The magazine of the ACCU

www.accu.org

{cvu}

Volume 20 Issue 1 February 2008 £3

Features

Regulars

Ivan Uemlianin
A Simple Calculator in Tkinter and wxPython

{cvu}
Volume 20 Issue 1
February 2008
ISSN 1354-3164
www.accu.org

Why bother?

After typing 20.1 a number of times in the many emails that fly around the review team and between myself, Pete and Alison (the production editor), the implication has only really just dawned on me. Twenty. Twenty years! That is quite some time. I just hope that some original ACCUer doesn't come back and point out some numbering irregularity at the dawn of time and spoil my moment.

Why do we do this? Why does ACCU exist? Why do all the contributors to C Vu and Overload do it? What about all the volunteers on the committees, the article review teams for C Vu and Overload. Why get together at the conference and at local ACCU meetings? Sure, alcohol is one reason, but surely it isn't the only one. You can go and get a drink anywhere.

I think a huge reason is 'the people'. ACCU members like being around other ACCU members. Like minded people. After being around other developers at work who never seem to really care, it is good to get into more interesting, occasionally lively, frequently highly technical, conversations (or email exchanges) with people who think more like you do. Design matters. Testing matters. And more importantly, elegance matters. Software development can be an art, but many of those who call themselves software developers have the artistic talent of a four-year-old who is inadvertently making a collage with poster paint, cookie crumbs and their own hair.

You care about developing yourself, and you care about not stagnating. It's been said that if you are not moving forward, you're moving backwards, there is no standing still. If you are not learning anything new at work, best go and look for it outside the workplace. This is why many contributors write articles. You learn a huge amount when forced to write stuff down. As you organise your thoughts into something that others can read, you find gaps in your own knowledge. You go and find out, then write that down too. Writing is an amazing way to learn.

Do you care enough to write? C Vu and Overload could both be bigger, and have more diverse content, if more people decided to put their thoughts down and learn something in the process. I dare you. I dare you to try to write an article and not learn something new. Do it!

TIM PENHEY,
EDITOR

The official magazine of ACCU

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

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

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

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

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

DIALOGUE

FEATURES

COPY DATES

C Vu 20.2: 1st March 2008
C Vu 20.3: 1st May 2008

IN OVERLOAD

'Watersheds and Waterfalls' by Stuart Golodetz, more from the 'PfA Papers' by Kevlin Henney and part 2 of 'The Regular Travelling Salesman' by Richard Harris, plus much more.

ADVERTISE WITH US

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

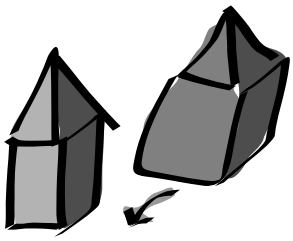
To advertise in the pages of C Vu or Overload, contact the advertising officer at ads@accu.org.

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

COPYRIGHTS AND TRADE MARKS

An Introduction to Town Planning

Pete Goodliffe exposes war wounds and contrasts software designs.



Build up, build up, prepare the road!
Remove the obstacles out of the way of my people.
Isaiah 57:14

Experience is a great teacher. Other people's experience is even better – if you can learn from other people's mistakes then you will save yourself a lot of pain. In this column we'll look at the consequences of good and bad design decisions (or the lack of them). We'll see how important it is to consider software design up-front.

We'll do this by contrasting two real world systems written by different teams of programmers in different companies. These are two comparable, reasonably large systems. Each software ecosystem was mature and had gone through many product releases. Both were for consumer audio products, and whilst 'embedded' in nature, both involved fairly large Linux-based C++ software applications. They turned out very differently.

Exhibit A: The Messy Metropolis

The first software system we'll look at is known as The Messy Metropolis. I've mentioned this system in a previous professionalism in programming column [1]. It's one I look back on fondly – not because it was good, or because it was enjoyable to work with, but because it taught me a valuable lesson about software development when I first came across it. Our tentative delve into the guts of this software 'design' should be regarded as a cautionary tale. I'm exposing one of my war wounds.

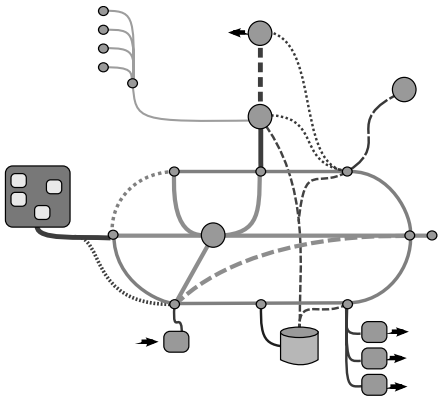
My first contact with the Messy Metropolis was when I joined the company that created it. It initially looked like a promising job; I was to join a team working on a Linux-based 'modern' C++ codebase that had been in development for a number of years. Exciting stuff, if you have the same peculiar fetishes as me.

The work wasn't plain sailing at first, but you never expect an easy ride when you start to work in a new team on a new codebase. But it didn't get any better as the days (and weeks) rolled by. The code took a fantastically long time to learn; there were no obvious routes into the system. That was a warning sign. At the micro-level, looking at individual lines, methods, and components, the code was messy, and badly put together. There was no consistency, no style, and no unifying concepts drawing the separate parts together. That was a warning sign. Control flew around the system in unfathomable and unpredictable ways. That was a warning sign. There were so many bad 'code smells' [2] that the codebase was not just putrid, it was a pungent land-fill site on a hot summer's day. A clear warning sign. The data was rarely kept near where it was used. Often extra baroque caching layers were introduced to try to persuade it to hang around in more convenient places. That was a warning sign.

As I tried to build a mental picture of the Metropolis, no one was able to explain the structure; no one knew all of its layers, tendrils, and dark secluded corners. In fact, no one actually knew how any of it really worked (it was actually by a combination of luck and heroic maintenance programmers). People knew the small areas they had worked on, but no one had an overall comprehension of the system. And, naturally, there was no documentation. That was a warning sign. What I needed was a map.

This was the sad story I had become a part of: the Metropolis was a town planning disaster. Before you can improve a mess you need to understand that mess, so with much effort and perseverance we pulled together a map of the 'architecture'. We charted every highway, all the arterial roads, the uncharted back roads, and all of the dimly lit side passages, and placed them on one master diagram. For the first time we could see what the software looked like. Not a pretty sight. It was a tangle of blobs and lines. In an effort to make it more comprehensible we colour coded the control paths to signify their type. Then we stood back.

It was stunning. It was psychedelic. It was as if a drunk spider had stumbled into a few pots of poster paint and then spun a chromatic web across a piece of paper. And then it became clear. We had all but drawn a map of the London Underground. It even had the circle line. It looked something like this (it's a simplified version, with details changed to protect the guilty):



That's not a 'good' architecture by any metric. The Metropolis' problems actually went beyond the architectural level, right up to the development process and company culture. This fact actually explained a lot of the architectural problems. The code had grown 'organically' over a period of years. This is a polite way to say that no one had performed any architectural design of note and that various bits had been bolted on over time without much thought. No one had ever stopped to impose a sane structure on the code. This was an example of a system with no architectural design. But a codebase never has no architecture. It just has a very poor one.

The Metropolis' state of affairs was understandable (but not condonable) when you looked at the history of the company: it was a start-up with heavy pressure to get many new releases out rapidly. Delays were not tolerable – they would spell financial ruin. The software engineers were driven to get code shipping as quickly as humanly possible (if not sooner). And so the code had been thrown together in a series of mad dashes.

The lack of clear design had these consequences:

- As you can already see, the Metropolis' architecture had lead to a system that was remarkably tricky to comprehend and practically impossible to modify. The bad design encouraged further bad design to be bolted onto it – in fact it literally forced you to do so – as there was no way to extend the design in a sane way. The path of least resistance for the job in hand was always taken.
- New recruits coming into the project (like myself) were stunned by the complexity and unable to get to grips with what was going on. This partially explains why very few new recruits stayed at the company for any length of time – staff turnover was very high.

PETE GOODLIFFE

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

- The system's components were not at all cohesive. Where each one should have had a single, well-defined role, they each contained a grab-bag of functionality that wasn't necessarily related. This made it hard to determine why a component existed at all, and hard to work out where a particular piece of functionality had been implemented.
- Both functionality and data were located in the wrong place in the system. Many things you'd consider 'core services' were not implemented in the hub of the system but were simulated by the outlying modules (at great pain and expense). Further software archaeology showed why: there had been personality struggles in the original team and so a few key programmers had begun to build their own little software empires. They'd grab the functionality they thought was cool and plonk it into their module, even if it didn't belong there. To deal with this they would then make ever-more baroque communication mechanisms to stitch the control back to the correct place.

Bad architecture can have a really profound effect and severe repercussions

- There was no clear layering. The dependencies between modules were not unidirectional; coupling was often bidirectional. Component A would hackily reach into the innards of component B to get its work done for one task. Elsewhere component B had hard-coded calls onto component A. There was no bottom layer, or central hub to the system. It was one monolithic blob of software.
- This meant that the individual parts of the system were so tightly coupled that you couldn't bring up a skeletal system without creating every single component. This made low-level testing impossible. Not only were code-level unit tests impossible to write but component-level tests could not be constructed as every component depended on almost every other component. Of course, testing had never been a particularly high priority in the company (we don't have anywhere near enough time to do that), so this 'wasn't a problem'. Needless to say, the software was not very reliable.
- The problems with bad top-level design had worked their way down to the code level. Problems beget problems (see the discussion of broken windows in [3]). Since there was no common design and no overall project 'style', no one bothered with common coding standards, the use of common libraries, or employing common idioms. There were no naming conventions for components, classes or files. There was even no common build system: duct tape, shell scripts, and perl glue nestled alongside makefiles and Visual Studio project files. Compiling this monster was considered a rite or passage!
- More software archaeology showed why: the Metropolis started out as a series of separate prototypes that got tacked together when they should have been thrown away. The Metropolis was actually an accidental conurbation.
- The problems weren't contained in the codebase. We've seen that they had spilled out into the development team, but they also affected the people supporting and using the product:
  - Support engineers had an awful time working out the intricate behavioural differences between relatively minor software releases.
  - An external control protocol was developed for other devices to control the Metropolis remotely. Since it was a thin veneer over the guts of the software it reflected the Metropolis' architecture, which means that it was baroque, hard to understand, prone to fail randomly and impossible to use.
  - The design was almost completely irredeemable. The amount of effort required to re-work, refactor, and correct the problems with the code structure had become prohibitive. A rewrite wasn't a cheap option, as support for the old control protocol was a requirement.

As you can see, the inevitable consequence of the Metropolis' 'design' was a diabolical situation that could only get worse. It was so hard to add new features to that people were just applying more bodges, sticking plasters and calculated fudges. No one was enjoying working with the code any more, and it was heading in a downward spiral. The lack of design had led to bad code, which led to bad team morale and increasingly lengthy development cycles, which eventually led to severe financial problems for the company.

Eventually the management acknowledged that the Messy Metropolis had become uneconomical to maintain and it was thrown away. This is a brave step for any organisation, especially one that is constantly running ten paces ahead of itself whilst trying to tread water. With all the C++ and Linux experience the team had gained form the previous version, the system was rewritten in C# on Windows.

The upshot

Bad architecture can have a really profound effect and severe repercussions. The lack of foresight and architectural design in the Messy Metropolis lead to:

- a low-quality product with infrequent releases
- an inflexible system which couldn't accommodate change or the addition of new functionality
- pervasive code problems
- staffing problems
- a lot of messy internal politics
- lack of success for the company
- many painful headaches.

Next time

So after that inspiring look at what happens when software takes a turn for the worse, we'll conclude this series with a happy contrast – the town planner's triumph. We'll look at a more successful system and see the consequences of a better software architecture. I bet you can't wait. ■

Endnotes

[1] Pete Goodliffe. Professionalism in Programming #21: Software Architecture. In: C Vu Volume 15, Issue 4.
[2] Martin Fowler. Refactoring: Improving the Design of Existing Code.
[3] Andrew Hunt and David Thomas. The Pragmatic Programmer. Chapter 1, Section 2.

Operator Names Influence Operator Precedence Decisions (Part 1 of 2)

Derek Jones plunges into the depths of operators.

Introduction

The 2006 ACCU experiment [1] threw up a surprising result; in 33% of cases experienced developers failed to correctly parenthesis an expression containing two binary operators to reflect the binding of operands to operator (i.e., the relative operator precedence). The result was surprising because if carried over into actual software development it would result in almost 1% of all expressions contained in source code [2] being wrong (i.e., they would evaluate to a value different from that intended by the author, for instance the author or a subsequent reader of the expression `x & y == z` might incorrectly expect it to behave as if it had been written in the form `( x & y ) == z`.

Four possibilities spring to mind:

1. Subject's performance on the problem presented in the experiment is not a good approximation of their performance when writing code in real life.
2. When writing complicated expressions developers do make many precedence mistakes, but most of these are fixed before software goes into production use.
3. The source code measurements of parenthesis usage made in connection with the 2006 ACCU experiment (which show that developers often do not use parenthesis to specify the intended binding of operands to operators; see Table 3 & 4 of the write-up [1]) are not representative of other source code. The claim your author often hears in face-to-face discussion with developers is that they always make use of parenthesis to specify the intended precedence in complicated expressions.
4. When reading complicated expressions developers make use of additional information that is available within the context in which the expression occurs (e.g., what the expression is calculating, the names of the variables appearing in the expression, or the amount of spacing between operators and operands [3]).

The 2007 ACCU experiment was designed, among other things, to investigate one case involving the fourth of these possibilities.

This is the first of a two part article that reports on an experiment carried out during the 2007 ACCU conference investigating both impact of operand names on binary operator precedence selection and the consequences of a limited capacity short term memory on subjects performance in recalling information about assignment statements. This first article provides general background on the study and discusses the results of the relative precedence selection problem, while part two discusses the assignment problem.

The hypothesis

Developers are often encouraged to use meaningful names for identifiers. The expectation is that meaningful names will provide information to subsequent readers of the code that will reduce the effort needed to comprehend that code. For instance, the name `flags` suggests an entity that denotes one or more on/off states.

The meaning associated with a name might also lead to an expectation about the kinds of operators used to manipulate variables having that name. For instance, an identifier having the name `flags` might be expected to appear much more often as the operand of a bitwise or logical operator than an arithmetic operator.

A name might have meaning to a developer because of its English language usage or because of being frequently encountered in a given context within source code. It is assumed here that presented with a problem involving source code, subjects will primarily make use of their experience of name usage in source code rather than any English language usage (i.e., occurrences in source code is used as a measure of context rather than occurrences in English prose).

Binary operators found in many programming languages might be divided into three broad categories.

1. Arithmetic operators. The operands of these operators can take on a range numeric values. Variables appearing in arithmetic operands might also appear as operands of relational or equality operators.
2. Bitwise operators. The operands of these operators are treated as a sequence of bits. A common mapping is for a bit to denote some quantity having one of two values and a sequence of bits to denote a set of such distinct quantities. These operands might also appear in equality tests, but their semantics is such that they are unlikely to appear in relational tests.
3. Logical operators. The operands of these operators essentially have two possible values, zero and non-zero. The semantics of these operators makes it possible that they may appear in equality against a literal zero, but are unlikely to appear in an equality test against a variable operand (testing two variables for non-zero cannot be simply mapped to a single equality test). They are unlikely to occur in relational tests.

The hypothesis being tested by the parenthesis problem is that the names of identifiers, appearing as operands in an expression, can have a significant impact on subject's selection of operator precedence to be applied within that expression. For instance, when attempting to comprehend an expression such as `a + b & c` readers have to decide whether `b` is added to `a` or bitwise ANDed with `c`. This experiment manipulates the name of the operand that has two operators with which it could bind. Possible naming impacts include:

1. The name does not appear to provide any information that readers might use to help make a precedence decision.
2. The name is chosen such that readers might make use of their experience of identifier naming conventions, along with knowledge of English language usage, when making an operator precedence decision. This naming information can be experimentally manipulated to:
   - increase the probability that the correct precedence decision is made,
   - decrease the probability that the correct precedence decision is made (e.g., if `b` is replaced by `flags` then a reader may be influenced to believe (incorrectly) that `flags` is ANDed with `c` and the result added to `a`.

DEREK JONES

Derek used to write compilers that translated what people wrote. These days he analyses code to try and work out what they intended to write. Derek can be contacted at derek@knosof.co.uk

What meaning context might a name have?

A name can acquire meaningfulness to a person through repeated encounters in a given context. There are a number of situations in everyday life that require the use of arithmetic and binary concepts (e.g., *switch light on/off*, *change mind* and *flag an error*, and measurements confirm that words and phrases commonly used in human conversation are carried over into identifier names.

To maximise the effect that operand names have on subject's decision making the names need to be highly representative of the kinds of names that appear within source code in the various operator contexts. The selection was based on your authors intuition (i.e., they were made before any source code measurements were available). See Table 7 for information on the names chosen and their frequency of occurrence in the measured source code.

Table 1

Names and literals that your author believed, when creating the text of the problems, to most likely to occur as the operands of bitwise, logical and arithmetic operands. The last column contains names that are believed not to provide any information that can be used to help decide which operator context they might belong.

bitwise operands	logical operands	arithmetic operands	anonymous operands
error	mask	count	resource
flags	finished	num	val
state	flag	offset	field
bits	done	rate	n
options	started	length	temp
0xff	TRUE	2	7

There are situations where an identifier name belonging to one category might appear as the operand of an operator in another category. For instance, there are algorithms that use bitwise operators to more efficiently perform an arithmetic operation (e.g., shift left to multiply by a power of 2), and it is possible that an arithmetic variable appears as the operand of a logical operator because the author omitted an explicit equality test against zero (because the language specifies that an implicit test is performed).

It is straightforward to assign a context to many of the binary operators (see Table 2). The only context that makes sense for the relational operators is an arithmetic one (source code measurements showed the same names appearing frequently as the operands of both kinds of operator). While there might be situations where meaning can be assigned to a relational comparison of bitwise or logical operands, these are likely to be rare. The equality operators might be applied to operands having any context and are classified as anonymous operators, which means that they do not provide any information that might be used to decide whether a particular operand binds to them or another operator.

Table 2

Binary operators and their associated context. The left operand of a shift operator has a bitwise context, while the right operand has an arithmetic context and the result has a bitwise context.

bitwise operator context	logical operator context	arithmetic operator context	anonymous operator context
`& | ^`	`&& ||`	`x * %`	`== !=`
		`+ -`	
		`< > >= <=`	
`<< >>`		`<< >>`	

Related work

Most of the operator/operand categorization techniques place restrictions on what are considered to be legitimate combination of pairs of operands, or operator/operand combinations. Possibilities include:

- most languages do not support the unrestricted mixing of operands of different types in an expression. For instance, arithmetic operations cannot be performed between an integer and string (some languages attempt to support this by performing implicit conversions). There are languages that support sophisticated typing mechanisms which allow developers to create new types which cannot be mixed with operands of a different type.
- in scientific calculations, dimensional analysis restricts the mathematical operations that can be performed on values denoting some physical quantity (because no physical meaning can be given to the operation). For instance, adding a distance to a time value has no meaning while dividing distance by time yields the velocity. Tools that check scientific software for consistent use of physical dimensions are available [4].

The names of the operands are not part of these categorization techniques. Information on these restrictions is explicit specified as part of a language or application specification and not something that a developer might subconsciously pick up over time.

In a study by Bassok, Chase and Martin [5] subjects were asked to create mathematical problems involving named everyday objects. The results showed that subjects often made use of semantic relationships between the named objects. For instance, given apples and baskets the problems generated by subjects might involve dividing the number of apples by baskets, but rarely involved adding them.

Measuring operand name usage

This subsection discusses measurement of names appearing as or in the operands of some binary operators within in the visible source of a number of large C programs (e.g., gcc, idsoftware, linux, netscape, openafs, openMotif, and postgresql). The visible, rather than the preprocessed, source was used as the basis for measurement because we are interested in what a reader of the source sees and has to make decisions about, and not what the compiler has to analyse.

The contents of preprocessor directives were not included in the measurements.

The source code constructs included in these measurements were all the ones in which the C grammar permitted an expression (a *full expression* to use the C Standard's terminology) to occur (excluding preprocessor directives).

When the operand was a function call the identifier denoting the function was used, when the operand was an array access the identifier denoting the array was used, when the operand was a structure or union member selection the identifier denoting the member was used (e.g., in `p.q` the identifier of interest was taken to be `q`).

Operands whose value was obtained by applying the indirection operator (i.e., unary `*`) were not counted.

A variety of conventions are used to create the character sequence in an identifier name. Developers extract information from identifiers by recognising character sequences that have meaning to them. These character sequences might be words, abbreviated words, acronyms or sequences of two or more of these.

There are three common conventions for explicitly highlighting the boundary between two character sequences:

1. Use of the underscore character (i.e., `_`). For instance, `mumble_flags`.
2. Writing the first character of a new sequence in upper case. For instance, `mumbleFlags`.
3. Relying on reader expertise to work out where the character sequence boundary exists. For instance, `mumbleflags`.

These measurements applied the first two conventions to extract names from identifiers. In the case of the second point above: any sequence of digit characters was treated as a break-point between two names; and a run of two or more upper case letters was treated as a single name (i.e., the last

upper case letter was not regarded as being the start of a name using the second convention given above).

An identifier name created by joining together two or more words might be viewed as part of a sentence or a compound word. For instance, `mumble_flags` might be interpreted as two or more flags associated with mumble (and probably occurring in a bitwise or logical context), while `num_flags` might be interpreted as an abbreviated form of the phrase *number of flags* (and probably occurring in an arithmetic context). These measurements do not attempt to deduce the interaction between any words appearing in an identifier considered to contain two or more words.

Table 3

The top 10 most common names appearing as or in identifiers as the operands of binary operators in the visible source of `.c` files. **Programs** the number of programs containing an instance of the identifier having or containing the name, **Occurrences** the number of occurrences of the name, **Arithmetic** the percentage of occurrences in an arithmetic context, **Bitwise** the percentage of occurrences in a bitwise context, **Logical** the percentage of occurrences in a logical context, **Equality** the percentage of occurrences as the operand of an equality operator.

Programs	Occurrences	Name	Arithmetic	Bitwise	Logical	Equality
7	35627	i	86.3	5.7	0.4	7.7
7	25201	null	0.1	0.1	0.1	99.7
7	14821	size	79.9	7.7	0.4	12.0
7	14121	type	8.3	8.4	2.4	81.0
7	12160	flags	0.5	97.4	0.2	1.9
7	12091	len	87.3	3.7	0.5	8.5
7	9396	count	78.2	4.7	1.2	15.9
7	9345	max	88.0	2.4	0.2	9.4
7	8668	code	7.0	3.8	0.6	88.6
7	8462	status	16.9	56.9	0.5	25.7

There were 136,127 identifiers that appeared as the operand of one of the binary operators of interest in this study, from which 30,683 unique names were extracted. Table 3 lists the 10 most commonly seen names, Table 4 the 10 most commonly seen names in a bitwise context and Table 5 the 10 most commonly seen names in a logical context (in this case the majority of *all* occurrences requirement was relaxed to the requirement that the percentage occurrence in a logical context be greater than any other context).

An application domain specific name might be frequently used in one program but not in any other, e.g., `inb` in Table 4. Unless a developer has experience working on that program they might not be aware that in some domains it has a common usage in a given context. The *Programs* column lists the number of different programs containing at least one instance of a name.

Table 4

The top 10 names appearing with greater that 50% frequency as or in identifiers as the operands of bitwise binary operators in the visible source of `.c` files.

Programs	Occurrences	Name	Arithmetic	Bitwise	Logical	Equality
7	12160	flags	0.5	97.4	0.2	1.9
7	8462	status	16.9	56.9	0.5	25.7
7	5854	mask	6.4	89.0	1.1	3.6
7	4815	read	20.6	50.8	1.5	27.0
7	4162	f	25.0	53.5	3.1	8.3
7	2827	val	28.7	50.8	0.8	19.6
7	2270	stat	14.5	71.0	0.8	13.7
7	2322	flag	3.8	70.6	6.4	19.3
1	1764	inb	2.2	83.7	0.1	14.1
7	1346	active	9.6	50.7	4.2	35.5

Table 5

The top 10 most common names, where the percentage occurrence in a logical context is greater than any other context, appearing as or in identifiers as the operands of logical binary operators in the visible source of `.c` files.

Programs	Occurrences	Name	Arithmetic	Bitwise	Logical	Equality
7	2631	is	2.4	18.4	55.5	23.7
6	963	user	20.6	9.6	38.8	31.0
4	329	operand	21.6	20.1	27.4	31.0
5	280	constant	15.0	15.7	26.4	42.9
3	251	reload	16.7	6.4	31.1	45.8
5	227	template	13.2	0.4	26.0	60.4
3	213	charset	10.3	4.2	36.2	49.3
3	191	put	22.5	11.5	59.7	6.3
7	182	after	9.3	13.7	26.9	50.0
3	165	pthread	5.5	0.0	12.7	81.8

Given the results of the 2006 experiment (i.e., 33% of answers given by subjects were incorrect) measurements based on the assumption that the author of the expression `a + b & c` expected `b` to be the operand of an arithmetic operation could be significantly inaccurate. The operand name measurements given here are restricted to those subexpressions where the intended binding is considered to be unambiguous. For instance, when brackets delimited a single binary operation (e.g., `(a + b) & c` or when an expression contained a single binary operator (the simple assignment or compound assignment operators were not treated as binary operators for the purposes of these measurements). This restriction did not have a significant impact on the amount of data obtained. As Table 6 shows, most expressions contain either zero or one of the binary operators of interest.

Table 6

Percentage of expressions containing a given number of binary operators in their visible source code. Note that function call, direct and indirect member selection, and assignment operators are not included here as binary operators (although the arguments of function calls and array subscript expressions are counted as separate expressions).

Binary operators	% occurrence	Binary operators	% occurrence
0	92.82	3	0.62
1	5.28	4	0.14
2	0.86	5	0.13

Occurrence of names used in the problems

Table 7 lists the number of occurrences of each name used in the parenthesis problems along with the percentage occurrence as the identifier (or part of an identifier) in the operand of arithmetic, bitwise, logical or equality operators. The measurements show that your author's intuition is not always a reliable indicator of what constitutes common usage of names in some contexts.

Table 7

Actual occurrences, in the visible `.c` files, of the names used (Table 1) in the parenthesis problem. The **Other operands** entry is included to provide supporting information on `TRUE` context usage. See Table 3 for a description of the columns.

Name	Programs	Occurrences	Arithmetic	Bitwise	Logical	Equality
arithmetic operands						
count	7	9396	78.2	4.7	1.2	15.9
num	7	7905	81.1	3.3	0.5	15.1
offset	7	5889	79.5	10.9	0.8	8.9
rate	5	528	62.9	17.6	0.4	19.1
length	7	5097	77.4	6.5	0.4	15.7
bitwise operands						
error	7	2960	12.1	21.8	1.4	64.7
flags	7	12160	0.5	97.4	0.2	1.9

Table 7 (cont'd)

Name	Programs	Occurrences	Arithmetic	Bitwise	Logical	Equality
state	7	5723	5.6	29.2	0.8	64.4
bits	7	2934	59.3	27.5	0.2	12.9
options	6	543	13.4	74.6	2.8	9.2
logical operands						
mask	7	5854	6.4	89.0	1.1	3.6
finished	6	50	46.0	8.0	0.0	46.0
flag	7	2322	3.8	70.6	6.4	19.3
done	7	578	14.5	29.9	4.5	51.0
started	4	91	16.5	61.5	2.2	19.8
TRUE	7	735	2.7	0.0	0.7	96.6
anonymous operands						
resource	3	481	33.9	15.8	0.0	50.3
val	7	2827	28.7	50.8	0.8	19.6
field	7	523	18.7	32.3	4.2	44.7
n	7	5141	78.8	6.4	0.6	14.1
temp	7	1588	30.5	45.0	1.0	23.5
Other operands						
FALSE	7	1046	0.0	0.1	0.4	99.5

- **Arithmetic operands**. All of the names used in the arithmetic context problems occurred much more frequently in this context than any other context.
- **Bitwise operands**. Two of the names used in the bitwise context problems occurred much more frequently in this context than any other context and two occurred just over twice as frequently in this context as an arithmetic context. One name `bits` appeared most often in an arithmetic context, although the singular form (i.e., `bit`) occurred more frequently in a bitwise context (see Table 8).
- **Logical operands**. All of the names used in the logical context problems occurred much more frequently in other contexts. The name `TRUE` was actually more common in an arithmetic context and less common in a bitwise context. Using this name as an operand was probably a mistake because in a real logical operand context within source it would effectively create dead code. However, it did occur in the problems seen by subjects and given that the corresponding name `FALSE` occurred most commonly in a logical operand context, this name maintained its logical context classification. The semantics of the logical operators (for the languages of interest here) specifies that each operand is implicitly compared against zero. Because of the implicit comparison there is a common developer practice of omitting an explicit comparison. For this common usage case the operands of the logical operators might be expected to have names that imply an arithmetic or bitwise usage. The source code measurements in Table 5 bear this out. Names appearing in the operand of a logical operator also frequently occur in operands of arithmetic and bitwise operators. As Table 2 in the 2006 article [1] shows, the logical operators occur sufficiently often in source code that if operator context specific names were commonly used they would show up in the source measurement counts.
- **Anonymous operands**. There is no consistent pattern of usage., with some occurring twice as frequently in one context as another. By far the most common occurrence of the name `n` was in an arithmetic context and its use was reclassified as such.

Some names appearing in the source might be considered to be variant spellings of a name appearing in a problem. Table 8 lists various character sequences found in the source which might be considered to denote the same name, along with number of occurrence information.

Table 8

Actual occurrences, in the visible of `.c` files, of the names used (Table 1) in the parenthesis problem. See Table 3 for a description of the columns.

Name	Programs	Occurrences	Variant	Arithmetic	Bitwise	Logical	Equality
error							
	7	2013	err	21.2	31.1	0.4	47.2
	3	175	errors	54.3	20.6	0.6	24.6
flag							
	2	1024	cflag	0.0	90.5	1.1	8.4
state							
	5	291	states	6.9	89.7	0.7	2.7
bits							
	7	2399	bit	35.0	48.2	4.5	12.3
	4	200	bitmap	42.5	15.0	1.0	41.5
options							
	6	453	option	16.6	48.3	3.5	31.6
finished							
	5	81	finish	27.2	1.2	3.7	67.9
started							
	7	4565	start	76.2	10.6	0.6	12.7
count							
	6	305	counter	79.0	7.2	0.0	13.8
num							
	7	1922	number	56.7	9.9	0.4	33.0
offset							
	7	1255	off	69.3	13.9	1.6	15.2
	2	137	offs	92.0	0.0	0.0	8.0
length							
	7	12091	len	87.3	3.7	0.5	8.5
val							
	7	4006	value	36.3	24.4	3.8	35.4
	7	492	retval	43.9	13.6	0.0	42.5
field							
	5	90	fields	55.6	0.0	3.3	41.1
temp							
	7	2223	tmp	49.8	27.8	0.8	21.7

Some of the possible variant spellings appearing in Table 8 are actually more common than the names actually used (e.g., compare `value` vs. `val` in Table 7). Any future experiments based on these problems might like to use the more common variant name.

When analysing the results the names were reclassified according to the findings of the source code measurements (see Table 9).

The experiment

The experiment was run by your author during a 45 minute lunchtime session of the 2007 ACCU conference (http://www.accu.org) held in Oxford, UK. Approximately 290 people attended the conference, 6 (2%) of whom took part in the experiment. Subjects were given a brief introduction to the experiment, during which they filled out brief background information about themselves, and they then spent 20 minutes working through the problems. All subjects volunteered their time and were anonymous.

The problem to be solved

The problem to be solved followed the same format as an experiment performed at a previous ACCU conference and the details can be found elsewhere [6].

The following is an excerpt of the text instructions given to subjects:

The task consists of remembering the value of three different variables and recalling these values later. The variables and their corresponding values appear on one side of the sheet of paper and your response needs to be given on the other side of the same sheet of paper.

1. Read the variables and the values assigned to them as you might when carefully reading lines of code in a function definition.
2. Turn the sheet of paper over. Please do **NOT** look at the assignment statements you have just read again, i.e., once a page has been turned it stays turned.
3. At the top of the page there is a list of five expressions. Each expression contains three different operands and two different operators. Insert parenthesis to denote how you think the operators will bind to the operands (i.e., you are inserting redundant parenthesis).
4. You are now asked to recall the value of the variables read on the previous page. There is an additional variable listed that did not appear in the original list.
   - if you remember the value of a variable write the value down next to the corresponding variable,
   - if you feel that, in a real life code comprehension situation, you would reread the original assignment, tick the "would refer back" column of the corresponding variable,
   - if you don't recall having seen the variable in the list appearing on the previous page, tick the "not seen" column of the corresponding variable.

Figure 1 is an example of one of the problems seen by subjects. One side of a sheet of paper contained three assignment statements while the second side of the same sheet contained the five expressions and a table to hold the recalled information. A series of X's were written on the second side to ensure that subjects could not see through to identifiers and values appearing on the other side of the sheet. Each subject received a stapled set of sheets containing the instructions and 40 problems (one per sheet of paper).

The parenthesis insertion task can be viewed as either a time filler for the assignment remember/recall problem, or as the main subject of the experiment. In the latter case the purpose of the assignment problem is to make it difficult for subjects to keep track of answers they had given to previous, related operator pair, problems.

Subject's performance on the parenthesis task is discussed in this article (the first of two).

The parenthesis problems

Based on the results of the 2006 ACCU experiment [1] it was anticipated that on average subjects would answer around 20 complete questions. This gave an estimated 100 answers per subject (subjects actually averaged 123.5 answers).

The selection of operands and operators followed the same procedure as the 2006 experiment, with the following exceptions (using the name contexts given in Table 1 and operator contexts given in Table 2):

- The name of the first operand was chosen from the set of names considered to have the same context as the first operand and similarly for the name of the third operand with respect to the second operator.
- The name of the second operand (i.e., the one that might bind to either operator) was selected (randomly with equal probability) from either the set of names having the same context as the left operand or the set of names having the same context as the right operand.

Figure 1

```
---------- first side of sheet starts here ----------
 propeller =  6;
 sofa =  5;
 chair =  4;
---------- second side of sheet starts here ----------
0xff | rate * num
field < rate ^ flags
error & 2 < count
done || finished | 0XFF
field != 7 ^ started
```

		remember	would refer back	not seen
chair	=	_____	_____	_____
table	=	_____	_____	_____
sofa	=	_____	_____	_____
propeller	=	_____	_____	_____

- The name of the middle operand was never the same as that of the first or third operand.
- There was a 10% probability that the letters of any operand were converted to upper case.

Results

Source code measurements showed that the names originally used in the creation of experimental problems (Table 1) did not always occur in the expected context. The findings of the source code measurements were used to reclassify some names as belonging to a different context, see Table 9.

Table 9

Names and literals with their associated context, recategorised based on measurements of C source code.

bitwise operands	logical operands	arithmetic operands	anonymous operands
error		count	resource
flags		num	val
state		offset	field
bits		rate	temp
options		length	7
0xff	TRUE	n	2
mask			
flag			
finished			
done			
started			

Subject experience

The average subject experience was 14.5 years (standard deviation 9.0). Because of the small number of subjects it is not possible to draw any statistically significant correlations between subject experience and other quantities.

Fortunately C, C++, C#, Java, Perl, and many other languages use the same relative precedence for those operators used in this experiment, so there is no need to worry about the possibility of subjects (attending the Association of C and C++ Users conference) getting confused about which language an expression might be written in.

Analysis of results

The 6 subjects gave a total of 697 answers to the operator precedence questions, an average 116.2 per subject (standard deviation 35.0, with 123.5 in 2006. The lowest number of problems answered was 75, highest number 174.

If subjects answered randomly the total number of answers expected to be correct would have a Binomial distribution. With such a distribution there is a 15.9% probability that more than 55% of answers would be correct, 2.3% probability that more than 60% of answers would be correct, and a 0.1% probability that more than 65% of answers would be correct

In this experiment 65.3% (standard deviation 8.7) of answers were correct (66.7 in 2006), poorest performer 51.2% correct (45.2 in 2006), best performer 77.8% correct (80.5 in 2006).

If the results support the hypothesis, then in some cases the name of the operand will have a significant impact on the percentage of correct answers given. There are two main possibilities, one of which has four combinations:

1. Both of the operators in the expression seen by a subject have the same context. In this case the name of the operand does not provide any additional information to subjects (or at least those used in these problems do not). Table 10 shows that this occurred in 114 (16.4%) answers, with subjects being correct in 76.3% of cases (356 answers with 71.3% correct in 2006) This is higher than the 65.3% average and while it suggests that subjects might have more accurate knowledge of the relative precedence of operators that belong to the same context, but Table 11 tells a different story.
2. The operators do not share the same context and:
   a) the operand has the same context as the operator with the highest precedence (i.e., the operator to which the operand will bind). This occurs in 251 answered problems and the answer was correct in 72.5% of cases.
   b) the name of the operand has the same context as the operator to which precedence does not bind it. This occurs in 129 answered problems and the answer was correct in 43.4% of cases.
   c) the name of the operand does not have the same context as either operator. In this case the name of the operand does not provide any additional information to subjects (or at least those used in these problems do not). This occurs in 177 answered problems and the answer was correct in 64.4% of cases.
   d) the name of the operand has the same context as both operators (this can only occur for a shift operator). In this case the name of the operand does not provide any additional information to subjects (or at least those used in these problems do not). This occurs in 26 answered problems and the answer was correct in 61.5% of cases.

It is possible for the identifier `TRUE` to be treated as having a bitwise or logical context (It was originally assigned to the logical context). Analysis of the answers show that the results are very similar whichever context it is treated as having.

The combination *Match higher/match lower* only occurs when of the operators is a shift operator.

The combination *Not match higher/Not match lower* only occurred when of the operators was either a relational, equality or a shift operator.

Table 10 suggests (first row) that subject performance is much better when the precedence decision involves operators having the same context compared to when the operators come from different contexts. However, a break-down by operator context (see Table 11) shows that this difference is primarily attributable to significantly better performance with arithmetic operators.

Table 10

Numbers of answers for various combinations of operator/operand context. *Match higher* occurs when the operand context matches the context of the operator with the high precedence (i.e., the one to which it binds). *Match lower* occurs when the operand context matches the context of the operator with the lower precedence (i.e., the one to which it does not bind). For example, *Match higher/Not match lower* refers to the case where the operand context matches that of the operand with the higher precedence and does not match the context of the operand with the lower precedence.

Operator/operand context match status	Total answers	% correct	% wrong
Both operators have same context	114	76.3	23.7
Match higher/Not match lower	251	72.5	27.5
Match higher/match lower	26	61.5	38.5
Not match higher/Not match lower	177	64.4	35.6
Not match higher/match lower	129	43.4	56.6

Table 11

Break-down of percentage of correct answers, by operator context and year of experiment, when both operators have the same context. Value in parenthesis is the total number of answers.

Year	Arithmetic	Bitwise	Logical
2007	96.2 (53)	56.5 (46)	58.3 (12)
2006	85.5 (172)	60.6 (137)	50.0 (40)

Discussion

Source code measurements confirmed that distinct operand names are commonly used for arithmetic (Table 3) and bitwise operators (Table 4). However there was no obvious distinct naming used for the operands of logical operators (Table 5).

Subject performance was significantly better when both operators were arithmetic. Arithmetic operators occur together much more often in source code, compared to bitwise operators occurring together and logical operators occurring together (see Table 3 of the 2006 article [1]). Subjects who had followed a numerate academic path will also have significantly more experience with arithmetic operators than other operators. So significantly better performance when both operators are arithmetic is to be expected.

The results, Table 10, show that the name of the operand can significantly increase or decrease the number of correct answers, depending on whether the name context is the same as the operator to which it does or does not bind. Compared to an approximate 64% correct answer rate when the name context matches both or neither operator contexts the percentage of correct answers either increased to 72.5% or decreased to 43.3%.

The use of numeric values as operands introduced unnecessary complications for little benefit. If you author had the chance to rerun the experiment they would probably be removed.

Further work

Beginner programmers are often told to use meaningful names, often without being told what constitutes a meaningful name. Naming conventions are therefore something that has to be picked up by experience. Experiments using subjects who are just about to graduate and subjects a year or so after graduating (with and without extensive software development experience during that time) could provide the data needed to calculate the impact on performance of formal training and experience with source code.

Within an expression containing only arithmetic operators (or an expression containing only bitwise operators), do developers make use of operand name information to make precedence decisions? For instance, the study by Bassok, Chase and Martin [5] found that subjects often used some name combinations (e.g., apples and baskets) prompted a division operation.

This experiment used English language names. What operand names are commonly associated with arithmetic and bitwise contexts in other human languages? Are developers who speak English as a second language as sensitive to naming context as native English speakers?

A replication of the same experiment would be very useful, perhaps with some changes to the format. Also measurements of source code written in other computer languages might find context differences in operand naming and operator usage.

Conclusion

The hypothesis that developers make use of context information contained in identifier names, when deciding which operator an operand binds to, was supported by the results of this experiment. The impact of naming information on precedence decision making was significant.

Subject precedence performance to significantly better when an expression only contained arithmetic operators, compared to when it contained operators from other contexts (mixed or other context).

Source code measurements showed that distinct operand naming only occurs for arithmetic and bitwise operators and that there was no obvious distinct naming for logical operators. ■

Acknowledgements

The author wishes to thank everybody who volunteered their time to take part in the experiment and the ACCU for making a conference slot available in which to run it. Thanks to Les Hatton for comments on an early draft.

Notes and References

[1] D. M. Jones, 'Experimental data and scripts for developer beliefs about binary operator precedence' http://www.knosof.co.uk/cbook/accu06.html, 2006

[2] Only 1.9% of expressions in source code contain two or more binary operators.

[3] D. Landy and R. L. Goldstone 'The alignment or ordering and space in arithmetic computation' in *Proceedings of the Twenty-Ninth Annual Meeting of the Cognitive Science Society*, pages 437–442, Aug. 2007

[4] G. W. Petty 'Automated computation and consistency checking of physical dimensions and units in scientific programs' in *Software-Practice and Experience*, 31(11):1067–1076, Sept 2001.

[5] M. Bassok, V. M. Chase and S. A. Martin 'Adding apples and oranges: Alignment of semantic and formal knowledge' in *Cognitive Psychology*, 35(2):99–134, Mar. 1998.

[6] D. M. Jones, Experimental data and scripts for short sequence of assignment statements study. http://www.knosof.co.uk/cbook/accu04.html, 2004.

Installing Apache & Subversion

Paul Grenyer makes Subversion accessible.

Every software development project needs a good development environment before it can even think about being successful. I'm not talking about Microsoft Visual Studio, Eclipse or any of the other Integrated Development Environments (IDEs). I am talking about the project wide resources such as:

- Version Control
- Continuous Integration
- Bug Tracking
- Wiki

These resources are there for everyone to use and allow a team to collaborate. Traditionally these sorts of services would run on some sort of *nix server, but Windows servers are getting better and many teams are using Windows exclusively now. I recently joined a startup that is using Windows exclusively and set about installing all the required services. It was quite an involved process and, as usual, I made some quite extensive notes. As I formalised these notes, the setup of each service gradually grew into material for an article.

The internet is full of setup instructions for Apache [1], Subversion [2] and combinations of the two. The Subversion book [3] also has extensive setup information and I'll refer to it quite a lot in this article (although it does not cover domain authentication). So why bother to write yet another article on Apache and Subversion? I wanted to setup domain authentication so that new starters could gain access to Subversion repositories with the minimal of configuration. The set of instructions I found on the internet for setting up domain authentication were quite good, but left out some vital details. The article addresses that with step by step instructions and actually shows what entries in the various configuration files should look like for both domain authentication and basic HTTP authentication.

Prerequisites

Hardware

I setup Apache and Subversion on an PC with a 2.0GHz AMD Athlon™ 2400+ processor and 1GB of RAM. Performance during setup was acceptable, but could have been better. I would recommend this as a minimum. All new PCs should be much faster. Lots of RAM is always a good thing. A network connection with a connection to the internet is also required (unless the applications are downloaded and transferred to the machine via another method).

Software

Apache and Subversion can be setup up on any modern Windows version (e.g. Windows XP, Vista, Windows Server 2003). I used a fresh install of Windows XP Professional with Service Pack 2 (SP2) and all updates applied.

All machines running Windows should be regularly updated and all patches applied. New releases of Apache and Subversion will rely Windows being up-to-date.

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

As well as the Windows operating system, some form of anti-virus, such as AVG [4], should be installed. Even on an internal network a firewall is a good idea. I'll be describing how to configure the Windows Firewall, but other firewalls, such as ZoneAlarm [5] can also be used.

Installing Apache Webserver

Download and install

Download the latest Win32 Apache webserver (HTTP server) MSI from the Apache website. There are a number of mirrors. Pick one close to you if possible. The file name will be along the lines of:

```
apache_2.2.6-win32-x86-no_ssl.msi
```

or

```
apache_2.2.6-win32-x86-open_ssl-0.9.8e.msi
```

There is a version that includes OpenSSL [6] and a version that doesn't. The Subversion book discusses SSL:

> Businesses that need to expose their repositories for access outside the company firewall should be conscious of the possibility that unauthorized parties could be 'sniffing' their network traffic. SSL makes that kind of unwanted attention less likely to result in sensitive data leaks.
>
> If a Subversion client is compiled to use OpenSSL, then it gains the ability to speak to an Apache server via https:// URLs. The Neon library used by the Subversion client is not only able to verify server certificates, but can also supply client certificates when challenged. When the client and server have exchanged SSL certificates and successfully authenticated one another, all further communication is encrypted via a session key.
>
> It's beyond the scope of this book to describe how to generate client and server certificates, and how to configure Apache to use them. Many other books, including Apache's own documentation, describe this task. But what can be covered here is how to manage server and client certificates from an ordinary Subversion client.

Installing Apache is easy. Simply run the `.msi` and click through the installation wizard. The only required input is the server information. The network domain is only relevant if your development environment server is on a domain. According to the Subversion book:

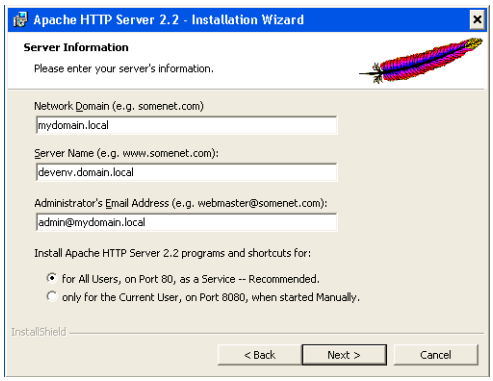
> Subversion makes use of the COPY request type to perform server-side copies of files and directories. As part of the sanity checking done by the Apache modules, the source of the copy is expected to be located on the same machine as the destination of the copy. To satisfy this requirement, you might need to tell mod_dav the name you use as the hostname of your server. Generally, you can use the `ServerName` directive in `httpd.conf` to accomplish this.

So when installing Subversion for use with Apache (as we will) it is important to specify the Server Name. The Apache documentation describes the fields (shown in Figure 1) as:

1. **Network Domain**. Enter the DNS domain in which your server is or will be registered in. For example, if your server's full DNS name is server.mydomain.net, you would type mydomain.net here.
2. **Server Name**. Your server's full DNS name. From the example above, you would type server.mydomain.net here.
3. **Administrator's Email Address**. Enter the server administrator's or webmaster's email address here. This address will be displayed along with error messages to the client by default.

Accept the default **All Users** setting on port 80.

Figure 1



The webserver is started when the installation wizard is completed.

To test that it works open a browser on the machine where Apache is installed and enter: **`http://localhost`**

That should bring up the message **`It works!`**

Firewall configuration

Now try it from another machine on the network using the Apache machines name, e.g:

`http://devenv`

or its IP address, e.g:

`http://192.168.0.103`

If the the Windows firewall or another firewall is installed and running on the Apache machine, this should fail as access to the webserver is not granted by default. To grant access through the Windows firewall:

1. Go to the Windows **Control Panel**.
2. Double click **Windows Firewall**.
3. Go to the **Exceptions** tab, click **Add Program** and then **Browse**.
4. Browse to the Apache `bin` directory (usually `C:\Program Files\Apache Software Foundation\Apache2.2\bin`) and select `httpd.exe`.
5. Dismiss the remaining dialogs by clicking **OK**.

Figure 2

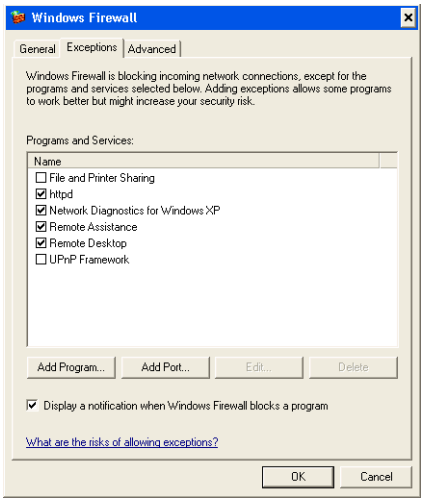


Figure 2 shows httpd added to the list of **Exceptions**.

Now try to access the development environment server from a remote machine again. This time it should work.

For some final feel good factor modify the `index.html` file (found in: `C:\Program Files\Apache Software Foundation\Apache2.2\htdocs`) and check that the changes are reflected both locally and remotely.

Installing Subversion version control system

Download and install

There are two different releases of version 1.4.5 of Subversion. One with binary compatibility for version 2.0 of Apache and one with binary compatibility for version 2.2 or higher of Apache. As we're using the latest Apache version we need the version 2.2 or higher release of Subversion. Unfortunately, this release doesn't have an msi installer and therefore the zip file containing the Win32 binaries must be downloaded from the Subversion site.

1. Unzip the Subversion Win32 binaries (e.g.: `C:\Program Files\svn-win32-1.4.5`).
2. Add the Subversion bin directory to your path environment variable (`C:\Program Files\svn-win32-1.4.5\bin`).
3. Move the `mod_authz_svn.so` and `mod_dav_svn.so` Apache modules from the Subversion `bin` directory (`C:\Program Files\svn-win32-1.4.5\bin`) to the Apache modules directory (`C:\Program Files\Apache Software Foundation\Apache2.2\modules`).

Creating a repository

A repository is needed before Subversion can be configured for use with Apache. An easy repository to create is an infrastructure repository that could be used to store configuration files for Apache and Subversion.

1. Create a repositories directory to store all repositories (e.g. `C:\Repositories`).
2. Open a command prompt and type:

```
svnadmin create c:\repositories\infrastructure
```

Configuring Apache

There are two basic methods of accessing Subversion over a network: svnserve and Apache. The advantages of using Apache, according to the Subversion book, are:

- Allows Subversion to use any of the numerous authentication systems already integrated with Apache (including Windows domain authentication).
- No need to create system accounts on server.
- Full Apache logging.
- Network traffic can be encrypted via SSL.
- HTTP(S) can usually go through corporate firewalls.
- Built-in repository browsing via web browser.
- Repository can be mounted as a network drive for transparent version control.

It states that the disadvantages are:

- Noticeably slower than svnserve, because HTTP is a stateless protocol and requires more turnarounds.
- Initial setup can be complex.

I have never found Subversion access via Apache too slow, but there is always room for improvement and, as we'll see below, initial setup need not be complex.

To configure subversion for use with Apache open `httpd.conf` (found in `C:\Program Files\Apache Software Foundation\Apache2.2\conf`) and add the following lines to the end:

```
LoadModule dav_module modules/mod_dav.so
LoadModule dav_svn_module modules/mod_dav_svn.so
LoadModule authz_svn_module modules/
   mod_authz_svn.so
```

This tells Apache to load the modules required for Subversion. After this add the following:

```
<Location /svn>
  DAV svn
  SVNParentPath "C:/Repositories"
  SVNListParentPath on
</Location>
```

This tells Apache where the Subversion repositories are stored and that they should be accessed as `http://<servername>/svn/<repositoryname>` and to pass the handling of those URLs onto the Subversion DAV Layer. **`SVNParentPath`** tells Apache that all repositories are located in `C:/Repositories`. Therefore only one **`Location`** block is required to serve several repositories. Setting **`SVNListParentPath`** to **`on`** lists all the repositories in the

SVNParentPath when accessing the `http://<servername>/svn/` URL.

Save `httpd.conf` and restart the Apache service (**Control Panel → Administrative Tools → Services**). To test the configuration try and access the infrastructure repository. For example:

```
http://devenv/svn/infrastructure/
```

This should bring a page resembling the following:

```
Revision 0: /
------------------------------------------------
Powered by Subversion version 1.4.5 (r25188).
```

TortoiseSVN

Checking files into a Subversion repository via the command line soon becomes very tedious. TortoiseSVN [7] is tool that integrates into Windows Explorer and provides a GUI for common Subversion commands, such as check-in and check-out and provides colour overlays to indicate the modified status of checked out files.

Configuring Domain Authentication

The simple Subversion configuration described above allows anyone who can access the Apache and Subversion server to access the Subversion repositories. This is probably fine for small organisations using the server internally but larger organisations or those wanting to use Subversion over the Internet require authentication.

Apache can be configured to control access to Subversion via Windows Domain Authentication. This is useful if the organization already has its users on a domain and doesn't want to have to add new users to Subversion, who are already on the domain. (If you don't have a domain, see Basic HTTP Authentication below for an alternative).

Open `httpd.conf` again and add the following under the `mod_dav.so`, `mod_dav_svn.so` and `mod_authz_svn.so` module load statements:

```
LoadModule dav_module modules/mod_dav.so
LoadModule dav_svn_module modules/mod_dav_svn.so
LoadModule authz_svn_module modules/
   mod_authz_svn.so
LoadModule sspi_auth_module modules/
   mod_auth_sspi.so
```

The `mod_auth_sspi.so` module is used by Apache for the Domain Authentication. Add the following to the location block.

```
<Location /svn>
  DAV svn
  SVNParentPath "C:/Repositories"

  AuthName "SVN Server"
  AuthType SSPI
  SSPIOfferBasic On
  SSPIAuth On
  SSPIAuthoritative On
  SSPIDomain MYDOMAIN

  Require valid-user
  AuthzSVNAccessFile "conf/svnaccess.conf"
</Location>
```

A brief description of each entry is given below:

AuthName	An arbitrary name that you give for the authentication domain. Most browsers will display this name in the pop-up dialog box when the browser is querying the user for his name and password.
AuthType	Type of authentication system to use.
SSPIOfferBasic	When set to on allows non-Internet Explorer based clients to access the repository.
SSPIAuth	When set to on to activates SSPI authentication.
SSPIAuthoritative	When set to on prevents other authentication methods being used if SSPI fails.
SSPIDomain	The name of the domain to authenticate against. The domain name must be in capitals.
Require	When set to valid-user, tells Apache that all requests require an authenticated user.
AuthzSVNAccessFile	Specifies a file containing the permissions policy for repositories.

There are more details for most of the above settings in the Subversion book.

Save `httpd.conf`, restart Apache and try to access the repository via a browser again. You should receive a 'Forbidden' message. This is because there is no file containing details of user permissions on the repository.

Create a file called `svnaccess.conf` in the `conf` directory (`C:\Program Files\Apache Software Foundation\Apache2.2\conf`). Open the file and add:

```
[infrastructure:/]
MYDOMAIN\user.name = rw
```

Substitute **MYDOMAIN** for the name of your domain (in upper case) and **user.name** for the user name of a user within that domain.

Save `svnaccess.conf` and restart Apache. Assuming the user specified is logged into the domain on the machine trying to access the infrastructure repository, access should be allowed.

Don't forget to check `svnaccess.conf` into the repository.

There must be an entry for each repository. As many users as required can be added, each on a separate line. The first word inside the square brackets is the name of the repository you are setting user permissions for. Following the colon is the path to the repository relative to **SVNParentPath**. The next line contains the domain and user name of the user being given permissions and **rw** indicates read and write permissions.

More details of the types of permissions and how to setup groups can be found in the Subversion book.

Configuring basic HTTP authentication

If you don't have a domain there is a basic authentication system supported by Apache. The system allows users to be created with passwords. The Subversion book includes detailed information on how to do this, so I will only cover the basics here.

Start by creating a user. Open a command prompt and type:

```
cd C:\Program Files\Apache Software Foundation\
   Apache2.2\bin
htpasswd -cm ..\conf\svn_auth_file paul
New password: *******
Re-type new password: *******
Adding password for user paul
```

This will create a file called `svn_auth_file`, containing the user name and an encrypted password for a user called **Paul**, in the `conf` directory (`C:\Program Files\Apache Software Foundation\Apache2.2\conf`). The **c** command line argument creates a new file and the **m** argument forces MD5 encryption of the password. Further users can be added by dropping the **m** command line argument:

```
htpasswd.exe -m ..\conf\svn_auth_file charlotte
New password: ****
Re-type new password: ****
Adding password for user charlotte
```

Personal Bazaar

Steve Love has a round-up of techniques for making Bazaar do what you want.

In a couple of previous articles [1] [2], I talked about how to use the Bazaar Version Control System [3] (or bzr for short) for regular tasks associated with version control – checking out, branching, merging, checking in. In this article I want to explore a little way off the beaten path, and look at some of the more advanced features and facilities of bzr. That's not to say that these features aren't regular version control tasks, though; only that bzr does some things differently and sometimes more comprehensively than other similar tools.

The behaviour of most bzr commands is most easily changed from the command line; most bzr commands accept one or more arguments which augment their behaviour. Sometimes, however, more persistent changes are needed, and bzr looks to its configuration files for those things you can't be bothered remembering to type in each time you run a bzr command. We'll take a short tour of the various configuration options understood by the bzr command line.

Sometimes, even with command line arguments and config file entries, bzr doesn't quite do what you need, and so bzr supports the concept of a plug-in, which might enhance the feature set, change some existing feature or add a completely new feature to bzr. There are some plug-in components which are indispensable to any serious undertaking, so a section on plug-ins is similarly mandated.

It's quite common for project admins to configure a version control system with notifications when important events – such as check-ins – occur. bzr manages this with hooks. A hook-handler is special kind of bzr plug-in, which gives you great flexibility and expressiveness in what can be achieved easily.

But before we get to any of this, let's have a look at the output from bzr log.

History revision

The output shown in Listing 1 is the log from a branch with a single check-in. In common with many version control systems, it shows the revision number, the time of the check in, who made it, and the message associated with that commit. By default, bzr uses a combination of your login name and the host name of your machine to generate a committer name (See sidebar).

```
revno: 1
committer: steve <steve@salisbury>
branch nick: paperclip
timestamp: Tue 2008-01-15 17:15:48 +0000
message:
  Initial project
```

Your Bazaar Identity

Each time you make a check-in, it is logged against your ID and email address, which shows up in the log output. This ID can be configured either globally, or for a specific branch, so in theory you could have different credentials for different projects. The command `bzr whoami "Steve <steve@arventech.com>"` sets my global ID and email address, and will be used for all branches where the ID hasn't been specifically set to something else. The command `bzr whoami --branch "Steve <steve@arventech.com>"` sets the ID for just the current branch, overriding the globally set value. To display the ID which will be used for the next check-in, use `bzr whoami`.

STEVE LOVE

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

Installing Apache and Subversion (continued)

Open `httpd.conf` and add the following to the location block:

```
<Location /svn>
  DAV svn
  SVNParentPath "C:/Repositories"
  AuthName "SVN Server"
  AuthType Basic
  Require valid-user
  AuthUserFile conf/svn_auth_file
</Location>
```

`AuthName`	An arbitrary name that you give for the authentication domain. Most browsers will display this name in the pop-up dialog box when the browser is querying the user for his name and password.
`AuthType`	Type of authentication system to use.
`Require`	When set to valid-user, tells Apache that all requests require an authenticated user.
`AuthUserFile`	Specified the location of the password file.

Save `httpd.conf` and restart Apache. Try accessing the infrastructure repository again. A dialog box asking for a user name should be displayed (Figure 3). Enter the user name and password of one of the users in your password file. Click **OK** and access the repository as normal.

Where next?

That's all there is to it. The Apache home page can be customized as needed or better still a Wiki installed for team collaboration. Subversion If needed, more repositories can be easily added to Subversion and accessed from any number of client machines.

In the next few issues I'll be describing how to install a Wiki, bug tracking software and an instant messaging client. ■

Figure 3

References

[1] http://httpd.apache.org/
[2] http://subversion.tigris.org/
[3] http://svnbook.red-bean.com/
[4] http://free.grisoft.com/
[5] http://www.zonealarm.com/
[6] http://www.openssl.org/
[7] http://tortoisesvn.tigris.org/

Listing 1

```
------------------------------------------------
revno: 2
committer: Steve <steve@salisbury>
branch nick: paperclip
timestamp: Wed 2008-01-16 11:17:35 +0000
message:
  Merged from Dave
    ------------------------------------------
    revno: 1.1.2
    committer: Dave
    branch nick: paperclip_dave
    timestamp: Wed 2008-01-16 11:17:23 +0000
    message:
      Merged from Sal
        --------------------------------------
        revno: 1.1.1.1.1
        committer: Sal
        branch nick: paperclip_sally
        timestamp: Wed 2008-01-16 11:17:08 +0000
        message:
          Started work on the docs
    ------------------------------------------
    revno: 1.1.1
    committer: Dave
    branch nick: paperclip_dave
    timestamp: Wed 2008-01-16 11:16:42 +0000
    message:
      Started work on the build
------------------------------------------------
revno: 1
committer: steve <steve@salisbury>
branch nick: paperclip
timestamp: Tue 2008-01-15 17:15:48 +0000
message:
  Initial project
added:
  build/
  doc/
pending merges:
  Dave 2008-01-16 Merged from Sal
    Sal 2008-01-16 Started work on the docs
    Dave 2008-01-16 Started work on the build
```

The log output becomes a bit more interesting when there have been commits and merges from different branches of the project.

Figure 1 shows two further branches made from Steve's branch. In fact, the branch being worked on by Sally was branched from Dave's branch, then Dave merged Sally's changes, and lastly Steve merged Dave's branch, which included the changes made by Sally.

What's interesting about the log is the visible indication – by the power of indentation – of the branching history. This indentation is also reflected in the output from **`bzr status`** in Steve's branch, *after* merging from Dave, but *before* committing it.

The 'pending merges' section shows not only what changes have just been merged, but an idea of the ancestry of those changes.

Bazaar behaviour

There are a number of ways to get bzr to do what you want it to do. The first and most obvious is via the command line, as shown in the previous section, where you either invoke a particular command – like **`bzr whoami`** – to explicitly set an option.

The second common way is to set an environment variable. The syntax for exactly how to do this will differ from system to system, and so I'll just mention some of the common variable names here.

- **`BZR_HOME`** sets what bzr thinks of as the home directory. This is where bzr will look for plug-ins (see a later section), and the main configuration file (covered next).
- **`BZR_EMAIL`** sets the email address for the current user. It has the same content as the payload for **`bzr whoami`**.
- **`BZR_PLUGIN_PATH`** sets the directories bzr looks in for plugins explicitly.
- **`BZR_EDITOR`** sets the path to an editor which will be used if no **`-m`** argument is provided for a commit.

Finally, bzr also looks in its configuration files to determine if you've decided you want it to do something different from the defaults. bzr has two main configuration files, which may be overridden on a per-branch basis.

The global config files are in `$BZR_HOME/.bazaar` on a Linux or Unix system (or `$HOME` if `$BZR_HOME` is not set), and in `%APPDATA%\Bazaar\2.0` on Windows (`%APPDATA%` is usually `C:\Documents and Settings\<user name>\Application Data`) [4]. The main configuration settings are in `bazaar.conf` and a file named `locations.conf` can be used for specific branch locations. Each branch can have its own local configuration, which will be in the `.bzr/branch/branch.conf` file.

This file can contain settings for the email (which is overridden by setting the **`BZR_EMAIL`** environment variable), the path to the editor (also overridden if set in the environment), settings for signing commits, emailing merge requests, and a whole slew of other bits and pieces, including information used by some plug-ins (we're getting to that, honestly!).

You can also set up aliases for commands here, so that if you decide that typing **`bzr whoami`** is too boring, you can set an alias so that **`bzr echo-my-email-address`** does the same thing. Aliases can also be used to make existing commands do different things. For example, the **`bzr commit`** command has a **`--strict`** argument which will refuse to make a check-in if there are unknown files in the branch. Setting an alias can make this behaviour the default rather than having to specify that argument each time, by adding the following to the `bazaar.conf` file (adding the **`[ALIASES]`** section if it doesn't already exist):

```
[ALIASES]
ci=ci --strict
```

This can be a very powerful way of customising bzr to your own preferences.

Some other interesting aliases you may want to try out, and maybe tweak to your own needs:

```
lastlog=log -r-2..-1
ll=log --line -r-10..-1
make_repo=init-repo --no-trees --default
```

Custom paint job

Sometimes configuring bzr's existing behaviour just isn't enough, and you want to make it do something entirely different by adding your own commands – or even repurposing existing commands to do something else.

It is beyond the scope of this article to go into the details of writing your own plug-ins for bzr; if you're interested, the bzr documentation is quite comprehensive on the subject. Just briefly, plug-ins are written in Python, the native language of bzr, and so the full flexibility and power of the language and libraries, coupled with the bzr library, is at your disposal. Instead, we'll take a quick look at some of the more common plug-ins that any bzr installation would be, frankly, embarrassed to be without. All of these plug-ins are available from [5].

Bzr tools – the basic kit

This plug-in contains several common utilities to enhance bzr with useful stuff. Highlights include a facility to clean-up a branch, removing unversioned files and ignored items, and the ability to 'shelve' changes in a revision for later retrieval.

Diff tools – beyond the built-in Diff

Typing **bzr diff** at the command prompt generates the classic unified diff format output which, while useful for many things, is largely incomprehensible to a human reader. What it's most useful for is a sane input to a variety of graphical diff tools. The diff-tools plug-in allows you to configure a diff tool of your choice.

External merge – beyond Diff!

Graphical diff tools are great for viewing the difference between two files, or two revisions of a file, but some go even further and provide facilities for managing merge conflicts. When a file is conflicted as a result of a merge, there are commonly four versions of that file in operation: the conflicted file, the revision of the file causing the conflict, the base revision of each of those, and the final, resolved, output version. This plug-in allows bzr to call such an external program (my personal recommendation is Kdiff3 – which despite appearances runs perfectly happily on MS Windows. See [6]).

Qbzr – Cute Tools

This plug-in has various graphical extensions for viewing the revision history log, a graphical diff tool, setting some of the configuration variables and browsing the contents of a branch.

Installing plug-ins

Installing new plug-ins could hardly be easier in bzr. The system-wide plug-ins directory is a `plugins` sub-directory of wherever bzr is installed (on MS Windows, this is `%ProgramFiles%\Bazaar`). Plug-ins can also be installed on a per-user basis, and the location for this is `$HOME/.bazaar/plugins` on Linux/Unix, and `%APPDATA%\bazaar\2.0\plugins` on MS Windows (the `plugins` sub-directory here may need to be created manually). Beyond knowing where to put the plug-ins, all that remains is to download one that takes your fancy and drop it in.

No, really!

Some plug-ins require configuration (for example, the **extmerge** plug-in mentioned above needs an entry in the `bazaar.conf` file to tell it the full path to the merge tool you wish to use).

The presence of a correctly installed plug-in can be checked using the **bzr plugins** command, and they will also be listed in **bzr help commands**. Each plug-in integrates nicely with the bzr command line help, so, for example, **bzr help extmerge** gives help on the external merge tool plug-in.

Running errands

It's fairly common in large teams, or those working in a truly distributed manner, to arrange for the versioning system to notify some or all of the team when a commit occurs, or something else that changes the repository like a rollback. bzr's commit hooks are special types of plug-in, written in Python as plug-ins are, giving you great power about what to do for your notifications.

The standard hooks are:

- **pre_commit** occurs before a check in has completed
- **post_commit** occurs after a check-in is committed
- **post_uncommit** occurs after an **uncommit** has finished
- **post_push** occurs after a **push** has occurred
- **post_pull** occurs after a **pull** has happened
- **set_rh** occurs whenever any change to revision history happens – **commit**, **push**, **pull** and **uncommit** all change the revision history.

You install the hook handler in the same way as for a plug-in. The bzr documentation explains clearly what is needed, but I'll reproduce the simple example from that documentation here:

```
from bzrlib.branch import Branch

def post_push_hook(push_result):
    print ("The new revno is %d"
    % push_result.new_revno)

Branch.hooks.install_hook(
    'post_push', post_push_hook)
Branch.hooks.name_hook(
    post_push_hook, 'My post_push hook')
```

(From [7]). This shows exactly how simple it is to install a hook handler for your repository.

In this simple example, the idea is to print the new revision number of the remote branch after a **push** has occurred from the current branch.

Create a file called `post_push.py` in your plug-ins directory (see 'Installing plug-ins'), and add the lines of Python script above.

```
The new revno is 6
All changes applied successfully.
Pushed up to revision 6.
```

That is all you need! Test your hook by **push**ing changes to a different branch. You should see output similar to that shown. The first line of the output was generated by the hook, and the latter lines are the regular output from the **push** command. Of course, for a real one, you would probably want to do something more interesting – but then, you have the whole of Python available to you, so be creative!

Recap

So after all that, have you tried bzr yet? As a learning tool for those new to revision control, it is ideal because it works straight out of the box, with no services or configuration to perform. A single command – **bzr init** – is all that is required.

bzr is at version 1.1 at the time of writing, and has therefore passed its 'version one' milestone. It is a very stable, performant and mature tool, gaining favour in the open-source community quickly. Distributed Version Control isn't necessarily for everyone, and bzr even manages to cover that base with dedicated shared repositories and centralised workflow integrated completely with bzr's distributed nature.

Features such as off-line working, smart merging, simple plug-in development, indented log files and all the others are more than just 'neat' and 'nifty'; these things are designed to make the gap between the developer and tool much smaller, to support the user instead of getting in the way.

For such a small tool, it handles a big job – The Version Control Job – with ease. ■

Acknowledgement

Many thanks to Tim Penhey for his comments and suggestions on initial drafts of this article.

Notes and references

[1] Steve Love, 'The Bazaar Thing', *CVu* August 2007
[2] Steve Love, 'The Version Control Job', *CVu* October 2007
[3] http://bazaar-vcs.org
[4] For MS Windows users: it seems to be a little known fact that you can enter **cd %APPDATA%** at the command line, and even just **%APPDATA%** in the Windows Explorer address bar to go straight to this location.
[5] http://bazaar-vcs.org/BzrPlugins
[6] http://kdiff3.sourceforge.net/
[7] http://doc.bazaar-vcs.org/bzr.dev/en/user-guide.index.html

A Simple Calculator in Tkinter and wxPython

Ivan Uemlianin compares two python GUI toolkits.

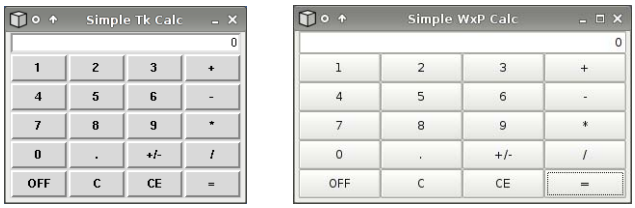
Overview

This article describes my experiences writing a very simple calculator GUI – the 'Hello World!' of GUI development – in two of the main GUI toolkits for Python: Tkinter and wxPython. Even with such a simple applcation I found it possible to evaluate the toolkits relative to each other.

In order to try and follow 'best practice' (and to avoid evaluating my own programming ability) I sourced initial implementations externally: my Tkinter-based calculator is based on the two calculator GUIs given in Chapter three of *Python and Tkinter Programming* [11], my wxPython-based calculator is based on several demo calculators from the web [1]. Figure 1 shows screenshots of the two calculators at launch.

There is also an informal, objective, though obviously biased, discussion of TKinter and wxPython at the wxPython wiki [2].

Figure 1



Requirements and downloads

The following scripts require Python, Tkinter and wxPython. On most linux systems, the package management software will install these for you (on Debian they are called python, python-tk and python-wxtools, and there is a handful of other packages available). On Windows, you have to install Python yourself: download a `.msi` from the Python website [3]. This includes Tkinter. For wxPython, go to the wxPython website and download their `.msi` [4].

Although not a requirement, the Boa Constructor IDE [5] is useful for debugging scripts which use wxPython widgets, as it catches and reports on wx exceptions.

This article and all the calculator scripts are available from my website [6].

IVAN UEMLIANIN

Ivan is a self-employed software developer. His background is in academic linguistics, psychology and philosophy, and his current focus is on speech technology. He has an irrational desire to learn C++. He can be contacted at ivan@llaisdy.com

Code overview

`myCalc.py` contains the class **`AbstractCalculator`**, which is a superclass for **`TkCalc`** (in `myTkCalc.py`) and **`WxCalc`** (in `myWxCalc.py`). I've aimed to put everything that the other two can hold in common into **`AbstractCalculator`**, making it easier to compare the two widget toolkits by comparing **`TkCalc`** and **`WxCalc`**. **`TkCalc`** and **`WxCalc`** contain the same methods: **`__init__`**(self), **`buttonPress`**(self, event), **`getDisplay`**(self), **`setDisplay`**(self, value), and **`quitApp`**(self).

In the last three of these, the difference between Tkinter and wxPython is purely lexical. Setters, getters and quitters are shown for comparison in Table 1.

I discuss the differences between **`init`** and **`buttonPress`** below.

Known bugs

In practice, the term 'known bug' seems to refer to a bug that no-one can be bothered to fix. The bugs I list here are not central to my purpose for writing these scripts (which was to compare Tkinter and wxPython). Fixing them is left as an exercise for the reader.

- Display

  The display displays operators as well as numbers (e.g. **123** + **45**), only refreshing after an = or **C**/**CE**. Of course, real calculators don't do this.

- The **`toggleSign()`** method

  The **`toggleSign()`** method (triggered by pressing the +/- key) just prepends/removes a **-** at the beginning of the display. This is too primitive (see Table 2).

Table 2

Display	toggleSign (Display)
123	-123
-23	123
123 -45	-123 -45 # should be 123 + 45

Table 1

AbstractCalculator	TkCalc	WxCalc

`def getDisplay(self):`
`  return self._display`
`def setDisplay(self, value):`
`  self._display = value`
`display = property(getDisplay,`
`  setDisplay)`
`def quitApp(self): pass`
	`def getDisplay(self):`
	`  return self._display.get()`
	`def setDisplay(self, value):`
	`  self._display.set(value)`
	`display = property(getDisplay,`
	`  setDisplay)`
	`def quitApp(self): self.quit()`
		`def getDisplay(self):`
		`  return self._display.GetValue()`
		`def setDisplay(self, value):`
		`  self._display.SetValue(value)`
		`display = property(getDisplay,`
		`  setDisplay)`
		`def quitApp(self): self.Destroy()`

Comparison of Tkinter and wxPython

SLOCs

The two scripts are about the same size.

Imports

The Tkinter and wx packages are imported slightly differently:

```
>>> from Tkinter import *
>>>
```

but:

```
>>> from wx import *
```

```
Traceback (most recent call last):
   File "<stdin>", line 1, in ?
AttributeError: 'module' object has no
   attribute '__DocFilter'
>>> import wx
>>>
```

Note that `from module import *` is discouraged, especially on Windows and Mac platforms [7].

to widgets as part of their initialisation, while wxPython calls a separate `Bind` method (see the loops to set up the buttons in the listings).

Because the documentation was not 100% comprehensive (see below) and because I decided to limit the time spent on these GUIs, there are a couple of minor details which remain mysterious:

For the calculator display, I found it necessary to use read-only text entry boxes to disable the user from entering text directly into the display (i.e., a disabled `Entry` on Tkinter, and a `READONLY TextCtrl` on wxPython).

Table 3

TkCalc	WxCalc

`keysym2label = {'plus': '+',`
`'minus': '-',`
`'asterisk': '*',`
`'period': '.',`
`'slash': '/',`
`'equal': '=',`
`'Return': '=',`
`'Delete': 'C',`
`'BackSpace': 'C',`
`'q': 'OFF'`
`}`
	`keyCode2label = {wx.WXK_RETURN: '=',  # return`
	`wx.WXK_BACK: 'C',  # backspace`
	`wx.WXK_DELETE: 'C',  # delete`
	`17: 'OFF'  # q`
	`}`

`def buttonPress(self, event):`
`k = event.keysym`
`if k not in '1234567890':`
`k = self.keysym2label.get(k)`
`self.keyAction(k)`
	`def buttonPress(self, event):`
	`c = event.GetKeyCode()`
	`k = chr(c)`
	`if k not in '1234567890+-*/.=':`
	`k = self.keyCode2label.get(c)`
	`self.keyAction(k)`

buttonPress( )

Because of the design of the superclass (see above), I have had to overload only five methods, and these show only minor idiomatic differences.

`buttonPress()` is the method bound to key-press events and, although the two versions differ only a little, it demonstrates the different approach to key events in the two toolkits (see Table 3).

In Tkinter, event objects of any kind have a standard set of attributes [8] including (for keyboard events) `char`, `keycode` and `keysym`. `keycode` is a code relating to the key pressed (not the ASCII character code), while `char` and `keysym` both return strings: for alphanumerics both return the string of the key pressed; for other keys (e.g., =, **<left-ctrl-key>**, or **Return**) `keysym` returns a description (i.e., `equal`, `Control_L`, `Return`). Note that two-key combinations (e.g., **shift-8**) count as two keypress events: keycode will give the same code for both events; `keysym` and `char` show the effect of the combinations (i.e., with **shift-8** `keysym` returns `Shift_R` then `asterisk`).

In wxPython, there are three different kinds of keypress events [9]: `wx.EVT_KEY_DOWN`, `wx.EVT_KEY_UP` and `wx.EVT_CHAR`. These event types have the same methods, the relevant one here being `GetKeyCode()`, which returns the ASCII value of the key pressed, or a wxWidgets constant [10] for non-alphanumerics. However, `wx.EVT_CHAR.GetKeyCode()` shows combination effects (like `keysym` above), while `wx.EVT_KEY_DOWN/UP.GetKeyCode()` do not.

__init__()

The `__init__()` method is where the two calculators differ most. See Table 4 for a side-by-side comparison.

The toolkits differ in the way they add widgets to the main frame. Both use the first parameter of a widget's `__init__()` for the widget's parent. However, wxPython uses an explicit sizer to place a widget inside the parent (i.e., with `sizer.Add()` methods); Tkinter uses the widget's `pack()` method to place it. Similarly, in Tkinter commands can be bound

The documentation points to `Label` or `Panel` widgets as being more appropriate, but I couldn't get these to work properly (e.g., Tkinter `Label` insisted on central justification, whatever I specified in initialisation).

In wxPython, the `lambda` function bound to each `Button` – `lambda e, k=key: self.keyAction(k)` – must carry the first argument `e` for the event object, even though it's not used (the Tkinter `lambda` is otherwise identical).

Once the GUI has been initialised, both toolkits use a `mainloop` method to set it going, and waiting for events. Here again, wxPython is slightly more complicated. In wxPython the `mainloop` is not run by the GUI object itself, but by a separate application object (in WxCalc, the application object `wx.PySimpleApp` is attached to the calculator at the beginning of `__init__()`). This helps abstract overall management of the application away from particular software objects or GUI widgets.

Documentation

Each toolkit has a book devoted to it published by Manning: Tkinter has *Python and Tkinter Programming* [11]; wxPython has *wxPython in Action* [12]. The two books are quite different in style. I found the book on Tkinter quite inspiring in its approach to Python in general, beyond GUI coding. The wxPython book is a huge FAQ: it's well-written and very useful, but not an inspiring read. On the other hand, the Tkinter book has 250 pages of fairly comprehensive appendices and a 30 page index; the wxPython book does not attempt to be comprehensive, has no appendices, and only six pages of index.

For wxPython, the book is nice to get started, but you really have to use the online documentation [13]. This is comprehensive but it's not actually wxPython documentation: the wxPython online docs page is actually a frame wrapped around the wxwidgets online docs page. Translating between C++ and Python is a bit of a drag but fairly trivial – for example, the class and method names and parameters and all the semantics are the same.

Tkinter also has good online documentation [14]. I haven't used it much, as [11] answers most of my questions.

Table 4

TkCalc	WxCalc

`def __init__(self):`
`  AbstractCalculator.__init__(self)`
 	
`  Frame.__init__(self)`
`  self.pack(expand=NO, fill=NONE)`
`  self.master.title('Simple Tk Calc')`
`  self.master.resizable(0,0)`
 	
`  self._display = StringVar()`
`  self._display.set('0')`
`  Entry(self, justify=RIGHT, relief=SUNKEN,`
`  disabledforeground='black',`
`  disabledbackground='white',`
`  state=DISABLED,`
`  textvariable=self._display).pack(side=TOP,`
`  expand=YES, fill=BOTH)`
 	
`  for row in self.keyLayout:`
`  rowFrame = Frame(self)`
`  for key in row:`
`  Button(rowFrame, text=key,`
`  command=lambda k=key: self.keyAction(k),`
`  width=4).pack(side=LEFT, expand=NO,`
`  fill=NONE)`
`  rowFrame.pack(side=TOP, expand=YES,`
`  fill=BOTH)`
 	
`  self.bind('<KeyPress>', self.buttonPress)`
`  self.focus_set()`
`  self.mainloop()`
	`def __init__(self):`
	`  AbstractCalculator.__init__(self)`
	`  self.app = wx.PySimpleApp()`
	
	`  wx.Frame.__init__(self, None, -1,`
	`  "Simple WxP Calc")`
	`  sizer = wx.BoxSizer(wx.VERTICAL)`
	
	`  self._display = wx.TextCtrl(self, -1, '',`
	`  style = wx.TE_READONLY | wx.TE_RIGHT)`
	`  sizer.Add(self._display, 0, wx.EXPAND)`
	
	`  gsizer = wx.GridSizer(4, 4)`
	`  for row in self.keyLayout:`
	`  for key in row:`
	`  b = wx.Button(self, -1, key)`
	`  b.Bind(wx.EVT_BUTTON,`
	`  lambda e, k=key: self.keyAction(k))`
	`  b.SetFocus()`
	`  gsizer.Add(b)`
	`  sizer.Add(gsizer, 1, wx.EXPAND)`
	
	`  self.SetSizer(sizer)`
	`  sizer.Fit(self)`
	
	`  self.Bind(wx.EVT_CHAR, self.buttonPress)`
	`  self.display = '0'`
	
	`  self.Show()`
	`  self.app.MainLoop()`

The documentation for neither system was fully reliable, so I occasionally had to use the absolute default and spend some time with Google. This seems to be par for the course with most software these days, open or closed source.

Conclusion

For simple GUIs I have found Tkinter easier to work with: less code is needed, there are fewer idiosyncracies, there is more/better documentation. WxPython seems to be designed with at least the possibility of more complex applications in mind. For example, Tkinter does not provide a **`tree`** widget (e.g., for browser GUIs), and Grayson has to code one up (in `Example_8_10.py`); wxPython provides **`TreeCtrl`**, a direct wrapper around the C++ wxWidgets class **`wxTreeCtrl`**. Consequently, for more complex GUIs I use wxPython. Presumably any porting from Python to C++ would be simpler from wxPython than from Tkinter, but I have yet to investigate porting a wxPython application to wxWidgets. ■

References

[1] Simple calculators in wxPython:
http://wiki.wxpython.org/index.cgi/CalculatorDemo
http://pythonwise.blogspot.com/2006/05/wxpython-calculator-in-50-lines-of.html
http://www.devshed.com/c/a/Python/Designing-a-Calculator-in-wxPython/

[2] http://wiki.wxpython.org/Choosing_wxPython_over_Tkinter

[3] http://www.python.org

[4] http://www.wxpython.org

[5] http://boa-constructor.sourceforge.net/

[6] http://www.llaisdy.com/tech/python/calc.html [TODO]

[7] http://www.python.org/doc/current/tut/node8.html#SECTION008410000000000000000

[8] pythonware doc ch7. Events and bindings

[9] wxPython/wxPython-2.6.3.2/docs/wx/wx_wxkeyevent.html

[10] wxPython/wxPython-2.6.3.2/docs/wx/wx_keycodes.html

[11] Grayson, J. (2000) *Python and Tkinter Programming*. Manning. ISBN: 1-884-77781-3. http://www.manning.com/grayson.

[12] Rappin, N & Dunn, R. (2006). *wxPython in Action*. Manning. ISBN: 1-932394-62-1. http://www.manning.com/rappin.

[13] http://www.wxpython.org/onlinedocs.php

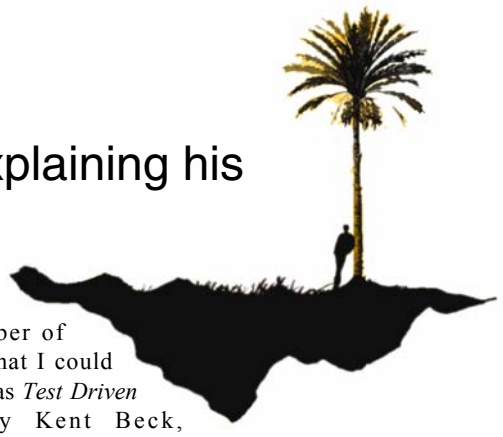
[14] http://www.pythonware.com/library/tkinter/introduction/index.htm

We'd love to hear from you!

If you read something in C Vu that you particularly enjoyed, you disagreed with or that has just made you think, why not put pen to paper (or finger to keyboard) and tell us about it? Send your thoughts to editor@accu.org

Desert Island Books

Paul Grenyer starts a new series by explaining his own selections.



I thought this was going to be easy as I had no doubt what my first main book would be, but then it got harder. A lot harder. Do I choose a design patterns book? What about a process book? Or a technique book? Or more straight language books? This is how I got on.

Programming books

The C++ Standard Library: A Tutorial and Reference

by Nicolai M. Josuttis, published by Addison Wesley

ISBN-10: 0201379260 ISBN-13: 978-0201379266

This book more than any other changed my career for the better. I was fresh out of university, had my first C++ job and only knew a little C. A number of ACCUers spear headed by Phil Nash and John Crickett were guiding me to better things. Phil Nash in particular persuaded me to invest in better books. When I got Josuttis I read it pretty much cover to cover. It's the book I go back to the most (when I'm doing C++) and I wouldn't be without it. Reading this book and learning about the C++ standard library allowed me to be the only candidate to complete a programming test for new position I went for and they gave me the job off the back of that.

C++ Templates: The Complete Guide

by David Vandevoorde and Nicolai M. Josuttis, published by Addison Wesley

ISBN-10: 0201734842 ISBN-13: 978-0201734843

This is where it gets difficult. For my second choice I couldn't decide between Scott Meyers' Effective Series, Herb Sutter's Exceptional Series and Vandevoorde and Josuttis' templates book. The Effectives and Exceptionals have no doubt made me a much better C++ programmer, but I mostly absorbed the information as I read and don't gt back to them so often. If I was on a desert island I'd want the templates book as I love the power that templates give C++ and, when programming in C++, constantly go back to the templates book for reference. That's the one I'd want on a desert island.

Design patterns : elements of reusable object-oriented software

by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, published by Addison Wesley

ISBN-10: 0201633612 ISBN-13: 978-0201633610

There are a number of technique books that I could have chosen. Such as *Test Driven Development* by Kent Beck, *Refactoring* by Martin Fowler or *Working with Legacy Code* by Micheal Feathers. However, these books confirmed stuff I'd mostly worked out for myself and I don't generally go back to them unless I want to look up the name of something. Although I never had the eureka moment with patterns than many others have described, the Gang of Four is a book I go back to again and again and really should be on my desert island book shelf.

Extreme Programming Explained: Embrace Change

by Kent Beck and Cynthia Andres, published by Addison Wesley

ISBN-10: 0321278658 ISBN-13: 978-0321278654

The final book is probably the most difficult book of all to choose. I've learnt so much from a few process and practice books including the *Pragmatic Programmer* (series) by Andrew Hunt and David Thomas, The *Practice of Programming* by Brian W. Kernighan and Rob Pike and *Lean Software Development* by Mary Poppendieck and Tom Poppendieck. They all have good common sense advice and I've learnt new stuff from them. However the book that has had the most impact on my career and the quality of my development skills was *Extreme Programming* by Kent Beck. It could be argued that a book about making teams work better together isn't much use when you're on your own on a desert island, but there is a lot that can be applied even by loan programmers.

The novel

Redemption Ark

by Alastair Reynolds, published by Gollancz; New Ed edition

ISBN-10: 0575073845 ISBN-13: 978-0575073845

I only discovered Alastair Reynolds about eighteen months ago and I'm hooked. I've always loved science fiction (and a little bit of fantasy). Reynolds' 'space opera' is dark, at times addictively complex and beats the likes of Arthur C. Clarke, Asimov and Tolkien hands down. I'd take all of his books given the chance, but *Redemption Ark* is the one I've enjoyed the most so far.

The albums

The Crimson Idol by WASP

Misplaced Childhood by Marillion

Choosing my favorite two albums from the hundreds (1000+) that I own was easy. Had I had to choose one I'd have had a serious problem (but it probably would have been *Misplaced Childhood*). I discovered both Marillion and WASP in my late teens and listened and still listen to these two albums over and over. Specifying just one thing that makes either of these albums great is near impossible. They both have superb lyrical content. Fish and Blackie Lawless excel themselves as both describe deep emotional pain they've been through. The guitar and drum work on *The Crimson Idol* is something else and includes the best guitar solo in the world. *Misplaced Childhood* incorporates superb guitar, keyboard and vocals and take you through emotional highs and lows. ■

Next issue: Jez Higgins picks his desert island books.

What's it all about?

Desert Island Disks is one of Radio 4's most popular and enduring programmes:

http://www.bbc.co.uk/radio4/factual/desertislanddiscs.shtml

The format is simple: each week a guest is invited to choose the eight records they would take with them to a desert island.

I've been thinking for a while that it would be entertaining to get ACCU members to choose their Desert Island Books. The format will be slightly different from the Radio 4 show. Members will choose about 5 books, one of which must be a novel, and up to two albums. The programming books must have made a big impact on their programming life or be ones that they would take to a desert island. The inclusion of a novel and a couple of albums will also help us to learn a little more about the person. The ACCU has some amazing personalities and I'm sure 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.

Code Critique Competition 50

Set and collated by Roger Orr.

A book prize is awarded for the best entry.

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

Last issue's code

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

As always, try to go beyond simply solving the initial problem. The code is shown in Listing 1.

Listing 1

```
// cmap.h
#include <map>
using namespace std;
// map that has a const operator[]
template <typename K, typename V>
class cmap : public map<K,V>
{
  typedef map<K,V> base;
public:
  using base::operator[];
  const V& operator[](K const &k) const
  {
    if ( count(k) == 0 )
    {
      static V v;
      return v;
    }
    return ((base)(*this))[k];
  }
};
// test_cmap.cpp
#include "cmap.h"
#include <iostream>
#include <string>
void test( cmap<int, string> & m, int idx )
{
  cout << "m[" << idx << "]="
    << m[ idx ] << endl;
}
void ctest( const cmap<int, string> & m,
            int idx )
{
  cout << "const m[" << idx << "]="
    << m[ idx ] << endl;
}
int main()
{
  cmap<int, string> map;
  map[0] = "Zero";
  map[1] = "One";
  map[2] = "Two";
  test( map, 2 );
  test( map, 3 );
  ctest( map, 2 );
  ctest( map, 3 );
}
//Example output when it goes wrong
// (const m[2] should be Two):
//m[2]=Two
//m[3]=
//const m[2]=
//const m[3]=
```

Critiques

From Chris Main <chris@cmain.entadsl.com>

When I first started using **std::map**, I too found it irritating that the notational simplicity of **operator[]** was unavailable for const maps. With more experience I have found that the need to check whether a key was found in a const map is more common than not.

In the common case, the STL design of returning an **iterator** is an elegant solution. There are, though, other cases where I think the proposed extension would be useful. It serves a similar purpose to a default branch of a **switch** statement. So I do not reject this attempt to extend the interface of an STL component out of hand.

std::map is a public base class of **cmap**. This is risky because the STL containers do not have any virtual functions, and in particular they do not have virtual destructors. **cmap** is therefore exposed to the dangers of slicing and memory leaks. The version submitted avoids those dangers by not overriding anything in the base class and by not having any member variables respectively. I think this is worth documenting in a comment.

The submitter has provided a failing test case, which is very helpful. The failing implementation has a C-style cast which casts away the const. There is no need to take such a brute force approach, as it can be implemented quite easily with the available const member functions:

```
const base::const_iterator i = find(k);
if (i == end()) {
    static V v;
    return v;
} else {
    return i->second;
}
```

This implementation is also more efficient than the original. It only searches the base class map once (in **find()**) rather than twice (in **count()** and **operator[]**).

The tests pass with this implementation. Because **test(map, 3)** has inserted a default value for the key 3, it is worth adding **ctest(map, 4)** to exercise the case when there is no value for a key passed to the const **operator[]**. This additional case also passes with the new implementation, so job done. Moral of the story: don't use casts when you don't need to.

But if you are as curious as I am, you will want to know why the original implementation didn't work. To get to the bottom of it, I split it up into its component parts:

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

```
//return ((base)(*this))[k];
base map((base)(*this));
const V &value = map[k];
return value;
```

In the debugger, I could see that the correct value was being assigned from `map[k]`, but it was being lost in the return statement. I re-wrote the code to use a pointer:

```
base *map((base *)(this));
const V &value = (*map)[k];
return value;
```

This worked OK. Nearly anything you can do with pointers you can do with references, so:

```
base &map((base &)(*this));
const V &value = map[k];
return value;
```

This also worked OK, so it finally became clear to me that the one line answer is:

```
return ((base &)(*this))[k];
```

The missing ampersand meant that the map was being copied into a temporary local variable which was destroyed when the function exited. The reference returned by the function was to a value in this destroyed variable, not to a value held in the `cmap`, and so it was invalid.

The submitter obviously understands that it is invalid to return a reference to a local variable from a function because a static variable has been used for the case when there is no value in the map for a particular key. The problem was that the use of casting obscured the fact that a reference to a local variable was being returned.

There are a couple of further modifications which may be worth considering. Using a static variable for the case when there is no value in the map for a key is fine in a single threaded application. If `cmap` needs to be used in a multi threaded application, a more useful implementation is to hold the default value as a member variable of `cmap`. That, though, means that `cmap` inheriting publicly from `std::map` is more of an issue. To be on the safe side, `cmap` should then inherit privately from `std::map` (or contain a `std::map` member variable) and provide forwarding functions for all the public member functions of `std::map`.

If `cmap` is modified to have the default value as a member variable, it may be useful to allow different default values from the default constructed value. For example, the string `"Not found"` or `"Undefined"` may be preferred to the empty string in some cases. This functionality could be provided by the constructor and/or an explicit setter:

```
cmap(const V &defaultValue = V());
set_default_value(const V &v = V());
```

From David Carter-Hitchin < david@carter-hitchin.clara.co.uk>

The first thing I notice is that there are no header include guards, i.e.

```
#ifndef CMAP_H_20071128_324234
#define CMAP_H_20071128_324234
... header stuff...
#endif
```

I always put the date and a random sequence of digits to be absolutely sure you don't mistakenly exclude another header with the same guard string. I was once told a story of a debugging session which was a total nightmare and was caused by duplicate header guard symbols.

Another no-no is `using namespace std` in the header. This is bad enough in cpp files but in headers it's evil. It means that you've got the entire `std` namespace even if you don't want all of it. This could actually be a show stopper for your customers, and even if you don't have customers, then it can still cause you problems. Best to fully qualify symbols in the header, e.g. `std::string`, but if that's too onerous then `using std::string;` is better, but can still cause problems.

The essential problem is this line in `cmap::operator[]`:

```
return ((base)(*this))[k];
```

This returns a const reference to some object as defined in the implementation of `std::map`, this object is not necessarily going to exist by the time the library function has done its job of looking up the key's value in the map, and returned control to the caller. In Visual Studio 2005, if I assign this to a string variable then I can see it's a BadPtr, presumably because it now points to memory which has been cleaned up. On *some* implementations you might find that this does return a decent address, but obviously you can't be sure. One solution is right before us, a few lines up, namely to create a static variable, assign the key's value to that and return it:

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

Another possibility, is to return a const copy of the object, rather than a reference:

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

This fix is for the lazy – a deletion of one character (the ampersand after the return value)! Of course, returning by copy could be expensive, so this may not be the best solution for all cases (perhaps this could then present a case for template specialisation, but that approach would require some maintenance). Other things to think about include inlining as the code is quite small and thread safety (adding a mutex to prevent multiple access).

Commentary

Code like this has, to me anyway, what Martin Fowler in *Refactoring* calls a 'bad smell'. My concern over this example, independent of whether there are known bugs in it, is that the writer has extended a class that is not designed to be base class. However, your mileage may vary as Stroustrup does something similar in *The C++ Programming Language* [p780 of 3rd edition] where he derives a class `Vector` from `std::vector`. The two main dangers with this are (as Chris points out) deleting via a pointer-to-base and inadvertent slicing when, for example, an object from the derived class is passed to a function taking a reference to the base class. Although in this case these dangers are missing they could easily be introduced by future code changes.

The 'interesting' side of the bug is that it is extremely hard to spot from the syntax – just a missing ampersand – and it also seems hard for compilers to detect the problem.

Two of us once spent nearly two days trying to find a problem very similar to this one where a cast created a temporary. What makes this sort of problem hard to track down is the dangling reference generated may remain valid some of the time – it depends on the specifics of the compiler, the runtime and the pattern of usage of memory.

Good advice in this particular case is to stop using C-style casts wherever possible. In the case shown the cast is being used to access the non-const `operator[]`. If a const_cast is used rather than the C-style cast then most compilers will reject the code: for example MSVC 8.0 gives the error message 'Conversion requires a constructor or user-defined-conversion operator, which can't be used by const_cast or reinterpret_cast'. I'm

waiting for an (optional) compiler switch to give me a warning for all C style casts.

I note that at least one compiler provides a warning for the construction of `static V v;` not being thread-safe. The good news is that the forthcoming C++0x standard should contain a standard way to ensure the construction is thread-safe. Interested readers can google for ISO C++ committee paper N2382: 'Dynamic Initialization and Destruction with Concurrency'. Less interested readers will probably ignore the problem – if they are lucky they may also escape being a victim of the problems that can ensue! Of course, even if the static is constructed in a thread-safe fashion its use will require that read access to the variable is thread-safe. Mutable instances of the `cmap` class will also require external synchronization if they are to be accessed by more than one thread. (Note that David's first solution is implicitly single-threaded as the `static V retVal` is shared among all users of the class.)

Finally there is a potential performance issue with the original code – it is searching the map once in `count()` and once in `operator[]`. One of the strengths of the C++ STL is the use of iterators in the API to the collection classes that allows code to avoid searching collections twice. There is a slight learning curve as the array-like access using `operator[]` is much more intuitive but practice in using `find()` gives confidence using the more efficient style. To me this is an example of avoid premature pessimisation when using C++, in the same way that many people use const references by default rather than copying objects.

The winner of CC 49

I thought Chris did a good job of both fixing the problem and explaining the steps he took to understand the underlying cause, so I have awarded him the prize.

Code critique 50

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

For a slight change here is a C# critique. Perhaps a different language will encourage some new readers to attempt their first entry? There are at least two major problems with the code (Listing 2) but, as always, try to help the writer help themselves.

I'm trying to sort a C# KeyedCollection generic class but I find sometimes the sorting seems to hang. Can you suggest what I'm doing wrong?

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

Listing 2

```
using System;
using System.Collections.ObjectModel;
using System.Reflection;
public class FieldCollection :
  KeyedCollection<object[],object> {
  private string[] fields;
  public FieldCollection(
    params string[] fields ) {
    this.fields = fields;
  }
  protected override object[] GetKeyForItem(
    object item ) {
    object[] keys =
      new object[ fields.Length ];
    for (int i=0; i<fields.Length; i++) {
      string field = fields[i];
      FieldInfo fieldInfo =
        item.GetType().GetField( field,
          BindingFlags.Instance |
          BindingFlags.NonPublic |
          BindingFlags.Public );
      keys[i] = fieldInfo.GetValue( item );
      }
    return keys;
  }
```

Listing 2 (cont'd)

```
  public void Sort() {
    bool sorted = false;
    while (!sorted) {
      sorted = true;
      for(int i=1;i<base.Count && sorted;i++){
        Collection<object> collection = this;
        object object1 = collection[i-1];
        object object2 = collection[i];
        object[] key1= GetKeyForItem(object1);
        object[] key2= GetKeyForItem(object2);
        for (int j=0; j<fields.Length; j++) {
           IComparable key =
           key1[j] as IComparable;
          if (key != null) {
            if (key.CompareTo(key2[j]) > 0) {
              base.RemoveAt(i);
              base.Insert(i-1, object2);
              sorted = false;
              break;
            }
          }
          else {
            throw new Exception();
          }
        }
      }
    }
  }
}
public class TestColl {
  private string firstname;
  private string lastname;
  public TestColl( string name ) {
    this.firstname = name.Split(' ')[0];
    this.lastname = name.Split(' ')[1];
  }
  public override string ToString() {
    return string.Format("{0} {1}",
      firstname, lastname);
  }
  public static void Main(string[] args) {
    try {
      test();
    }
    catch (System.Exception ex) {
      System.Console.WriteLine(ex);
    }
  }
  private static void test() {
    FieldCollection coll = new
      FieldCollection(new string[]
      {"lastname", "firstname"});
    coll.Add(new TestColl("Roger Orr"));
    coll.Add(new TestColl("Alan Griffiths"));
    coll.Add(new TestColl("Tim Penhey"));
    coll.Add(new TestColl("Kevlin Henney"));
    // Doesn't sort if I add this one:
    // coll.Add(new TestColl("Jez Higgins"));
    coll.Sort();
    foreach (TestColl test in coll) {
      System.Console.WriteLine(test);
    }
  }
}
```

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.

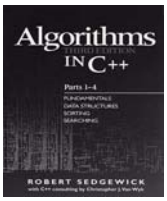
I must thank Blackwells and Computer Bookshop for their continued support in providing us with books.

C++

Algorithms in C++ (3rd Edition)

by Sedgewick and Van Wyk

Reviewed by Colin Paul Gloster



I have written this review as the ACCU reviewer for April 2002 was too kind. Like other authors, Robert Sedgewick seems to have become so respected that future editions are not denigrated for being inferior to his earlier editions. (The last time I read a book on algorithms by him in Pascal was in the previous century and I no longer have it so I can not reliably say whether an earlier edition truly was better.) This book is aimed at beginners to programming, but it is too dangerous for inexperienced programmers. For example, the following function

```
int factorial(int N)
{
   if (N == 0) return 1;
   return N*factorial(N-1);
}
```

is supposedly proven to be correct on the next page, but the supposed proof contained no restriction that N must be greater than or equal to zero. Someone who did not know through experience that too many programmers continue with bad habits for years learnt early on might forgive many deliberately error-prone techniques in the book (such as not using a vendor's implementation of the algorithms) because, as the authors themselves admitted, this book does not present dependable code for real use, merely education, but declaring N as int N instead of unsigned int N can not be explained away in this manner. The factorial function is not the only recursive function in the book which should have had this trivial defensive programming style. Even if a beginner uses this book despite using a different language, this would probably not be safe as that language would probably be Java which does not have an unsigned type. One reason to study algorithms is for efficiency, so an economical embedded product might use a small 8-bit processor whose stack could easily be corrupted by excessively deep recursion but unfortunately no warning of this was given.

I rarely have anything but contempt for diagrams, but I was actually impressed by many of the graphs the chapters on sorting as they genuinely do make good use of the medium to show characteristic differences between alternatives.

The topic is difficult enough already, and I suspect that oxymorons might serve more to confuse than motivate beginners. E.g. 'The study of algorithms is interesting because it is a new field (almost all the algorithms that we study are less than 50 years old, and some were just recently discovered) with a rich tradition (a few algo-rithms have been known for thousands of years)' and another example is: 'There are several reasons for studying these simple sorting algorithms [..] Second, these simple methods are actually more effective than the more powerful general-purpose methods in many applications of sorting. Third, several of the simple methods extend to better general-purpose methods or are useful in improving the efficiency of more sophisticated methods.'

This book was coauthored by Van Wyk as a C++ consultant. One wonders how an ordinary programmer can master several languages and attain a practical level of confidence with algorithms while a Princeton University professor of computer science needed a C++ consultant.

Sometimes, something is referred to which might not have been covered yet and it is not always clear (I wondered did I forget something or whether it was about to be shown in a few pages). Fairly harmless examples of this abound, e.g. Property 6.2 is mentioned five pages early, but the term 'sentinel' appears for the first time without explanation on page 229 and next appears on page 275 accompanied by a fairly unhelpful explanation (bearing in mind this is a beginners' book, this is bad).

I remember many not particularly clever classmates were utterly baffled by a perfect explanation of big-Oh notation (nothing by Sedgewick was used on the course). A textbook should provide a way for students to practise exercises so that they will finally understand. This textbook contains no answers so cannot be used (unless a lecturer provides answers) and the questions in the big-Oh section are too few and unvaried to help struggling students. (The book *Schaum's Outline of Theory and Problems of Essential Computer Mathematics* by Seymour Lipschutz published by McGraw-Hill inexcusably contains no big-Oh notation.)

The valid point that a fast computer running a slow algorithm is unlikely to compensate for a slow computer running a fast algorithm is made. The point that abstraction interferes with speed is made a number of times and should be heeded by supposed software engineers. E.g. 'When comparisons are expensive–for example, when the keys are strings–then insertion sort is much faster than each of selection sort and bubble sort. The blurb contains 'the implementations by Van Wyk and Sedgewick [..] exploit the natural match between C++ and ADT implementations' which

Bookshops

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

- **Computer Manuals** (0121 706 6000)
  www.computer-manuals.co.uk
- **Holborn Books Ltd** (020 7831 0022)
  www.holbornbooks.co.uk
- **Blackwell's Bookshop**, Oxford (01865 792792)
  blackwells.extra@blackwell.co.uk

was clearly written by someone else as this is contradicted on page 235.

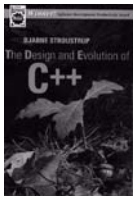
In the preface, all languages are treated as being equivalent for an algorithm. This is in contrast to the body of the book. High-level languages are disparaged on page 418 for awkwardness of bit banging; and on page 589 for needing explicitly casting of a string to an integer. The only languages recommended other than C++ are: 'machine-language' (machine code) on page 448; exploiting 'assembly or machine language' is actually recommended on page 332 instead of expending effort on further algorithmic improvements; and PostScript (which is the only 'Language, programming' pointed to by the index).

Not recommended for undergraduates who have studied for less than four semesters. Recommended with reservations for more experienced programmers.

The Design and Evolution of C++

By Bjarne Stroustrup, published by Addison-Wesley 1994, ISBN 0201543303

Reviewed by Colin Paul Gloster



This chronicle should be read by everyone involved in standardizing any language, regardless of one's opinion of C++. The lesson that a good, convincing theoretical argument can be used to advocate anything (even something bad) but support for even genuinely good ideas might be retracted after unpleasant experiences (e.g. 11.2.2 Ambiguity Control; 11.2.4 The overload Keyword; 13.9 Protected Members; and 16.6 Resumption vs. Termination) should be heeded. Stroustrup's tolerance of programmers who do not make much of an effort to improve in a timely manner is remarkable, and though this led to C++'s popularity, it has resulted in many flaws. The opposition to introducing reserved words which he documented in this book is not unique to C++.

A modicum of what Stroustrup had written in this book is at odds with better advice in Kernighan's and Pike's book *The Practice of Programming* but overall such discord is not representative of these books. A major influence on the unpleasant nature of C++ is that in order to gain popularity, it was based on a language whose creators opted for mutilation instead of consistency. Of the examples in the book on this theme, I am most fond of 'C experts' not being able to reach a consensus on how to interpret the consequences of an ANSI C rule for enumerations. Instead, Stroustrup seems to have been more struck by the following: 'I am still amazed, though, by the rule that accepts the result of any constant expression evaluat-ing to 0 as the null pointer. This rule makes 2-2 and ~-1 null pointers.' However, the book's most commonly repeated complaints of C++'s 'warts' from C are implicit int; 'irregularities in C's confusing variety of precedences'; 'messy anarchic conversions'; and syntax (not that all of C++'s unhelpful syntax can be blamed on C). Francis Glassborow claimed in his review of Kim Bruce's book *Foundations of Object-Oriented Languages*: 'Many of us recognize that within C++ there is a simpler and yet more powerful language trying to escape and this book [..] serves to confirm that feeling'. That also applies to this book.

Overall, Stroustrup's attitude seems to be fairly uniform throughout the book. However, perhaps germane to his guiding philosophy of not rigidly adhering to purists' idealism, one topic which he has not treated strictly uniformly should be reconsidered: i.e. his principle of trusting the programmers to not make mistakes. Stroustrup perceived 'idealists prone to ignore experience and experiment that inconveniently clashes with dogma' yet C++'s ability to have a drastic difference of meaning by mistyping a single character are legendary; it used to be common for an article in 'C/C++ Users Journal' to be followed the next month with a warning that its source code contained typos when that magazine used to be still in print (and *CVu* has had many typos, though maybe not in source code); all three printings of *The Design and Evolution of C++* have incorrect source code; other typos are present in the book; and he has documented mistakes with software (some his own) in e.g. 3.6.2 Members and Friends; 3.7 References; 11.3.3 Retrospective; 11.4.4 Memberwise Copy; page 273 ('The warnings didn't help – I even forgot my own rules and got caught'); and 13.9 Protected Members. I do not know of a human being who is not prone to perpetrate an accident due to habit. A disconcertingly large proportion of programmers are human. People are too reckless to drive a car safely and to vote sensibly so they cannot be expected to program computers attentively. Stroustrup may have been briefly aware of this as he correctly noted in 17.4.4 Using Namespaces to Manage Releases 'I might encourage users not to derive from my library classes in this way, but they'll do it anyway and complain about having to recompile even when they have been warned' which is almost exactly what happened with the FreeType library approximately ten years later.

WWW.research.ATT.com/~bs/dne_errata.html does not contain just errata: it has more notes which are worth reading. Even more notes can be found in the supplement at WWW.research.ATT.com/~bs/DnE2005.pdf during the long wait for the second edition.

Stroustrup has been justifiably pleased with the accuracy of predictions he has made with regards to C++, but parts of 7.4.3 Expectations and Attitudes do not seem to be accurate for Perl and Java, but neither are they systems programming languages.

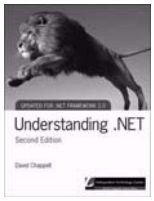
I am grateful to Dr. Jeremy A. Jones for recommending this book to me.

.NET

Understanding .NET, Second Edition

by David Chappell, published by Addison-Wesley, ISBN: 0321194047

Reviewed by Albrecht Fritzsche



Are you one of those always wondering what .NET actually means? What all these buzz words like ADO.NET, ASP.NET, etc stand for? How to get a first understanding of the capabilities of Microsoft's framework? Then this book might be the right choice for you – providing answers to all those questions and giving a very good overview in just 300 pages.

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

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

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

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

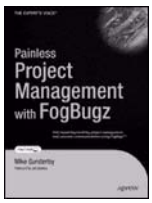
Recommended.

Miscellaneous

Painless Project Management with FogBugz

By Mike Gunderloy, published by Apress, ISBN-10: 159059486X ISBN-13: 978-1590594865

Reviewed by Tom Hughes



FogBugz, from Fog Creek Software, is a web based tool for bug tracking and project management based largely on the ideas of it's co-creator, Joel Spolsky (well known for his Joel on Software blog).

[continued on back page]

ACCU Information

Membership news and committee reports

View From The Chair

Jez Higgins
chair@accu.org

This time last year, I publicly resolved to get half of my View From The Chairs to Tim before the deadline. Didn't manage it. Wasn't even close.

Happily for me, I can avoid further enumerating my many failures and talk about the conference instead, because booking are now open for accu2008. As you may know, Giovanni Asproni took over as conference chair when Ewan stepped down after last year's conference. For his first conference he's secured some real heavyweights with this year's keynote speakers including process giant Tom Gilb and Haskell big-brain Simon Peyton Jones. Many other conference favourites return, including Andrei Alexandrescu, James Coplien, Kevlin Henney, and astonishingly energetic and generous John Lakos. No doubt the hotel will also be pleased to welcome Mr Lakos again.

Last year, two of my favourite sessions were given by debuting speakers, Ric Parkin and Richard Harris, and I'm looking forward to what they have this year. I'm sure that this year's new speakers will acquit themselves equally as well.

The approaching conference also means, of course, the approaching AGM. While you can get involved in the running of ACCU at pretty much anytime, the AGM is the obvious jumping on point. Is there something you feel ACCU should be or could be doing? If so, put yourself in a position to do something about it, and join the committee. As a volunteer run organisation ACCU cannot exist without the efforts of the committee, and there's only so much time that people have to give. I'm not asking for your sympathy, by the way, because I find it all quite good fun (with the exception of this letter which is intensely difficult). Of course, you needn't necessarily need to come whirling in filled with revolutionary zeal. I initially joined the committee on whim, if I'm honest, just to see what went on. I subsequently rationalised this by suggesting it's always useful to have people around who know what's going on and who can pitch in where necessary. By getting involved you can help shape the future of the organisation. Drop me a line if you'd like to discuss your idea or to find out more.

Membership Report

Mick Brooks
accumembership@accu.org

This time of year sees a peak in both the number of new members joining the association and the number of existing members whose renewals are becoming due. This is obviously linked to the extra awareness of ACCU generated by the annual conference, and, of course, the sign-up incentive provided by the member's discount on conference attendance rates. Now is a great time of year to mention your membership of ACCU to friends and colleagues, and to do your bit keep the peak healthy.

For those of you with expiry dates at the end of February, a reminder email will be being sent at about the same time that this magazine reaches you, and will contain full details of the renewal process. It describes the payment methods available (credit and debit cards are accepted via the website, or you can pay by standing order or cheque), how you can get an invoice or a receipt for your payment, and how you can resign your membership. (I'm always sad to lose a member, but am always happy to hear about the reasons for that decision, so I'd encourage you to let me know.) The email should explain things clearly, but don't hesitate to contact me if you have any questions or problems. Some of you will have opted out of receiving those messages, but you can of course contact me to request more information.

While you're renewing, I'll just remind you that it's also a good time to review the information that we store about you. You can get access to your mailing details etc. by logging into the website, where you can modify them if necessary. As always, don't hesitate to contact me if you have any questions.

Publicity Officer Report

David Carter-Hitchin
publicity@accu.org

The 2008 campaign is gathering momentum, but before I talk about that WHERE ARE THOSE SIGNATURES?! I couple of CVu's ago, I pleaded with you all to put a link to the ACCU in your e-mail signatures. Now some of you may have heard my plea, but many haven't as I can see from accu-general (mentioning no names). My signature reads, simply:

> ACCU - Professionalism in programming – http://www.accu.org/

So you know who you are, fire that e-mail client up and set your signature!

So, 2008. A generic publicity letter has been written now and a list of universities has been drawn up (there are about 150 univerisities and colleges out there in the UK alone). Now I need to get the names of the heads of department for the various departments (C.S., C. Graphics, Physics, Maths, Engineering and so on) and this consists of visiting each website and tracking the information down. It's a very laborious task, and if anyone could lend a hand then that would be great – please contact me, publicity@accu.org. Even if you could just manage a handful of sites that would be a great help. Equally important will be to target the students (thanks to Allan Kelly for reminding me of this). So if anyone knows of contact names for programming clubs or similar then please let me know. I think this information might be harder to find from websites so any 'inside contacts' would be appreciated. On another note, we have set up a reciprocal arrangement with the Code Generation folk, whereby ACCU members will have a 10% discount to their 2008 conference (Cambridge, UK from June 25th-27th, http://www.codegeneration.net/conference/index.php) in exchange for some marketing on our side. They have also put some links and ACCU logos on their site to ours. This kind of reciprocal agreement is very valuable to us and if anyone out there knows of other conferences which would be interested in similar arrangements then please contact me. For example, I contacted SD West 2008 (http://www.sdexpo.com/) but didn't get a response – maybe I can try from another angle. Finally, I have contacted BBC Radio Oxford to see if they'd be interested in doing something either before or during the conference.

The recent London meeting deserves a mention as this was very popular and will help raise the profile of the ACCU. The talk given was by Roger Orr on C++0x and was recorded by 7City Learning. The recording should go onto the ACCU website soon, for anyone who couldn't make it or those outside London. It's good to see that we're multimedia capable!

The 20th AGM

Alan Bellingham
secretary@accu.org

Notice is hereby given that the 20th Annual General Meeting of The C Users' Group (UK) publicly known as ACCU will be held during the lunchtime break on Saturday 5th April 2007 at the Paramount Oxford Hotel, Godstow Road Oxford, OX2 8AL, United Kingdom.

Current Agenda

1. Apologies for absence
2. Minutes of the 19th Annual General Meeting
3. Annual reports of the officers
4. Accounts for the year ending 31st December 2007
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. ■

Book Reviews (continued)

This book covers version 4.0 of FogBugz, although by the time I received it version 5 had already been out for some time, and version 6 is currently in the process of being released. A second edition with coverage of the new features in FogBugz 6 has recently been published.

The first chapter is a high level overview of FogBugz, including general information on the approach to bug tracking and project management favoured by FogBugz and a number of case studies showing how cases move through the system.

Chapter two provides a detailed description of how to get new cases into the system and the information that can be recorded for a case. It also shows how to work with lists of cases and use filters and sorting to slice and dice your way through the cases in the system.

The next chapter covers the customisation of FogBugz for a particular environment – how to setup lists of projects, versions and priorities as well as the security model and how to set up users and assign permissions to them.

Chapter four continues where chapter two left off, with the process of working on a case and using features such as time estimates, due dates and escalation reports to monitor progress. It finishes with coverage of resolving cases and using FogBugz to record changes and create release notes.

The final two chapters cover using FogBugz to communicate with customers (both on a one to one basis using its email integration features, and on a one to many basis using the builtin system of web based discussion groups) and integrating FogBugz with source code control systems to allow cases to be linked to related changes in the source.

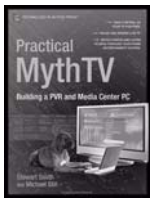
There are no obvious problems with the book, and it provides good coverage of the features of FogBugz and its philosophy of project management and bug tracking and I would certainly recommend it to anybody intending to use FogBugz and wanting to know more about it.

My only reservation would be that FogBugz is largely so simple to use that you probably don't really need a book, even if you're acting (as I do) as an administrator for FogBugz!

Practical MythTV

By Stewart Smith and Michael Still, published by Apress, ISBN: 1590597796

Reviewed by John Lear



MythTV, in case you haven't come across it, is an Open Source Personal Video Recorder (PVR) system. In addition to watching and recording television, it can be extended to include a digital music system, photo viewer, news and weather information, games and even an Internet phone. With the full set of plugins it can easily provide all the functionality required of a Home Media Server. However setting up such a system from scratch requires a lot of planning and thought. To provide the best user experience the correct mix of hardware as well as software must be combined and this is where this book endeavours to help out.

Practical MythTV covers setting up a system completely from scratch. The first two chapters cover the pre-requisites; hardware selection and supporting software. MythTV runs on Linux and as a consequence it is important to choose Linux friendly hardware. Hardware compatibility is less of an issue these days but the choice of Video Capture card is the most important. It was good to see that the book provides an in depth discussion of the factors to consider when choosing the hardware for your system. A step-by-step guide to installing and configuring Ubuntu and the requisite supporting packages then follows. In this chapter and throughout the rest if the book no previous knowledge of Linux is assumed. After installation, a small section details a number of steps that can verify hardware and that supporting software is working correctly before MythTV is installed.

Chapters 3 and 4 then cover the installation and the basics of recording TV. What I found most useful while progressing through the installation steps were the large and well-printed screenshots. This made it easy to understand what was being discussed in the accompanying text. These first four chapters are almost a third of the book (345 pages in total), which shows the importance and the amount of detail given over to setting up the basic system.

The remaining chapters describe advanced recording functionality, display themes, exporting recorded video to DVD, plus the additional plugins that can be added to the base system to extend functionality. Each chapter goes into the same level of detail and screenshots showing various on-screen displays.

One thing I would have liked to have seen was a troubleshooting section. With a large number of moving parts it can be tricky to track down where a problem lies and its resolution. A troubleshooting guide, even one that aided the user in simply tracking down the most likely module at fault would have been a welcome addition.

Conclusion

Practical MythTV gives a good step-by-step guide to building a PVR from scratch. It also not just a simple PVR either but able to compete and surpass some of the commercially available systems while being completely open. While some of the information is probably available on the Internet, Practical MythTV would be invaluable to resource to anyone building such a system from scratch for the first time.

Recommended