows the far more aesthetic version of the application with the CSS changes.

Figure 6


User Manager - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
http://localhost:8888/UserManager.html?gwt.codesvr=192.168.106.1:9997
Live Search
Most Visited Getting Started Latest Headlines
User Manager
Username:
Find
Save
Password:
Enabled
Authorites:
Remove
Add
Done

Finally

There is plenty more you would want to do to this web application, such as disable all widgets when the find is in progress, provide a list of users to choose from instead of relying on the find, provide a list of authorities and prevent duplicates etc, but implementing these is not necessary to demonstrate creating a GWT application with RPC and Hibernate.

The Google Web Toolkit is a far more powerful and elegant tool for creating web application front ends than any of the JSP based libraries used by most. Here I have demonstrated just how easy it is, as well as demonstrating how to avoid some of the pitfalls. GWT can do a lot more than I have shown here and I hope to demonstrate further features in future articles. ■

Listing 15

```
h1
{
  font-size: 2em;
  font-weight: bold;

  text-align: center;
}

.gwt-Label
{
  font-size: 12pt;
}

.gwt-Button
{
  font-size: 12pt;
  width: 80;
}

.gwt-TextBox
{
  font-size: 12pt;
  width: 200;
}

.gwt-PasswordTextBox
{
  font-size: 12pt;
  width: 200;
}

.gwt-listBox
{
  font-size: 12pt;
  width: 200;
}

.gwt-CheckBox
{
  font-size: 12pt;
  width: 200;
}

.gwt-Grid
{
  border: 1px solid black;
}
```

Listing 16

```
public class Grid extends com.google.gwt.user.
client.ui.Grid
{
  public Grid(int rows, int columns)
  {
    super(rows, columns);
    this.setStyleName("gwt-Grid");

    for(int x = 0; x < columns; ++x)
    {
      for(int y = 0; y < rows; ++y)
      {
        this.getCellFormatter()
          .setAlignment(  y, x,
              HasAlignment.ALIGN_LEFT,
              HasAlignment.ALIGN_TOP);
      }
    }
  }
}
```

Appendix A: Installing GWT 2.0 Eclipse plugin

Although Eclipse [10] is not required for working with GWT it can make development much easier. If you do not want to use Eclipse the SDK can be downloaded by itself and a supplied tool used to generate projects from the command line. The GWT plugin is available for versions 3.5 (Galileo), 3.4 (Ganymede) and 3.3 (Europa) of Eclipse. More information is available from the GWT Plugin [11] download page.

The GWT plugin is best installed into a fresh install of Eclipse. I am currently using Galileo for Java EE developers (**`eclipse-jee-galileo-SR1-win32`**), but the plugin works just as well with the standard version. The GWT plugin has a dependency on Eclipse's Web Standard Tools (WST), which if not already installed, can be installed as follows:

1. Go to the Help menu and select Install new software.
2. If it isn't already present, manually add the appropriate update url: http://download.eclipse.org/releases/galileo/
3. Then expand Web, XML, and Java EE Development from the list and select Eclipse Web Developer Tools.
4. Click next and follow the install instructions.
5. Restart Eclipse when asked.

Installing the GWT plugin is much the same process:

1. Go to the Help menu and select Install new software.
2. Add the update url: http://dl.google.com/eclipse/plugin/3.5 (or the apropriate link for your version of Eclipse)
3. Tick Google Plugin for Eclipse 3.5 and Google Web Toolkit SDK 2.0.0. Untick Google App Enginer SDK 1 unless you particularly want to install it. It is not needed for the GWT plugin.
4. Click next and follow the install instructions.
5. Restarted Eclipse when asked.

More detailed instructions can be found on the Google Plugin for Eclipse 3.5 (Galileo) Installation Instructions [12] page of the GWT website.

References

[1] Model View Controller: http://www.marauder-consulting.co.uk/Model_View_Controller.pdf

[2] Apache Struts: http://struts.apache.org/

[3] Google Web Toolkit: http://code.google.com/webtoolkit/

[4] Apache Tomcat: http://tomcat.apache.org/

[5] Organising Projects Development Guide: http://code.google.com/webtoolkit/doc/latest/DevGuideOrganizingProjects.html

[6] Data Access Layer Design for Java Enterprise Applications: http://www.paulgrenyer.com/Data_Access_Layer_Design_for_Java_Enterprise_Applications.pdf

[7] Swing Layout Managers: http://java.sun.com/docs/books/tutorial/uiswing/layout/using.html

[8] Download Data Access Layer project: http://paulgrenyer.net/dnld/dataaccesslayer.zip

[9] Dozer: http://dozer.sourceforge.net/

[10] Eclipse: http://www.eclipse.org/

[11] Google Web Toolkit Download: http://code.google.com/eclipse/docs/download.html

[12] Google Plugin for Eclipse 3.5 (Galileo) Installation Instructions http://code.google.com/eclipse/docs/install-eclipse-3.5.html

More About Bash

Ian Bruntlett discovers more *nix shell tools.

Here are some more titbits of information about using the Linux command line prompt (in Linux parlance 'a shell') – **bash**. As a technicality I'd like to point out that **bash** is just another program and can be replaced with a different program, also acting as a shell. I wrote [1] and [2] while learning about **bash**. This short article is intended to further inform the reader, building on the previous articles. I am trying to balance this text, hopefully being thorough without stating the obvious. Note: I use *nix as an abbreviation of Unix, GNU/Linux.

Pipelines

Even DOS had pipelining, where the output of one program – eg **grep** – could be piped to the input of another program e.g. **more**.

```
grep *.c | more
```

This invokes **grep**, passes it the parameter ***.c** and sends the resulting output to a temporary file. When **grep** has finished, **more** is executed with the temporary file sent to the input of **more** which lists the output in pages on the screen.

This idea was borrowed from Unix. However, the implementation wasn't complete. In DOS, **grep** would be started, its output saved in a temporary file, **grep** would terminate OK and the resulting temporary file sent as input to the more command.

In Unix, however, **grep** and **more** are started as a pipeline of programmes. How does this work? Well both **grep** and **more** are started together, with a 'pipeline' connecting the output (**stdout**) of **grep** to the input (**stdin**) of **more**.

Arcane commands – man, info and apropos

It's pretty easy to pick up all the DOS commands. *nix commands are documented in *Big Fat Books* from O'Reilly and many of the commands seem trivial. However, in *nix, the sum of the combined parts is greater than the whole. See my review of *Linux and the Unix Philosophy*, a book that hammers home the tenet that all data should be text based (however I still believe that binary files are important). So if you're looking at a *nix shell script and you're wondering what the individual commands do, you can either buy a book or use the inbuilt help. Both have their values. My copy of the *Linux Pocket Guide* has got lots of notes in the margins. Despite liking the dead tree version, I think one day I will mainly use the on-line help.

Quite often people are encouraged to read the **man** pages. The **man** command, short for 'manual', searches a database. On Rutherford PC running Ubuntu 9.10, that database is scattered into folders held inside `/usr/share/man`. For example running **man man** gives you 600+ lines of text that describe **man** and lists its command line options. It handles paging so you'll never need to type **man widget | less**.

The **info** command also searches online information (this time in `/usr/share/info`). The twist is that the **info** pages are an early form of hypertext. Typing in the command **info** brings you into the top level of the **info** tree and gives information to guide the user through **info**'s particular interface.

The **apropos** command searches the **man** pages for relevant pages. It is identical to running man **-k** keyword but I learnt to **apropos** long before finding out about **man-k**.

Testing – regular expressions (regexps)

Regexps [3] [4] can get very complicated and, therefore, difficult to test. The simplest test can be made from the command line using two commands – **echo** and **grep** (short for global regular expression print).

```
echo The quick brown fox | grep 'o'
```

This could be tested on a more industrial scale by processing a whole file of lines using the command **grep <expression> test_input.txt** For example, here is my `test_input.txt`:

> The quick brown fox jumped over the lazy brown dog.
>
> The quick brown fop jumped over the quick brown fox.
>
> Humpty Dumpty could not be put back together again.

```
ian@Rutherford:~/wrk$ grep -E 'fo?' test_input.txt
The quick brown fox jumped over the lazy brown dog.
The quick brown fop jumped over the quick brown fox.
```

Testing and developing complex pipelines

Here is an example of a slightly complex pipeline that lists programs in sorted order:

```
ps x | awk '{print $5, $4}' | sort
```

It's not that complicated and I built it up, step by step at the keyboard. However, there are times when a tool could make life easier. There are two particularly useful commands – **cat** and **tee**.

The tee command – peeking into a pipeline.

Similar to the **cat** command (see next paragraph), **tee** copies its input (**stdin**) to its output (**stdout**). However, it can be told to duplicate the data flowing through the pipeline. By giving a filename as a parameter, **tee** will 1) copy its input (**stdin**) to its output (**stdout**) and 2) send a copy of that data to the named file.

The cat command – injecting a file into a pipeline.

I usually overlook the **cat** command, preferring to use the **more** command to see a text file page by page. The parameters passed to **cat** tell it what to output. In its simplest form, **cat wibble.txt** simply copies the contents of `wibble.txt` to its output (**stdout**). There can be more parameters passed to **cat** – e.g. **cat wibble.txt dibble.txt** would instruct **cat** to copy the contents of `wibble.txt` to its output (**stdout**) followed by the contents of `dibble.txt`.

However, **cat** has a special ability. A parameter can either be a filename or the dash character **-**. When **cat** is running and it reaches the **-** parameter, it copies its own input (**stdin**) to its output (**stdout**) before going on to process any remaining parameters.

What is the use of that? If you want to inject data into a pipeline, this is where the **cat** command begins to be useful.

```
ps x | tee ps_output.txt | awk '{print $5, $4}'
| sort | cat heading.txt -
```

IAN BRUNTLETT

On and off, Ian has been programming for some years. He is a volunteer system administrator for a mental health charity called Contact (www.contactmorpeth.org.uk). As part of his work, Ian has compiled a free Software Toolkit (http://contactmorpeth/wikispaces.com/SoftwareToolkit).

Inspirational (P)articles

Dáire Stockdale shares a recent source of inspiration.

Thanks to Dáire Stockdale, a senior software engineer working for http://www.realtimeworlds.com, for sharing with us inspiration from a a seemingly unlikely source: a book about evolution. The idea that complicated systems can be formed from simple rules is beautiful and inspirational.

I'm not sure I have the time or wordcraft to write a piece, but I did find reading Dawkins latest book, *The greatest Show on Earth*, explaining the inner details of evolution to be almost an epiphany.

I suspect that similar to many reading this list, biology was my least favourite science, I preferred maths and physics, and of course I thought I understood evolution. However Dawkins' book highlights the elegant simplicity of our world in such a beautiful way, and I found it really appealed to my programmer's heart. Whereas previously I had understood evolution to mean a process that was fantastically complex, but which worked due to the sheer timescale available to it (a 'brute force' algorithm, searching for best fits), he explains that in fact it is even simpler, and from the moment the process begins, it necessarily and quite rapidly produces results.

a process that was fantastically complex, but which worked due to the sheer timescale available to it

He explains, far better then I can here, how the apparent complexity of nature really is emergent from a few 'simple' rules, and that its not so hard to fashion the changes we see around us. I think programmers will really appreciate and 'get' these sections of the book, more than most. After reading the book, I can't see cats and dogs without appreciating just how easy it is for natural selection to morph between them. Also, when he talks of the impossibility of knowing what a given DNA sequence 'produces' without actually 'growing' it [1] I was reminded of Turing program stopping problems. I also found myself admiring the painstaking attention to detail gone to in by experimental biologists in proving, through example, evolution at work.

Even if you think you understand evolution, this book will probably surprise and enlighten you :) After reading my copy I have gone and bought some more for friends :)

Note

[1] My terminology. I'm not sure what this process might actually be called.

More About Bash (continued)

This command line lists processes and sends a copy of that data to `stdout` and to `ps_output.txt`. `awk`'s input (`stdin`) is tied to `tee`'s output (`stdout`) which prints the 5th and 4th text fields of the `ps` output.Finally, `cat` is used to send a heading to `stdout`, closely followed by `stdin` to `stdout`.

For instance, if I wasn't sure what ps was outputting, I could try:-

```
ps x | tee ps_output.txt | awk '{print $5, $4}' |
sort
```

Once the pipeline has finished executing, looking at `ps_output.txt` might yield clues regarding the pipeline's behaviour. ■

References

[1] Ian Bruntlett, 'Interpreting Custom Unix Shell Scripts' in *CVu* 21.4, September 2009
[2] Ian Bruntlett, 'Beyond Pipelining Programmes in Linux' in *CVu* 21.5, November 2009
[3] Tony Stubblebine, *Regular Expression Pocket Reference*, O'Reilly. This book refers to Mastering Regular Expressions by Jeffrey Friedl (also from O'Reilly).
[4] D. J. Barrett, *Linux Pocket Guide*, O'Reilly.

Bibliography

Mike Gancarz, *Linuz and the Unix Philosophy*, Digital Press.
Arnold Robbins, *sed & awk Pocket Reference*, O'Reilly.

Code Critique Competition 62

Set and collated by Roger Orr.

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

Last issue's code

I am trying to write a simple two-way map to allow items to be addressed by two different keys but I'm getting some odd behaviour. I've written a simple test program and everything works fine except for the last line output – I expect the last size to be 1 but I get 2. Can you help?

Last issue's code is shown in Listing 1.

Listing 1

```
//twowaymap.h
#include <map>
template <typename T, typename U>
class twoway_map
{
public:
  typedef typename
  std::map<T,U>::size_type size_type;
  void insert(const T& key1, const T& key2,
    const U& value)
  {
    U & p = map1[key1] = value;
    map2[key2] = &p;
  }
  void erase1(const T& key1)
  {
    if (map1.find(key1) != map1.end())
    {
      for (typename std::map<T,U*>::iterator
        it = map2.begin();
        it != map2.end(); ++it)
      {
        if (it->second == &map1[key1])
        {
          map2.erase(it->first);
          map1.erase(key1);
        }
      }
    }
  }
  void erase2(const T& key2)
  {
    if (map2.find(key2) != map2.end())
    {
      for (typename std::map<T,U>::iterator
        it = map1.begin();
        it != map1.end(); ++it)
      {
        if (&it->second == map2[key2])
```

Listing 1 (cont'd)

```
        {
          map1.erase(it->first);
          map2.erase(key2);
        }
      }
    }
  }
  U& at1(const T& key1)
  {
    return map1[key1];
  }
  U& at2(const T& key2)
  {
    return *map2[key2];
  }
  size_type size() const
  {
    return map1.size();
  }

private:
  std::map<T,U> map1;
  std::map<T,U*> map2;
};

//test.cpp
#include "twowaymap.h"
#include <string>
#include <iostream>

struct colour
{
  colour() {}
  colour(int r, int g, int b)
  {
    rgb = r << 16 | g << 8 | b;
  }
  operator int() { return rgb; }
  int rgb;
};
using std::cout;
using std::endl;

int main()
{
  twoway_map<std::string, colour> m;
  m.insert("Red", "Rouge", colour(255,0,0));
  m.insert("Green", "Vert", colour(0,255,0));
  m.insert("Blue", "Bleu", colour(0,0,255));
  cout << "size: " << m.size() << endl;
  cout << std::hex << m.at1("Red") << endl;
  cout << std::hex << m.at2("Vert") << endl;
  cout << std::hex << m.at1("Blue") << endl;
  m.erase2("Bleu");
  cout << "size: " << m.size() << endl;
  m.erase1("Blue");
  cout << "size: " << m.size() << endl;
  m.erase1("Red");
  cout << "size: " << m.size() << endl;
}
```

ROGER ORR

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

Critiques

Balog Pal <pasa@lib.hu>

I started wondering even before I got to the code. The target is said to be a 'two way map' and continues, 'as it shall have two keys'. For me a two-way map means a different beast, a map where you can swap the roles of key and value, and knowing either one can fetch the other. What is described here I'd name maybe a two-key or alternative-keyed map. And I have used many of the first kind, but nothing like this.

So the first thing I do is try design one such class in my head with essential functions and test cases. Then compare that with the presented code. The interface does not look hard, we'll need some insertion with the keys and value and fetch by either key. Possibly erase by either key too. How many test cases will that need? .... And here comes some explosion. Like a zillion, including repeated inserts and erases of the two-key combinations. The real problem is I cannot tell the expected behaviour of almost any of those.

Here I look at the code for some help, to find indeed the basic functions are like imagined, but neither the implementation nor the 'test' suggests the solution. The visible insert works with triplets – good – and we have the sneaky inserts with the **atX()** functions, that make no sense, so look more like a bug. Suppose not having that, let's start with empty map and insert triplets (a, A, 1), (b, B, 2). After that what should happen on inserts: (a, A, 1) (a, A, 2) (a, C, 1) (a, B, 1) (b, A, 1) ? And after those, issuing erase on any of the existing keys, or on a non-existing one? Even without an extra insertion, what erase shall do, remove the whole triplet or let the other half-pair linger? The 'testing' part just writes out the result without telling what is expected. The 'just the last' is bad is hardly enough information with the code that smells undefined behaviour with the pointer and iterator juggling. The expected result of the second erase would be interesting but all we're told it is 'okay' at executing there...

Trying to run it lands in a crash right at the first erase attempt. Incrementing the iterator that was just invalidated by the erase... Fixing that the output is 2,2,2, whether just by accident is to be seen.

So let's look the code for problems. I start with the **struct colour**. It is hardly meant seriously – probably just dropped in to have something to drive the test, but still it is better put it into shape.

- Its default ctor does not init the state. Member **rgb** shall be initialised to something (say 0), preferably in the init list. The way it is it is hardly a 'constructor'.
- In the other ctor the expression is probably correct, but only due to wild chance. As operator precedence is a royal mess in C, expressions shall use ()s, like **(r << 16) | (g << 8) | b**.
- The most accepted approach to stuff using bit operations (**|**) is to use unsigned types, not signed ones. So I'd use **unsigned int** instead of **int** for the member and the operator.
- The second **ctor** would provide wild results if passed values not in 0–255 range so some checks or an **assert** would be a benefit.

On to **twoway_map**. The first thing to inspect is how the state is stored. We see two maps, one holding the first key and the value, and another that has the second key and a pointer to the value. In member functions we see iteration over the whole maps in a search. This makes the choice a suspect. With maps couldn't we do the job just through lookups? And using pointers imposes fragility. It is not an error in itself, but drags in the obligation to be extremely careful about the pointed thing's lifetime and cross consistency.

The first function, **insert**, looks simple. And it does what is likely intended for empty maps, or new keys, creating an entry in the main holder and a referring one in the other. A pointer to an element in **map1** is stored so we'll have to review any erase operations to remove all the referring pointers. If the passed **key1, key2** argument already exists in the map, multiple pointers can refer to the same node in **map1** or some nodes may be not referred any longer. We can't tell whether it is a bug or a feature.

The **size** function returns size of the first map. To me that would imply the author thought the two maps will have the same element count. It is clearly not enforced in any way, with proper calls to just the insert function alone we can create more or less entries in either map. So the meaning of the return value is moot.

The assessor functions are non-**const**, and return non-**const** reference, that may work like **map**'s **[]** operator. Though this class has the state in two maps and the functions show no attempt to keep them in synch. I'd bet this is a serious overlook on the creator's part, who did not consider here the map's feature of auto-inserting nodes. Possibly they even started out as **const** functions and **const** got removed due to a compiler error. A correct implementation should use **find()** and handle the not found case some way. As it stands **at1()** would create a default-initialized value to be returned, while **at2()** would dereference a null pointer invoking problems.

The **erase** members are the most interesting beasts – with all kind of problems. One already mentioned earlier, **erase** happens on the actual iterator that has a **++** in the **for**. Any code that walks a **map** (and many like collections) shall have the bare **++it** only on the **else** branch. Usually **map.erase(it++)** is seen on the other, here **it->first** is called for, that invokes an extra search for the already found element. That hardly makes sense unless **multimap** was used. The placement of **erase(key)** is off too, normally it should either follow the loop or being followed by a break;

On my reviews even a fraction of discovered problems would invoke 'square 1' – and for a fix we better really go back to the design table. Not having the requirements I make up some, to show a solution route.

The one in Listing A is created as a version to be easily read and verified against expectations. (Beyond juggling with first and second...).

- The class is called **twokey_map** better describing the purpose.
- The template has a third argument to allow different types for the two keys, with the default of using the same type, so usage is similar to the original, with no real cost, and even more clarity in the code
- The state is held in two maps without pointers, or any other hard cross-reference. One binds **key1** to **key2** and the other backwards. So we can find out the key-pair having either of the keys, and by a simple map lookup. The first map also stores the value, to avoid another map with the same key.
- The design rule of the state here is that the key pairs are 'exclusive', instances of **key1** have 1-1 relation to those in **key2**, that is enforced in both **insert** and **erase** operations.
- fetch operations are **const** key-wise, attempt to ask a key not stored is an error (signaled by exception here). A non-**const** version may be provided that uses the usual add/remove const trick, making the value part modifiable, if it is needed.

Understanding the design constraints should make the code evident, so just a light explanation:

- Private **typedef**s are for convenience. **map** binds both **key2** and **value** to **key1**, and the stock **pair** is used instead of a custom struct. **backmap** provides the reverse lookup of the keys.
- The find functions are the error filters to be used when we expect the key be there. (unfortunately **map** has no **const []** operator that would do that stock).
- **insert()** is similar to the original code's version, but first makes sure to clean up, and leaving no doubt of the possible post-condition.
- **at1()** does the obvious thing, and **at2()** just looks up the first key to be usable with **at1()**.
- the **erase** operations do nothing if the key-pair is not there, and remove the entries from both maps otherwise.
- **size()** now makes sense, as all operations ensure the two maps have the same element count.

This is good for a reference version that does the job, but has many redundant lookups. To cover those a different implementation can be created. The code in Listing B is supposed to replace the marked section in the first version. Importantly it has the very same public interface, and shall work with the same test suite. The goal was to keep lookups to the minimum. (It is easier to understand and review having the first version, in practice it is a good idea to have solid code before doing optimizations.)

- The first map is kept the same, and as it did the whole job optimally on **`key1`**-using functions. The other map now stores an iterator to the first map as the 'value' instead of the key itself, so lookups in the first map are avoided. We can do that because the other design constraints ensure proper pairs with the same lifetime, no ugly cases for invalidated iterators.
- The **`find`** functions left a single use and got inlined.
- **`at1()`** is really the same as before, just losing a readability point to a missing abstraction barrier.
- **`at2()`** fetches the iterator to the first map ready to use, nuking one key lookup.
- The **`erase`** functions stayed exactly with the same source thanks to our **`typedef`**s, and the luck that **`map`**'s **`erase`** has an overload, and the code switched to the one taking iterator from one taking the key.

The massive-looking change is in the **`insert`** member. Demonstrating how simplicity is flushed as we try to squeeze operations. Content from **`erase()`** is getting inlined so we can capture some of the internal state.

If the entry for **`key1`** is present it is no longer removed then inserted, but just reassigned in place.

A test suite is not provided to save space.

Listing A

```
#include <map>
#include <utility>
#include <stdexcept>

// class to map a value to two alternative keys
template <typename KEY, typename VALUE,
          typename KEY2 = KEY>
class twokey_map
{
private:
  typedef std::map<KEY,
    std::pair<KEY2, VALUE> > tMap;
  typedef typename tMap::const_iterator
    tMapIter;
// ->
  typedef std::map<KEY2, KEY> tBackMap;
  typedef typename tBackMap::const_iterator
    tBackMapIter;

  tMap map;  // main holder of value,
    // auxillary info is the second key
  tBackMap backmap; // links the other key to
    // the main holder

private:
// helpers to access element that is expected
// to be there
  tMapIter find1(const KEY& key ) const
  {
    const tMapIter res = map.find(key);
    if(res == map.end())
      throw std::out_of_range("no key1");
    return res;
  }

  tBackMapIter find2(const KEY2& key2 ) const
  {
    const tBackMapIter res =
      backmap.find(key2);
    if(res == backmap.end())
      throw std::out_of_range("no key2");
    return res;
  }
public:
// overwrites any existing association on
// either key
  void insert(const KEY& key, const KEY2&
    key2, const VALUE& value)
  {
    erase1(key);
    erase2(key2);
    backmap[key2] = key;
    map[key] = std::make_pair(key2, value);
  }

// throws if called for non-existing key
  const VALUE& at1(const KEY& key) const
  {
    return find1(key)->second.second;
  };

// throws if called for non-existing key
  const VALUE& at2(const KEY2& key2) const
  {
    return at1( find2( key2 )->second );
  }
// <-

// no-op if called for non-existing key
  void erase1(const KEY& key)
  {
    const tMapIter it = map.find(key);
    if(it == map.end())
      return;
    backmap.erase(it->second.first);
    map.erase(it);
  }

// no-op if called for non-existing key
  void erase2(const KEY2& key2)
  {
    const tBackMapIter it =
      backmap.find(key2);
    if(it == backmap.end())
      return;
    map.erase(it->second);
    backmap.erase(it);
  }

  void clear()
  {
    map.clear();
    backmap.clear();
  }

  typedef typename tMap::size_type size_type;
  size_type size() const
  {
    return map.size();
  }
};
```

Listing B

```
  typedef std::map<KEY2, tMapIter> tBackMap;
  typedef typename tBackMap::const_iterator
    tBackMapIter;

  tMap map;  // main holder of value,
    // auxillary info is the second key
  tBackMap backmap; // links the other key to
    // the main holder

public:
// overwrites any existing association on
// either key
  void insert(const KEY& key, const KEY2&
    key2, const VALUE& value)
  {
    {
      const tBackMapIter it2 =
        backmap.find(key2);
      if(it2 != backmap.end())
        map.erase(it2->second);
    }

    typename tMap::iterator it1 =
      map.find(key);

    if(it1 != map.end())
    {
      backmap.erase(it1->second.first);
      it1->second =
        std::make_pair(key2, value);
    }
    else
      it1 = map.insert(
        std::make_pair(key,
        std::make_pair(key2, value)) ).first;

    backmap[key2] = it1;
  }

// throws if called for non-existing key
  const VALUE& at1(const KEY& key) const
  {
    tMapIter it = map.find(key);
    if(it == map.end())
      throw std::out_of_range("no key1");
    return it->second.second;
  };

// throws if called for non-existing key
  const VALUE& at2(const KEY2& key2) const
  {
    const tBackMapIter it =
      backmap.find(key2);
    if(it == backmap.end())
      throw std::out_of_range("no key2");
    return (it->second)->second.second;
  }
```

Commentary

As Pal wrote when thinking about test cases ‘The real problem is I cannot tell the expected behaviour of almost any of those.’ In particular, it is unclear what the code should do when one of the keys is a duplicate.

Pal also included the following comment along with his entry: ‘This is another of those poor quality examples, where the context is too thin to show a solution. So we're stuck with what-ifs and pointing at mistakes in the code without really going out to the light....’

I must agree with him, at least in part. In actual practice when people present code like this they often fail to explain what they're trying to do; fortunately you can ask questions to elucidate the purpose of the code and point out corner cases that the original design missed. This isn't always the case – for example when maintaining a program after the original author has moved on, but hopefully in these cases there are other ways to find out the code's desired behaviour.

However, as the code critique is non-interactive, I will try to add a little more context to the problem in future.

This isn't a problem with the bug in the sample program the user has provided though, as they chose unique keys; so we can resolve the bug here without necessarily needing to decide on the correct response to duplicate keys.

Many modern C++ environments provide extensive help with debugging usage of the standard library – a good reason for preferring it, in general, to hand written containers, so the first thing is to turn on the appropriate settings and see if additional light is thrown.

For example with g++ you might use `-D_GLIBCXX_DEBUG`; with this option executed the program now generates the error message:

```
attempt to increment a singular iterator.
```

Or, using the debugging runtime with Visual Studio, for example with /MDd, you get an exception at the point of error: the `++it` in the loop inside `erase1()`.

Either, or even both, of these techniques can greatly assist debugging this sort of trouble. As Pal stated, the problem is that *modifying* the map while iterating through it is dangerous, and must be done using the iterator itself (and taking care even so). The debugging messages produced will make this easier to identify. The easiest way to avoid the trouble in this case is simply to break out of the loop. After all, if we have found the key there's no point continuing to search the map anyway.

The Winner of CC 61

As Pal was the only entrant I am awarding him the prize. However, I have no compunction in doing so as his critique provided good coverage of the problems with the code.

Code Critique 62

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

A new version of the C++ standard is nearing finalisation and is partly implemented in both Microsoft's Visual Studio 2010 compiler (out in beta) and the latest versions of g++ (also available on cygwin for Windows users). So I thought it was a good time to present a code sample trying to use one of the new standard libraries: `<regex>`.

I found some example code implementing a minimal subset of the well-known `grep` command that was originally written to use the boost `regex.hpp` library and converted it to use the new standard library. I was expecting this to be a simple matter of changing the include file and some namespace names, but to my surprise the code didn't work correctly on Visual Studio 2010; it compiled cleanly and at first appeared to run successfully but when I used with the `-i` option (for a case insensitive search) the program failed with a runtime error:

```
C:>grep -i test
Error: regular expression error
```

So this issue's code is slightly unusual in that I'm also providing part of the `<regex>` header to help you identify the problem. As usual, please also use the opportunity to comment on other parts of the code.

The code is in Listing 2.
The relevant bits of the implementation of `<regex>` are in Listing 3.

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 helps overseas members who typically get the magazine much later than members in the UK and Europe.

Listing 2

```
// define USE_BOOST for the old way
#ifdef USE_BOOST
#include <boost/regex.hpp>
using namespace boost;
#else
#include <regex>
#if defined(_MSC_VER) && (_MSC_VER <= 1500)
// Visual Studio 2008 has it in 'tr1'
using namespace std::tr1;
#else
using namespace std;
#endif
#endif

#include <map>
#include <stdio.h>
#include <stdlib.h>
static const int MAX_LINE_LEN = 65536;
std::map<std::string, bool> option;

void usage()
{
  printf( "Usage: grep [-i -l] <pattern>\n");
  printf( "Options:\n");
  printf( "-i        case insensitive\n");
  printf( "-l        line numbers\n");
}

int process(regex expbuf)
{
  char buff[ MAX_LINE_LEN + 1 ];
  long lineno = 0;
  /* Start reading the file */
  while (fgets( buff, MAX_LINE_LEN, stdin))
  {
    lineno++;
    bool rc = regex_search(buff, expbuf);
    if (rc)
    {
      if (option["l"])
        printf("%li:", lineno);
      printf(buff);
    }
  }
  return 0;
}

int main(int argc, char **argv)
{
  regex expbuf;
  int cflags = regex_constants::ECMAScript;
  if ( argc <= 1)
    usage(), exit(1);
  argv++;
  argc--;
  if (**argv == '-')
  {
    switch (*++*argv)
    {
    case 'i':
    case 'l':
      option[*argv]++;
      break;
    default:
      printf( "-i or -l expected\n");
      exit(1);
    }
    argv++;
    argc--;
  }
```

Listing 2 (cont'd)

```
  try
  {
    if (option["i"])
    {
      cflags |= regex_constants::icase;
    }
    expbuf.assign(argv[0], cflags);
  }
  catch (std::exception & ex)
  {
    fprintf(stderr, "Error: %s\n",
      ex.what());
    exit(1);
  }

  return process(expbuf);
}
```

Listing 3

```
enum option_type
{
  ECMAScript = 0x01,
  // ...
  icase = 0x0100,
  // ...
};

template<class charT,
  class traits = regex_traits<charT> >
  class basic_regex
{
//...
typedef option_type flag_type;
// ...
basic_regex& assign(const basic_regex& that);

basic_regex& assign(const charT *ptr,
  flag_type f = regex_constants::ECMAScript);

basic_regex& assign(const charT *ptr,
  size_t len,
  flag_type f = regex_constants::ECMAScript);

template<class s_traits, class A>
basic_regex& assign(
  const basic_string<charT, s_traits, A>& s,
  flag_type f = regex_constants::ECMAScript);

template<class InputIterator>
basic_regex& assign(InputIterator first,
  InputIterator last,
  flag_type f = regex_constants::ECMAScript);
// ...
};

typedef basic_regex<char> regex;

template<class charT, class traits>
bool regex_search(const charT *s,
  const basic_regex<charT, traits>& re,
  regex_constants::match_flag_type f =
    regex_constants::match_default);
```

Desert Island Books

Paul Grenyer strands Terje Slettebø with nothing but a few books and some music for company.

Although he has been rather quiet of late, Terje exploded onto accu-general only a few years ago. Terje always has a lot to say and makes an immense contribution to the ACCU. As with so many others, I have been lucky enough to meet him at the ACCU conference.

Terje Slettebø

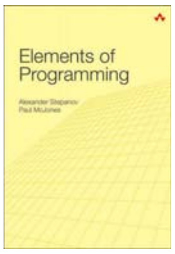
I was honoured when Paul asked me if I could write an instalment in this series. I've felt I've been quite out of the circulation, both regarding C++ and ACCU, for a while, but some things are timeless, and it would be timeless books (and albums) that you'd want to bring with you on a desert island (you'd never know if or when you'd be 'rescued').

Some others have wondered if we've all ended up on the same island (would it be a desert island, then, though?), and if that's the case, I might borrow some of their books (Knuth's *The Art of Programming* comes to mind...), so that I can bring some other ones, instead...

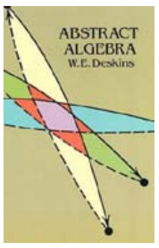
The choices will be based partly on 'This is an important book for me', and 'This is a book/series I've never got around to reading (fully)' (because it's so large, or requires in-depth study and contemplation).

Assuming I've got a (solar powered?) laptop with me, and/or have a remote hope of getting back to 'civilisation' some time, some of the books may then be computer-related.

Such lists also tend to reflect one's current interests, as when thinking of this, I chose books that I'd like to bring, as if I was going to be stranded on that island tomorrow...



The first book I'd like to bring is *Elements of Programming* by Alexander Stepanov and Paul McJones. I sense this is an important book (and a study group for it has recently been started at the 'C++Next' site [1]). It's not an easy book; it requires study and contemplation, but such is the case for most of the valuable things in life. This is one of the few books that contribute to putting our field on a firm mathematical/scientific foundation, so that software development one day may arise as a true engineering discipline...



To really get the full advantage of that book, I think I'll be needing another book that I've recently acquired: *Abstract Algebra* by W. E. Deskins. This appears to be an accessible, yet comprehensible introduction to the interesting field of abstract algebra, a field that until recently I didn't know had any connection with generic programming, but where it seems that this may be connected to taking generic programming to the next level.

That's two books I haven't yet read, but where I've started reading both, and so far, they are very promising. Definitely desert-island material...

Let's then move to a book I consider a classic, one which I have read, which is *Multi-Paradigm Design for C++* by Jim Coplien. This is one of the first and few books that address the concept of multi-paradigm software development, in a systematic way. Moreover, it applies this to the C++ language. Having read this book, you'll hardly look at the language the same way, again. It's been many years since I've read this one, and I'd really appreciate getting into it, again, so that one's in (despite the rather tough limit of four non-novel books).

Some runners-up include *Generative Programming* by Krzysztof Czarnecki and Ulrich W. Eisenecker. This is a rather 'heavy' book, and it addresses quite a bit of the same issues as Coplien's book, and, unlike Coplien's book, having read it once, I don't really feel like reading it again (at least not this time).

Another great book is *The Cluetrain Manifesto* by Rick Levine, Christopher Locke, Doc Searls and David Weinberger. This is one of the best books I've read, which tells the important story of what is really happening in our networked world, devoid of any hype. If it hadn't been for that I've read it twice, so I know it almost by heart (and due to the limit of five books) it would have been included. The authors are not 'hippies' removed from reality: These are hard-core business people having had positions like VP of marketing at major companies. One of my favourite quotes is: 'We're immune to advertising. Just forget it.'

These people tell it like it is, and reading the book gave me a similar experience as when I read Kent Beck's *Extreme Programming Explained* in 2001: Finally, someone who tells it like it is...

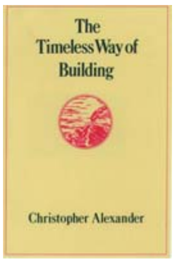
For a similar reason (having read a lot of 'agile' and 'lean' books since then), agile/lean books are not on my list for the desert island.

There's still room for one more technical book. *The Science of Programming* by David Gries (thanks to Hubert Matthews for recommending this book, many years ago) is a clear candidate. I sense this is also a great book, but it requires quite a bit of in-depth study and contemplation, so I've only started reading it. A desert island would be a perfect place to go deeply into this one. That the foreword is written by Edsger W. Dijkstra is quite telling...

This is a tough call, as another book I'd like to (re)read is: *Domain-Driven Design* by Eric Evans, which is also one of the best books I've read. However, given the tough constraints on technical books for the desert island, and that I've already read *Domain-Driven Design*, Gries is in, and Evans is out.

Now for the novel... After all this hard-core technical stuff, it would be nice with a novel or two to balance things...

The More Than Complete Hitchhiker's Guide to the Galaxy would be an obvious candidate for me (and it cleverly includes four of the five books in the Hitchhiker 'trilogy', so it could count as one book in this context...). However, given that I've already read it twice, it'll have to face some competition...



It may not count as a novel, but it's not a technical book, either, and that is *The Timeless Way of Building* by Christopher Alexander (by many regarded as the father of the patterns movement, and the book that really started it all), which is, again, one of the best books I've read (these books are great in different ways...), and it would certainly be a pleasure to read it again...

Moreover, a while ago, I bought Christopher Alexander's *Magnum Opus*: *The Nature of Order* (four books), which I haven't even started reading, and which would have been perfect for a desert island stranding... Yet, my non-novel book quota is already full (and this is not even one book), so I'd reluctantly leave these behind...

Regarding architecture and buildings, *How Buildings Learn* by Steward Brand is another book I've appreciated, but it's rather more limited in

Bookcase

The latest roundup of book reviews.

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

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

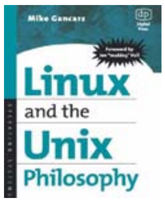
I must thank Blackwells and Computer Bookshop for their continued support in providing us with books.

Jez Higgins (jez@jezuk.co.uk)

Linux and the Unix Philosophy

By Mike Gancarz, published by Digital Press, ISBN: 1-55558-273-7

Reviewed by Ian Bruntlett



Highly recommended

This book should have been sub-titled 'Anagathics for programs and data' (anagathics are medicines which inhibit aging).

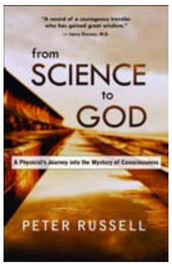
I find that when I disagree with parts of this book that either the disagreement is caused by 1) misunderstanding some of the text or 2) where I would opt for optimisation, the book would opt

Desert Island Books (continued)

scope than Alexander's book (*The Timeless Way of Building*), so it's not considered further here.

One of my favourite books of all time is Richard Bach's: *Jonathan Livingston Seagull*, which definitely is a good candidate for a novel to bring... However, it's quite short (it can be read in one sitting, more or less), which is a great thing, but perhaps not so great for a potentially very long stay at a desert island...

For the novel, or non-technical book, I'd like to include something that goes beyond 'mere' technical stuff, something that talks about life, itself, about personal development... Something that I could ponder on those long nights at the desert island...



For this, I'd like to include some spiritual/philosophical stuff from either Richard Bach or Peter Russell (and I'd hope that Paul would be lenient in letting it pass as a 'novel'...). I've recently bought several books by Peter Russell, and his background with a Ph.D. in theoretical physics, as well as his 'spiritual journey', gives him a quite interesting background (something missing in most 'pseudo-scientific' books in this genre). He's written several interesting-looking books, including *Waking Up in Time: Finding Inner Peace in Times of Accelerating Change* and *From Science to God*, and based on what I've read of him before, such as *Passing Thoughts*, a booklet he published many years ago (now partially available on the Internet), these are probably both good.

What's it all about?

Desert Island Disks is one of Radio 4's most popular and enduring programmes. The format is simple: each week a guest is invited to choose the eight records they would take with them to a desert island (http://www.bbc.co.uk/radio4/factual/desertislanddiscs.shtml).

The format of 'Desert Island Books' is *slightly* different from the Radio 4 show. You choose about five books, one of which must be a novel, and up to two albums. Some people even throw in the odd film. Quite a few ACCUers have chosen their Desert Island Books to date and there are plenty more to go.

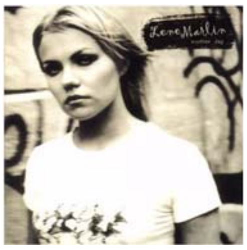
The rules aren't too strict but the programming books must have made a big impact on your programming life or be ones that you would take to a desert island. The inclusion of a novel and a couple of albums helps us to learn a little more about you. The ACCU has some amazing personalities and Desert Island Books has proved we only scratch the surface most of the time.

Each issue of CVu will have someone different. If you would like to share your Desert Island Books please email me: paul.grenyer@gmail.com.

Given that I only can bring one more book (although it's not really a novel), I'd like to choose *From Science to God*.

If I can't bring that one, it would be some other 'spiritual' book, such as *Bhagavad Gita*, *The Prophet* or *Jonathan Livingston Seagull*. It would be a difficult choice...

Two albums... This is even tougher than the choice of books... To be honest, I don't have two (or even one) album that stands out to such a degree that they are obvious candidates to bring. It's more the case of a series of albums that I'd like to bring.



My taste in music is perhaps more mainstream than my choice of novels... I guess it can be described as pop/rock/ambient. Artists I appreciate include Celine Dion, Sandra (Cretu), Enigma, Deep Forest, etc. Once I start thinking about my CDs, the choice becomes difficult... For those long days at the island, I'd like to bring something that speaks to my heart... Savage Garden's *Affirmations* is surprisingly good, and can move you to tears... Then there are instrumental masterpieces (IMO) like Delerium's *Karma*... However, given that I may be alone on the island, I'd like to bring something with a human voice...

Perhaps surprisingly, I've come to that I'd like to bring Lene Marlin's *Playing My Game*, and perhaps even more surprising is that I'd like to bring her next album, *Another Day* as my second choice... Lene Marlin has a delightful, intimate voice, which goes straight to your heart... I'd have liked to have brought all three of her first albums, but I know Paul won't allow it...

That's it: My five books and two albums. Now it's just a matter of preparing to get stranded...

P.S. This time, the prolonged 'dispute' about what's the best Pink Floyd album was not continued...!

A few books on complexity, self-organisation and emergence (which may well turn out to be some of the most important sciences of the 21st century...) would also be very desirable, but my quota is full, and I'd already have plenty to study and contemplate...

References

[1] http://www.cpp-next.com/

Next issue: Pete Goodliffe

for flexibility or longevity. To quote the book 'That which is cheap and effective is more useful that that which is big and expensive'. This book states that you should always store data in text files and that programs should act as filters with well defined interfaces.

There are a lot of pre-written Linux/Unix commands that are filters (take an input stream – `stdin`, process/filter text and send it to an output stream – `stdout`). Lots of applications can be written as shell scripts processing data that is text based far cheaper than a conventional monolithic program. The pre-existing commands mean that becoming a proficient programmer is tougher on these platforms than other platforms. However, the effort is worth it. According to this book, the 'Unix' way of things consists of up to three layers: 1) The UI layer which in turn sits on top of 2) The application layer which sits on top of 3) The small programs layer which implements the guts of the system. This means as a user you can opt to use a GUI or the command line/schell script to automate your intentions. This also means as a programmer you can change any of the three layers without having to change the other two layers for example if your GUI falls out of favour, you can replace the UI layer with new code.

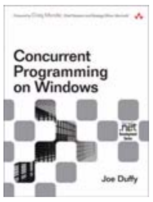
The Three Systems rule is covered here. The First System gets built and either dies or it gets developed into the Second System with the knowledge that it will get replaced by the Third (final) System.

As a semi-final recommendation, here are the key tenets stated by this book:

- Small is beautiful
- Make each program do one thing well
- Build a prototype as soon as possible
- Choose portability over efficiency
- Store data in flat text files
- Use software leverage to your advantage
- Use shell scripts to increase leverage and portability
- Avoid captive user interfaces
- Make every program a filter

Finally – this book is an engaging read that should be read by all Linux/Unix software developers.

Concurrent Programming on Windows



By Joe Duffy, published by Addison Wesley, ISBN: 978-0321434821

Reviewed by Anthony Williams

This is one of the thicker books on my bookshelf, with a whopping 958 pages, and with good reason – it is a comprehensive guide to both the Win32 and .NET multithreading APIs on Windows.

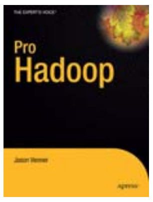
As well as explaining in detail how to use the various functions and classes, Joe goes into some detail explining the basics of concurrency and why things are done a particular way. This covers everything from the basics of the x86 memory model and how that affects the visibility and synchronization of data between processors, to the thread scheduling model for Windows.

There is in-depth coverage of all the synchronization primitives, as well as the Windows thread pool support, and asynchronous I/O. The comprehensive coverage of the .NET Parallel Extensions has been relegated to an appendix, as the library was only available as a preview at time of writing, but there's more than enough material in the main part of the book on the main .NET library.

Joe hasn't just covered the APIs either – there's a good discussion of various programming models for concurrency, such as data parallelism and task parallelism, a section on lock free programming, as well as sample implementations of data structures that are safe for concurrent access from multiple threads

This is an indispensible book for anyone wishing to use the native APIs when writing multithreaded programming on Windows.

Pro Hadoop



By Jason Venner, published by Apress, ISBN 978-1-4302-1942-2

Reviewed by Alan Lenton

For a start, I have to say, this book didn't exactly inspire me to rush out and play with Hadoop. Then there is the real problem. For all that it came out in 2009, the book is based on Cloudera's 0.18.3 version of Hadoop, while the current version is 0.20.1. As I understand it there are radical changes to the API in the 0.20.x versions. And the final killer, for me, at least, was the fact that a substantial number of pages throughout the book are devoted to regurgitating code, tables of machine settings, copies of initialisations files, and masses of extracts from output and log files. It even has pictures of test output in Eclipse!

I was left with the feeling that the author had a much slimmer volume in mind, but was forced to bulk it out to a set number of pages with a vast amount of mono-spaced font material. There may well be something more inspiring struggling to get out.

In all honesty, I can't recommend this book as it stands. I would suggest that unless you have a burning need to understand Hadoop instantly, you wait for the second editions of the current Hadoop books to come out. If you do need instant material, then you need to find a real bookshop (i.e., not Amazon), browse through and compare the three books that seem to be available on Hadoop, and pick the one that most suits you.

Not 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

- **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

In an indicative example of my occasionally chaotic approach to planning and deadlines, I've been mentally preparing to write my last View From The Chair. A last minute review of the copy dates shows that unless the new Chair is escorted from the AGM on the 17 of April and has their feet held to the fire until they've written something, I've got one more to go.

I can't really complain, though, because it's one of the few obligations of being Chair. Many aspects of the organisation run themselves. Take accu-general, for instance, which is almost entirely constructive and polite, if occasionally given to a pun too far. There is work involved in running ACCU though, and the bulk of it is done by Alan, Stewart, Mick, and Tony, respectively our Secretary, Treasurer, Membership Secretary, and Web Editor, and by the journal editors, currently Steve and Ric. I'm hugely grateful for their time and their quiet efficiency over the past several years.

ACCU's big event, the Conference, doesn't appear out of nowhere, of course. When I took over from Ewan as ACCU Chair, we formalised the position of Conference Chair. Ewan, and now Giovanni, together with their Conference committees have worked long and hard to continue the tradition of formidably good programmes. This year's programme is, once again, ridiculously good, including keynotes by Jeff Sutherland, James Bach, Dan North, and Russel Winder. I'm also excited that Robert Martin is coming back – Uncle Bob gave me a hug last year and it was ***great***.

The AGM is, of course, held during the Conference. My understanding is that the conference was originally created to give people something else to do while attending the AGM. As a means of bringing a large chunk of the membership together in one place, you have to say it's proved wildly succesful. You don't have to attend the Conference to attend the AGM, though, so do please come along if you can. While the administrative functions of the organisation are important, getting together with your fellow members, putting faces to names, making and renewing friendships, is even more so.

The 22nd AGM

Notice is hereby given that the 22nd Annual General Meeting of The C Users' Group (UK) publicly known as ACCU will be held at 13:00 on Saturday 17th April 2010 at the Oxford Barceló Hotel, (formerly the Oxford Paramount Hotel), Godstow Road, Oxford OX2 8AL, United Kingdom.

Current Agenda

1. Apologies for absence
2. Minutes of the 21st Annual General Meeting
3. Annual reports of the officers
4. Accounts for the year ending 31st December 2009
5. Election of Auditor
6. Election of Officers and Committee
7. Other motions for which notice has been given.
8. Any other Annual General Meeting Business (to be notified to the Chair prior to the commencement of the Meeting).

The attention of attendees under a Corporate Membership is drawn to Rule 7.8 of the Constitution:

> '... Voting by Corporate bodies is limited to a maximum of four individuals from that body. The identities of Corporate voting and non-voting individuals must be made known to the Chair before commencing the business of the Meeting. All individuals present under a Corporate Membership have speaking rights.'

Also, all members should note rules 7.5:

> 'Notices of Motion, duly proposed and seconded, must be lodged with the Secretary at least 14 days prior to the General Meeting.'

and 7.6:

> 'Nominations for Officers and Committee members, duly proposed, seconded and accepted, shall be lodged with the Secretary at least 14 days prior to the General Meeting.'

and 7.7:

> 'In addition to written nominations for a position, nominations may be taken from the floor at the General Meeting. In the event of there being more nominations than there are positions to fill, candidates shall be elected by simple majority of those Members present and voting. The presiding Member shall have a casting vote.'

For historical and logistical reasons, the date and venue is that of the last day of the ACCU Spring Conference. Please note that you do not need to be attending the conference to attend the AGM.

(For more information about the conference, please see the web page at http://accu.org/conference.)

More details, including any more motions, will be announced later. A full list of motions and electoral candidates will be supplied at the meeting itself.

Alan Bellingham
Secretary, ACCU