TUG*India* JOURNAL

Vol. 2 No. 1

INDIAN TEX users group

1 Who's Who

TUG*India*
JOURNAL

Vol. 2 No. 1

C. V. Rajagopal
University Observatory, Trivandrum
email: cvr@vsnl.com
P. Ramesh Kumar
School of Applicable Mathematics, M. G. Univ., Kottayam
A. Jayaram
Thomson Press (India) Ltd., Chennai
email: tpchenai@md2.vsnl.net.in

Editor:
K. S. S. Nambooripad

TUG*India* Journal is the quarterly publication of Indian TEX Users Group. The jounrnal will be distributed to all its members in electronic form (PDF format) once in every three months.

Typeset in TEX and PDF generated by PDFTEX with ConTEXt macros and PDFLATEX with hyperref.sty in an intel PII 350 MHZ system running Linux kernel ver. 2.0.36.

The miniature painting on the title page depicts the *gopis* adoring Krishna's footprints. Jaipur, *Rasapanchadhyayi* series, c. 1820, 21×14 cms, Courtsey: Ashok Kapur collection.

INDIAN TEX users group

⏪ ◁ ▷ ⏩ ●

2 Editorial

Apologies for the delayed release of the second volume. TUG*India* suffered paucity of human resources for the right upkeep of the journal and other intellectual pursuit, which it has been authorized to do. As you are aware, TUG*India* runs on the mercy of the resources shared by the volunteer members who have their own preoccupations at their work. However, we could gather few enterprising members who can spare little more time and effort for the cause of TUG*India* and as such we hope, future releases of the journal will not be unduly delayed.

The deviation you'll find in this issue is the key article, *The Cathedral and the Bazaar*, by Eric Raymond, who wrote the program *Fetchmail* and became one of the most vocal proponents of open source development. His article compares proprietary and open development methodologies. *The Cathedral and the Bazaar*, has become a manifesto for open source development, where a mob of programmers with little centralization yields finished software. Raymond says such a software is richer and more stable than any commercial software he has ever used. Though this article may seem irrelevant at least to some of our readers, I hope the message/philosophy outlined by Raymond assumes importance in the event of the controversies on *free* and *nonfree* directories at CTAN, raised by some of the

TUG*India*
JOURNAL

Vol. 2 No. 1

package mainteners who got agitated by the inclusion of their packages in the *nonfree* tree.

The other articles are *Book Design for TEX Users Part 2: Practice* by Philip Taylor, the first part of which appeared in the first issue of TUG*India* Journal and the last one is *Review of TUG98* by Kaveh Bazargan and Philip Taylor.

Hope that these articles will be of use to the readers. The PDF generation techniques employed in the current issue is a mix of ConTEXt for the first two articles and LATEX for the last one. Readers can have a comparison of the output generated by both the opposing document formatting technologies in the TEX world. I earnestly solicit your comments on quality, suggestions for improvement and all that.

K. S. S. NAMBOORIPAD

TUG*India*
JOURNAL
Vol. 2 No. 1

3 The Cathedral and the Bazaar

Eric S. Raymond
esr@snark.thyrsus.com

I anatomize a successful free-software project, *fetchmail*, that was run as a deliberate test of some surprising theories about software engineering suggested by the history of Linux. I discuss these theories in terms of two fundamentally different development styles, the "cathedral" model of FSF and its imitators versus the "bazaar" model of the Linux world. I show that these models derive from opposing assumptions about the nature of the software-debugging task. I then make a sustained argument from the Linux experience for the proposition that "Given enough eyeballs, all bugs are shallow", suggest productive analogies with other self-correcting systems of selfish agents, and conclude with some exploration of the implications of this insight for the future of software.[1]

[1] You may check for the latest version of this article at http://www.tuxedo.org/~esr/writings/cathedral-bazaar

TUG*India* JOURNAL
Vol. 2 No. 1

INDIAN TEX users group

TUG*India*
JOURNAL

Vol. 2 No. 1

INDIAN
TEX
users
group

Contents

3.1 The Cathedral and the Bazaar

Linux is subversive. Who would have thought even five years ago that a world-class operating system could coalesce as if by magic out of part-time hacking by several thousand developers scattered all over the planet, connected only by the tenuous strands of the Internet?

Certainly not I. By the time Linux swam onto my radar screen in early 1993, I had already been involved in Unix and free-software development for ten years. I was one of the first GNU contributors in the mid-1980s. I had released a good deal of free software onto the net, developing or co-developing several programs (nethack, Emacs VC and GUD modes, xlife, and others) that are still in wide use today. I thought I knew how it was done.

Linux overturned much of what I thought I knew. I had been preaching the Unix gospel of small tools, rapid prototyping and evolutionary programming for years. But I also believed there was a certain critical complexity above which a more centralized, a *priori* approach was required. I believed that the most important software (operating systems and really large tools like Emacs) needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time.

Linus Torvalds's style of development – release early and often, delegate everything you can, be open to the point of promiscuity – came as a surprise. No quiet, reverent cathedral-building here – rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who'd take submissions from *anyone*) out of which a coherent and stable system could seemingly emerge only by a succession of miracles.

The fact that this bazaar style seemed to work, and work well, came as a distinct shock. As I learned my way around, I worked hard not just at individual projects, but also at trying to understand why the Linux world not only didn't fly apart in confusion but seemed to go from strength to strength at a speed barely imaginable to cathedral-builders.

By mid-1996 I thought I was beginning to understand. Chance handed me a perfect way to test my theory, in the form of a free-software project which I could consciously try to run in the bazaar style. So I did – and it was a significant success.

In the rest of this article, I'll tell the story of that project, and I'll use it to propose some aphorisms about effective free-software development. Not all of these are things I first learned in the Linux world, but we'll see how the Linux world gives them particular point. If I'm correct, they'll help you understand exactly what it is that makes the Linux community such a fountain of good software – and help you become more productive yourself.

3.2 The Mail Must Get Through

Since 1993 I'd been running the technical side of a small free-access ISP called Chester County InterLink (CCIL) in West Chester, Pennsylvania (I co-founded CCIL and wrote our unique multiuser BBS software – you can check it out by telnetting to locke.ccil.org Today it supports almost three thousand users on nineteen lines.) The job allowed me 24-hour-a-day access to the net through CCIL's 56K line – in fact, it practically demanded it!

Accordingly, I had gotten quite used to instant Internet email. For complicated reasons, it was hard to get SLIP to work between my home machine (`snark.thyrsus.com`) and CCIL. When I finally succeeded, I found having to periodically telnet to `locke` to check my mail annoying. What I wanted was for my mail to be delivered on snark so that `biff(1)` would notify me when it arrived.

Simple sendmail forwarding wouldn't work, because snark isn't always on the net and doesn't have a static IP address. What I needed was a program that would reach out over my SLIP connection and pull across my mail to be delivered locally. I knew such things existed, and that most of them used a simple application protocol called POP (Post Office Protocol). And sure enough, there was already a POP3 server included with locke's BSD/OS operating system.

TUGIndia JOURNAL

Vol. 2 No. 1

I needed a POP3 client. So I went out on the net and found one. Actually, I found three or four. I used pop-perl for a while, but it was missing what seemed an obvious feature, the ability to hack the addresses on fetched mail so replies would work properly.

The problem was this: suppose someone named 'joe' on `locke` sent me mail. If I fetched the mail to `snark` and then tried to reply to it, my mailer would cheerfully try to ship it to a nonexistent 'joe' on `snark`. Hand-editing reply addresses to tack on '`@ccil.org`' quickly got to be a serious pain.

This was clearly something the computer ought to be doing for me. (In fact, according to RFC1123 section 5.2.18, `sendmail` ought to be doing it.) But none of the existing POP clients knew how! And this brings us to the first lesson:

- Every good work of software starts by scratching a developer's personal itch.

Perhaps this should have been obvious (it's long been proverbial that "Necessity is the mother of invention") but too often software developers spend their days grinding away for pay at programs they neither need nor love. But not in the Linux world – which may explain why the average quality of software originated in the Linux community is so high.

So, did I immediately launch into a furious whirl of coding up a brand-new POP3 client to compete with the existing ones? Not on your life! I looked carefully at the POP utilities I had

TUG*India*
JOURNAL

Vol. 2 No. 1

in hand, asking myself "which one is closest to what I want?". Because

- Good programmers know what to write. Great ones know what to rewrite (and reuse).

While I don't claim to be a great programmer, I try to imitate one. An important trait of the great ones is constructive laziness. They know that you get an A not for effort but for results, and that it's almost always easier to start from a good partial solution than from nothing at all.

Linus, for example, didn't actually try to write Linux from scratch. Instead, he started by reusing code and ideas from Minix, a tiny Unix-like OS for 386 machines. Eventually all the Minix code went away or was completely rewritten – but while it was there, it provided scaffolding for the infant that would eventually become Linux.

In the same spirit, I went looking for an existing POP utility that was reasonably well coded, to use as a development base.

The source-sharing tradition of the Unix world has always been friendly to code reuse (this is why the GNU project chose Unix as a base OS, in spite of serious reservations about the OS itself). The Linux world has taken this tradition nearly to its technological limit; it has terabytes of open sources generally available. So spending time looking for some else's almost-good-enough is more likely to give you good results in the Linux world than anywhere else.

TUG*India*
JOURNAL
Vol. 2 No. 1

And it did for me. With those I'd found earlier, my second search made up a total of nine candidates – `fetchpop`, `PopTart`, `get-mail`, `gwpop`, `pimp`, `pop-perl`, `popc`, `popmail` and `upop`. The one I first settled on was `fetchpop` by Seung-Hong Oh. I put my header-rewrite feature in it, and made various other improvements which the author accepted into his 1.9 release.

A few weeks later, though, I stumbled across the code for 'popclient' by Carl Harris, and found I had a problem. Though `fetchpop` had some good original ideas in it (such as its daemon mode), it could only handle POP3 and was rather amateurishly coded (Seung-Hong was a bright but inexperienced programmer, and both traits showed). Carl's code was better, quite professional and solid, but his program lacked several important and rather tricky-to-implement `fetchpop` features (including those I'd coded myself).

Stay or switch? If I switched, I'd be throwing away the coding I'd already done in exchange for a better development base.

A practical motive to switch was the presence of multiple-protocol support. POP3 is the most commonly used of the post-office server protocols, but not the only one. Fetchpop and the other competition didn't do POP2, RPOP, or APOP, and I was already having vague thoughts of perhaps adding IMAP (Internet Message Access Protocol, the most recently designed and most powerful post-office protocol) just for fun.

But I had a more theoretical reason to think switching might be as good an idea as well, something I learned long before Linux.

- "Plan to throw one away; you will, any how." (Fred Brooks, *The Mythical Man-Month*, Chapter 11).

Or, to put it another way, you often don't really understand the problem until after the first time you implement a solution. The second time, maybe you know enough to do it right. So if you want to get it right, be ready to start over at least once.

Well (I told myself) the changes to `fetchpop` had been my first try. So I switched.

After I sent my first set of popclient patches to Carl Harris on 25 June 1996, I found out that he had basically lost interest in popclient some time before. The code was a bit dusty, with minor bugs hanging out. I had many changes to make, and we quickly agreed that the logical thing for me to do was take over the program.

Without my actually noticing, the project had escalated. No longer was I just contemplating minor patches to an existing POP client. I took on maintaining an entire one, and there were ideas bubbling in my head that I knew would probably lead to major changes.

In a software culture that encourages code-sharing, this is a natural way for a project to evolve. I was acting out this:

- If you have the right attitude, interesting problems will find you.

But Carl Harris's attitude was even more important. He understood that

- When you lose interest in a program, your last duty to it is to hand it off to a competent successor.

Without ever having to discuss it, Carl and I knew we had a common goal of having the best solution out there. The only question for either of us was whether I could establish that I was a safe pair of hands. Once I did that, he acted with grace and dispatch. I hope I will act as well when it comes my turn.

3.3 The Importance of Having Users

And so I inherited popclient. Just as importantly, I inherited popclient's user base. Users are wonderful things to have, and not just because they demonstrate that you're serving a need, that you've done something right. Properly cultivated, they can become co-developers.

Another strength of the Unix tradition, and again one that Linux pushes to a happy extreme, is that a lot of users are hackers too – and because source code is available, they can be *effective* hackers. This can be tremendously useful for shortening debugging time. Given a bit of encouragement, your users will diagnose problems, suggest fixes, and help improve the code far more quickly than you could unaided.

- Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.

The power of this effect is easy to underestimate. In fact, pretty well all of us in the free-software world drastically underestimated how well it would scale up with number of users and against system complexity, until Linus showed us differently.

In fact, I think Linus's cleverest and most consequential hack was not the construction of the Linux kernel itself, but rather his invention of the Linux development model. When I expressed this opinion in his presence once, he smiled and quietly repeated something he has often said: "I'm basically a very lazy person who likes to get credit for things other people actually do." Lazy like a fox. Or, as Robert Heinlein might have said, too lazy to fail.

In retrospect, one precedent for the methods and success of Linux can be seen in the development of the GNU Emacs Lisp library and Lisp code archives. In contrast to the cathedral-building style of the Emacs C core and most other FSF tools, the evolution of the Lisp code pool was fluid and very user-driven. Ideas and prototype modes were often rewritten three or four times before reaching a stable final form. And loosely-coupled collaborations enabled by the Internet, a la Linux, were frequent.

Indeed, my own most successful single hack previous to fetchmail was probably Emacs VC mode, a Linux-like collaboration by email with three other people, only one of whom (Richard Stallman) I have met to this day. It was a front-end for SCCS, RCS and later CVS from within Emacs that offered "one-touch" version control operations. It evolved from a tiny, crude sccs.el mode somebody else had written. And the development of VC succeeded because, unlike Emacs itself, Emacs Lisp code could go through release/test/improve generations very quickly.

(One unexpected side-effect of FSF's policy of trying to legally bind code into the GPL is that it becomes procedurally harder for FSF to use the bazaar mode, since they believe they must get a copyright assignment for every individual contribution of more than twenty lines in order to immunize GPLed code from challenge under copyright law. Users of the BSD and MIT X Consortium licenses don't have this problem, since they're not trying to reserve rights that anyone might have an incentive to challenge.)

3.4 Release Early, Release Often

Early and frequent releases are a critical part of the Linux development model. Most developers (including me) used

TUGIndia
JOURNAL
Vol. 2 No. 1

to believe this was bad policy for larger than trivial projects, because early versions are almost by definition buggy versions and you don't want to wear out the patience of your users.

This belief reinforced the general commitment to a cathedral-building style of development. If the overriding objective was for users to see as few bugs as possible, why then you'd only release one every six months (or less often) and work like a dog on debugging between releases. The Emacs C core was developed this way. The Lisp library, in effect, was not – because there were active Lisp archives outside the FSF's control, where you could go to find new and development code versions independently of Emacs's release cycle.

The most important of these, the Ohio State elisp archive, anticipated the spirit and many of the features of today's big Linux archives. But few of us really thought very hard about what we were doing, or about what the very existence of that archive suggested about problems in FSF's cathedral-building development model. I made one serious attempt around 1992 to get a lot of the Ohio code formally merged into the official Emacs Lisp library. I ran into political trouble and was largely unsuccessful.

But by a year later, as Linux became widely visible, it was clear that something different and much healthier was going on there. Linus's open development policy was the very opposite of cathedral-building. The `sunsite` and `tsx-11` archives were burgeoning, multiple distributions were being floated.

TUGIndia
JOURNAL
Vol. 2 No. 1

And all of this was driven by an unheard-of frequency of core system releases.

Linus was treating his users as co-developers in the most effective possible way:

- Release early. Release often. And listen to your customers.

Linus's innovation wasn't so much in doing this (something like it had been Unix-world tradition for a long time), but in scaling it up to a level of intensity that matched the complexity of what he was developing. In those early times it wasn't unknown for him to release a new kernel more than once a *day*! And, because he cultivated his base of co-developers and leveraged the Internet for collaboration harder than anyone else, it worked.

But *how* did it work? And was it something I could duplicate, or did it rely on some unique genius of Linus's?

I didn't think so. Granted, Linus is a damn fine hacker (how many of us could engineer an entire production-quality operating system kernel?). But Linux didn't represent any awesome conceptual leap forward. Linus is not (or at least, not yet) an innovative genius of design in the way that, say, Richard Stallman or James Gosling are. Rather, Linus seems to me to be a genius of engineering, with a sixth sense for avoiding bugs and development dead-ends and a true knack for finding the minimum-effort path from point A to point B. Indeed, the whole design of Linux breathes this quality and mirrors Linus's essentially conservative and simplifying design approach.

So, if rapid releases and leveraging the Internet medium to the hilt were not accidents but integral parts of Linus's engineering-genius insight into the minimum-effort path, what was he maximizing? What was he cranking out of the machinery?

Put that way, the question answers itself. Linus was keeping his hacker/users constantly stimulated and rewarded – stimulated by the prospect of having an ego-satisfying piece of the action, rewarded by the sight of constant (even *daily*) improvement in their work.

Linus was directly aiming to maximize the number of person-hours thrown at debugging and development, even at the possible cost of instability in the code and user-base burnout if any serious bug proved intractable. Linus was behaving as though he believed something like this:

- Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone

Or, less formally, "Given enough eyeballs, all bugs are shallow." I dub this: "Linus's Law".

My original formulation was that every problem "will be transparent to somebody". Linus demurred that the person who understands and fixes the problem is not necessarily or even usually the person who first characterizes it. "Somebody finds the problem," he says, "and somebody *else* understands it. And

I'll go on record as saying that finding it is the bigger challenge." But the point is that both things tend to happen quickly.

Here, I think, is the core difference underlying the cathedral-builder and bazaar styles. In the cathedral-builder view of programming, bugs and development problems are tricky, insidious, deep phenomena. It takes months of scrutiny by a dedicated few to develop confidence that you've winkled them all out. Thus the long release intervals, and the inevitable disappointment when long-awaited releases are not perfect.

In the bazaar view, on the other hand, you assume that bugs are generally shallow phenomena – or, at least, that they turn shallow pretty quick when exposed to a thousand eager co-developers pounding on every single new release. Accordingly you release often in order to get more corrections, and as a beneficial side effect you have less to lose if an occasional botch gets out the door.

And that's it. That's enough. If "Linus's Law" is false, then any system as complex as the Linux kernel, being hacked over by as many hands as the Linux kernel, should at some point have collapsed under the weight of unforseen bad interactions and undiscovered "deep" bugs. If it's true, on the other hand, it is sufficient to explain Linux's relative lack of bugginess.

And maybe it shouldn't have been such a surprise, at that. Sociologists years ago discovered that the averaged opinion of a mass of equally expert (or equally ignorant) observers is quite a bit more reliable a predictor than that of a single

TUGIndia
JOURNAL
Vol. 2 No. 1

randomly-chosen one of the observers. They called this the "Delphi effect". It appears that what Linus has shown is that this applies even to debugging an operating system – that the Delphi effect can tame development complexity even at the complexity level of an OS kernel.

I am indebted to Jeff Dutky for pointing out that Linus's Law can be rephrased as "Debugging is parallelizable". Jeff observes that although debugging requires debuggers to communicate with some coordinating developer, it doesn't require significant coordination between debuggers. Thus it doesn't fall prey to the same quadratic complexity and management costs that make adding developers problematic.

In practice, the theoretical loss of efficiency due to duplication of work by debuggers almost never seems to be an issue in the Linux world. One effect of a "release early and often policy" is to minimize such duplication by propagating fed-back fixes quickly.

Brooks even made an off-hand observation related to Jeff's: "The total cost of maintaining a widely used program is typically 40 percent or more of the cost of developing it. Surprisingly this cost is strongly affected by the number of users. *More users find more bugs*." (my emphasis).

More users find more bugs because adding more users adds more different ways of stressing the program. This effect is amplified when the users are co-developers. Each one approaches the task of bug characterization with a slightly

TUG*India*
JOURNAL
Vol. 2 No. 1

different perceptual set and analytical toolkit, a different angle on the problem. The "Delphi effect" seems to work precisely because of this variation. In the specific context of debugging, the variation also tends to reduce duplication of effort.

So adding more beta-testers may not reduce the complexity of the current "deepest" bug from the *developer's* P.O.V., but it increases the probability that someone's toolkit will be matched to the problem in such a way that the bug is shallow *to that person*.

Linus coppers his bets, too. In case there *are* serious bugs, Linux kernel version are numbered in such a way that potential users can make a choice either to run the last version designated "stable" or to ride the cutting edge and risk bugs in order to get new features. This tactic is not yet formally imitated by most Linux hackers, but perhaps it should be; the fact that either choice are available makes both more attractive.

3.5 When Is A Rose Not A Rose?

Having studied Linus's behavior and formed a theory about why it was successful, I made a conscious decision to test this theory on my new (admittedly much less complex and ambitious) project.

But the first thing I did was reorganize and simplify popclient a lot. Carl Harris's implementation was very sound, but exhibited a kind of unnecessary complexity common to many C programmers. He treated the code as central and the data structures as support for the code. As a result, the code was beautiful but the data structure design ad-hoc and rather ugly (at least by the high standards of this old LISP hacker).

I had another purpose for rewriting besides improving the code and the data structure design, however. That was to evolve it into something I understood completely. It's no fun to be responsible for fixing bugs in a program you don't understand.

For the first month or so, then, I was simply following out the implications of Carl's basic design. The first serious change I made was to add IMAP support. I did this by reorganizing the protocol machines into a generic driver and three method tables (for POP2, POP3, and IMAP). This and the previous changes illustrate a general principle that's good for programmers to keep in mind, especially in languages like C that don't naturally do dynamic typing:

- Smart data structures and dumb code works a lot better than the other way around

Fred Brooks, Chapter 11 again: "Show me your code and conceal your data structures, and I shall continue to be mystified. Show me your data structures, and I won't usually need your code; it'll be obvious."

Actually, he said “flowcharts” and “tables”. But allowing for thirty years of terminological/cultural shift, it’s almost the same point.

At this point (early September 1996, about six weeks from zero) I started thinking that a name change might be in order – after all, it wasn’t just a POP client any more. But I hesitated, because there was as yet nothing genuinely new in the design. My version of popclient had yet to develop an identity of its own.

That changed, radically, when fetchmail learned how to forward fetched mail to the SMTP port. I’ll get to that in a moment. But first: I said above that I’d decided to use this project to test my theory about what Linus Torvalds had done right. How (you may well ask) did I do that? In these ways:

- I released early and often (almost never less often than every ten days; during periods of intense development, once a day).
- I grew my beta list by adding to it everyone who contacted me about fetchmail.
- I sent chatty announcements to the beta list whenever I released, encouraging people to participate.
- And I listened to my beta testers, polling them about design decisions and stroking them whenever they sent in patches and feedback.

TUGIndia JOURNAL

Vol. 2 No. 1

INDIAN TEX users group

The payoff from these simple measures was immediate. From the beginning of the project, I got bug reports of a quality most developers would kill for, often with good fixes attached. I got thoughtful criticism, I got fan mail, I got intelligent feature suggestions. Which leads to:

- If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource

One interesting measure of fetchmail's success is the sheer size of the project beta list, fetchmail-friends. At time of writing it has 249 members and is adding two or three a week.

Actually, as I revise in late May 1997 the list is beginning to lose members for an interesting reason. Several people have asked me to unsubscribe them because `fetchmail` is working so well for them that they no longer need to see the list traffic! Perhaps this is part of the normal life-cycle of a mature bazaar-style project.

3.6 Popclient becomes Fetchmail

The real turning point in the project was when Harry Hochheiser sent me his scratch code for forwarding mail to the client machine's SMTP port. I realized almost immediately that a

TUG*India* JOURNAL
Vol. 2 No. 1

reliable implementation of this feature would make all the other delivery modes next to obsolete.

For many weeks I had been tweaking `fetchmail` rather incrementally while feeling like the interface design was serviceable but grubby – inelegant and with too many exiguous options hanging out all over. The options to dump fetched mail to a mailbox file or standard output particularly bothered me, but I couldn't figure out why.

What I saw when I thought about SMTP forwarding was that popclient had been trying to do too many things. It had been designed to be both a mail transport agent (MTA) and a local delivery agent (MDA). With SMTP forwarding, it could get out of the MDA business and be a pure MTA, handing off mail to other programs for local delivery just as sendmail does.

Why mess with all the complexity of configuring a mail delivery agent or setting up lock-and-append on a mailbox when port 25 is almost guaranteed to be there on any platform with TCP/IP support in the first place? Especially when this means retrieved mail is guaranteed to look like normal sender-initiated SMTP mail, which is really what we want anyway.

There are several lessons here. First, this SMTP-forwarding idea was the biggest single payoff I got from consciously trying to emulate Linus's methods. A user gave me this terrific idea – all I had to do was understand the implications.

TUGIndia JOURNAL
Vol. 2 No. 1

- The next best thing to having good ide as is recognizing good ideas from your users. Sometimes the latter is better.

Interestingly enough, you will quickly find that if you are completely and self-deprecatingly truthful about how much you owe other people, the world at large will treat you like you did every bit of the invention yourself and are just being becomingly modest about your innate genius. We can all see how well this worked for Linus!

And after a very few weeks of running the project in the same spirit, I began to get similar praise not just from my users but from other people to whom the word leaked out. I stashed away some of that email; I'll look at it again sometime if I ever start wondering whether my life has been worthwhile :-).

But there are two more fundamental, non-political lessons here that are general to all kinds of design.

- Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong

I had been trying to solve the wrong problem by continuing to develop popclient as a combined MTA/MDA with all kinds of funky local delivery modes. Fetchmail's design needed to be rethought from the ground up as a pure MTA, a part of the normal SMTP-speaking Internet mail path.

When you hit a wall in development – when you find yourself hard put to think past the next patch – it's often time to ask not whether you've got the right answer, but whether you're asking the right question. Perhaps the problem needs to be reframed.

Well, I had reframed my problem. Clearly, the right thing to do was (1) hack SMTP forwarding support into the generic driver, (2) make it the default mode, and (3) eventually throw out all the other delivery modes, especially the deliver-to-file and deliver-to-standard-output options.

I hesitated over step 3 for some time, fearing to upset long-time popclient users dependent on the alternate delivery mechanisms. In theory, they could immediately switch to `.forward` files or their non-sendmail equivalents to get the same effects. In practice the transition might have been messy.

But when I did it, the benefits proved huge. The cruftiest parts of the driver code vanished. Configuration got radically simpler – no more grovelling around for the system MDA and user's mailbox, no more worries about whether the underlying OS supports file locking.

Also, the only way to lose mail vanished. If you specified delivery to a file and the disk got full, your mail got lost. This can't happen with SMTP forwarding because your SMTP listener won't return OK unless the message can be delivered or at least spooled for later delivery.

Also, performance improved (though not so you'd notice it in a single run). Another not insignificant benefit of this change was that the manual page got a lot simpler.

TUG*India* JOURNAL

Vol. 2 No. 1

Later, I had to bring delivery via a user-specified local MDA back in order to allow handling of some obscure situations involving dynamic SLIP. But I found a much simpler way to do it.

The moral? Don't hesitate to throw away superannuated features when you can do it without loss of effectiveness. Antoine de Saint-Exupery (who was an aviator and aircraft designer when he wasn't being the author of classic children's books) said:

- "Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away"

When your code is getting both better and simpler, that is when you *know* it's right. And in the process, the fetchmail design acquired an identity of its own, different from the ancestral popclient.

It was time for the name change. The new design looked much more like a dual of `sendmail` than the old popclient had; both are MTAs, but where sendmail pushes then delivers, the new popclient pulls then delivers. So, two months off the blocks, I renamed it `fetchmail`.

3.7 Fetchmail Grows Up

There I was with a neat and innovative design, code that I knew worked well because I used it every day, and a burgeoning beta list. It gradually dawned on me that I was no longer engaged in a trivial personal hack that might happen to be useful to few other people. I had my hands on a program every hacker with a Unix box and a SLIP/PPP mail connection really needs.

With the SMTP forwarding feature, it pulled far enough in front of the competition to potentially become a "category killer", one of those classic programs that fills its niche so competently that the alternatives are not just discarded but almost forgotten.

I think you can't really aim or plan for a result like this. You have to get pulled into it by design ideas so powerful that afterward the results just seem inevitable, natural, even foreordained. The only way to try for ideas like that is by having lots of ideas – or by having the engineering judgment to take other peoples' good ideas beyond where the originators thought they could go.

Andrew Tanenbaum had the original idea to build a simple native Unix for the 386, for use as a teaching tool. Linus Torvalds pushed the Minix concept further than Andrew probably thought it could go – and it grew into something wonderful. In the same way (though on a smaller scale), I took some ideas by Carl Harris and Harry Hochheiser and pushed them hard.

TUG*India* JOURNAL

Vol. 2 No. 1

Neither of us was 'original' in the romantic way people think is genius. But then, most science and engineering and software development isn't done by original genius, hacker mythology to the contrary.

The results were pretty heady stuff all the same – in fact, just the kind of success every hacker lives for! And they meant I would have to set my standards even higher. To make `fetchmail` as good as I now saw it could be, I'd have to write not just for my own needs, but also include and support features necessary to others but outside my orbit. And do that while keeping the program simple and robust.

The first and overwhelmingly most important feature I wrote after realizing this was multidrop support – the ability to fetch mail from mailboxes that had accumulated all mail for a group of users, and then route each piece of mail to its individual recipients.

I decided to add the multidrop support partly because some users were clamoring for it, but mostly because I thought it would shake bugs out of the single-drop code by forcing me to deal with addressing in full generality. And so it proved. Getting RFC822 parsing right took me a remarkably long time, not because any individual piece of it is hard but because it involved a pile of interdependent and fussy details.

But multidrop addressing turned out to be an excellent design decision as well. Here's how I knew:

TUGIndia JOURNAL

Vol. 2 No. 1

INDIAN TEX users group

- Any tool should be useful in the expected way, but a **great** tool lends itself to uses you never expected.

The unexpected use for multi-drop fetchmail is to run mailing lists with the list kept, and alias expansion done, on the client side of the SLIP/PPP connection. This means someone running a personal machine through an ISP account can manage a mailing list without continuing access to the ISP's alias files.

Another important change demanded by my beta testers was support for 8-bit MIME operation. This was pretty easy to do, because I had been careful to keep the code 8-bit clean. Not because I anticipated the demand for this feature, but rather in obedience to another rule:

- When writing gateway software of any kind, take pains to disturb the data stream as little as possible – and **never** throw away information unless the recipient forces you to!

Had I not obeyed this rule, 8-bit MIME support would have been difficult and buggy. As it was, all I had to do is read RFC 1652 and add a trivial bit of header-generation logic.

Some European users bugged me into adding an option to limit the number of messages retrieved per session (so they can control costs from their expensive phone networks). I resisted this for a long time, and I'm still not entirely happy about it. But if you're writing for the world, you have to listen to your customers – this doesn't change just because they're not paying you in money.

3.8 A Few More Lessons From Fetchmail

Before we go back to general software-engineering issues, there are a couple more specific lessons from the `fetchmail` experience to ponder.

The `rc` file syntax includes optional 'noise' keywords that are entirely ignored by the parser. The English-like syntax they allow is considerably more readable than the traditional terse keyword-value pairs you get when you strip them all out.

These started out as a late-night experiment when I noticed how much the `rc` file declarations were beginning to resemble an imperative minilanguage. (This is also why I changed the original popclient 'server' keyword to 'poll').

It seemed to me that trying to make that imperative minilanguage more like English might make it easier to use. Now, although I'm a convinced partisan of the "make it a language" school of design as exemplified by Emacs and HTML and many database engines, I am not normally a big fan of "English-like" syntaxes.

Traditionally programmers have tended to favor control syntaxes that are very precise and compact and have no redundancy at all. This is a cultural legacy from when computing resources were expensive, so parsing stages had to be as cheap and simple as possible. English, with about 50% redundancy, looked like a very inappropriate model then.

This is not my reason for fighting shy of English-like syntaxes; I mention it here only to demolish it. With cheap cycles and core, terseness should not be an end in itself. Nowadays it's more important for a language to be convenient for humans than to be cheap for the computer.

There are, however, good reasons to be wary. One is the complexity cost of the parsing stage – you don't want to raise that to the point where it's a significant source of bugs and user confusion in itself. Another is that trying to make a language syntax English-like often demands that the "English" it speaks be bent seriously out of shape, so much so that the superficial resemblance to natural language is as confusing as a traditional syntax would have been. (You see this in a lot of 4GLs and commercial database-query languages.)

The `fetchmail` control syntax seems to avoid these problems because the language domain is extremely restricted. It's nowhere near a general- purpose language; the things it says simply are not very complicated, so there's little potential for confusion in moving mentally between a tiny subset of English and the actual control language. I think there may be a wider lesson here:

- When your language is nowhere near Turing-complete, syntactic sugar can be your friend.

TUG*India* JOURNAL
Vol. 2 No. 1

Another lesson is about security by obscurity. Some `fetchmail` users asked me to change the software to store passwords encrypted in the rc file, so snoopers wouldn't be able to casually see them.

I didn't do it, because this doesn't actually add protection. Anyone who's acquired permissions to read your `rc` file will be able to run `fetchmail` as you anyway – and if it's your password they're after, they'd be able to rip the necessary decoder out of the `fetchmail` code itself to get it.

All `.fetchmailrc` password encryption would have done is give a false sense of security to people who don't think very hard. The general rule here is:

- A security system is only as secure as its secret. Beware of pseudo-secrets.

3.9 Necessary Preconditions for the Bazaar Style

Early reviewers and test audiences for this paper consistently raised questions about the preconditions for successful bazaar-style development, including both the qualifications of the project leader and the state of code at the time one goes public and starts to try to build a co-developer community.

It's fairly clear that one cannot code from the ground up in bazaar style. One can test, debug and improve in bazaar style, but it would be very hard to *originate* a project in bazaar mode. Linus didn't try it. I didn't either. Your nascent developer community needs to have something runnable and testable to play with.

When you start community-building, what you need to be able to present is a *plausible promise*. Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is convince potential co- developers that it can be evolved into something really neat in the foreseeable future.

Linux and `fetchmail` both went public with strong, attractive basic designs. Many people thinking about the bazaar model as I have presented it have correctly considered this critical, then jumped from it to the conclusion that a high degree of design intuition and cleverness in the project leader is indispensable.

But Linus got his design from Unix. I got mine initially from the ancestral popmail (though it would later change a great deal, much more proportionately speaking than has Linux). So does the leader/coordinator for a bazaar-style effort really have to have exceptional design talent, or can he get by on leveraging the design talent of others?

I think it is not critical that the coordinator be able to originate designs of exceptional brilliance, but it is absolutely critical that he/she be able to *recognize good design ideas from others*.

TUG*India*
JOURNAL
Vol. 2 No. 1

INDIAN
TEX
users
group

⏪ ◁ ▷ ⏩ ●

Both the Linux and fetchmail projects show evidence of this. Linus, while not (as previously discussed) a spectacularly original designer, has displayed a powerful knack for recognizing good design and integrating it into the Linux kernel. And I have already described how the single most powerful design idea in `fetchmail` (SMTP forwarding) came from somebody else.

Early audiences of this paper complimented me by suggesting that I am prone to undervalue design originality in bazaar projects because I have a lot of it myself, and therefore take it for granted. There may be some truth to this; design (as opposed to coding or debugging) is certainly my strongest skill.

But the problem with being clever and original in software design is that it gets to be a habit – you start reflexively making things cute and complicated when you should be keeping them robust and simple. I have had projects crash on me because I made this mistake, but I managed not to with `fetchmail`.

So I believe the `fetchmail` project succeeded partly because I restrained my tendency to be clever; this argues (at least) against design originality being essential for successful bazaar projects. And consider Linux. Suppose Linus Torvalds had been trying to pull off fundamental innovations in operating system design during the development; does it seem at all likely that the resulting kernel would be as stable and successful as what we have?

A certain base level of design and coding skill is required, of course, but I expect almost anybody seriously thinking of launching a bazaar effort will already be above that minimum. The free-software community's internal market in reputation exerts subtle pressure on people not to launch development efforts they're not competent to follow through on. So far this seems to have worked pretty well.

There is another kind of skill not normally associated with software development which I think is as important as design cleverness to bazaar projects – and it may be more important. A bazaar project coordinator or leader must have good people and communications skills.

This should be obvious. In order to build a development community, you need to attract people, interest them in what you're doing, and keep them happy about the amount of work they're doing. Technical sizzle will go a long way towards accomplishing this, but it's far from the whole story. The personality you project matters, too.

It is not a coincidence that Linus is a nice guy who makes people like him and want to help him. It's not a coincidence that I'm an energetic extrovert who enjoys working a crowd and has some of the delivery and instincts of a stand-up comic. To make the bazaar model work, it helps enormously if you have at least a little skill at charming people.

3.10 The Social Context of Free Software

It is truly written: the best hacks start out as personal solutions to the author's everyday problems, and spread because the problem turns out to be typical for a large class of users. This takes us back to the matter of rule 1, restated in a perhaps more useful way:

- To solve an interesting problem, start by finding a problem that is interesting to you.

So it was with Carl Harris and the ancestral popclient, and so with me and fetchmail. But this has been understood for a long time. The interesting point, the point that the histories of Linux and `fetchmail` seem to demand we focus on, is the next stage – the evolution of software in the presence of a large and active community of users and co-developers.

In *The Mythical Man-Month*, Fred Brooks observed that programmer time is not fungible; adding developers to a late software project makes it later. He argued that the complexity and communication costs of a project rise with the square of the number of developers, while work done only rises linearly. This claim has since become known as "Brook's Law" and is widely regarded as a truism. But if Brooks's Law were the whole picture, Linux would be impossible.

A few years later Gerald Weinberg's classic *The Psychology of Computer Programming* supplied what, in hindsight, we can

TUG*India*
JOURNAL
Vol. 2 No. 1

see as a vital correction to Brooks. In his discussion of "egoless programming", Weinberg observed that in shops where developers are not territorial about their code, and encourage other people to look for bugs and potential improvements in it, improvement happens dramatically faster than elsewhere.

Weinberg's choice of terminology has perhaps prevented his analysis from gaining the acceptance it deserved – one has to smile at the thought of describing Internet hackers as "egoless". But I think his argument looks more compelling today than ever.

The history of Unix should have prepared us for what we're learning from Linux (and what I've verified experimentally on a smaller scale by deliberately copying Linus's methods). That is, that while coding remains an essentially solitary activity, the really great hacks come from harnessing the attention and brainpower of entire communities. The developer who uses only his or her own brain in a closed project is going to fall behind the developer who knows how to create an open, evolutionary context in which bug-spotting and improvements get done by hundreds of people.

But the traditional Unix world was prevented from pushing this approach to the ultimate by several factors. One was the legal contraints of various licenses, trade secrets, and commercial interests. Another (in hindsight) was that the Internet wasn't yet good enough.

TUG*India*
JOURNAL
Vol. 2 No. 1

Before cheap Internet, there were some geographically compact communities where the culture encouraged Weinberg's "egoless" programming, and a developer could easily attract a lot of skilled kibitzers and co-developers. Bell Labs, the MIT AI Lab, UC Berkeley – these became the home of innovations that are legendary and still potent.

Linux was the first project to make a conscious and successful effort to use the entire *world* as its talent pool. I don't think it's a coincidence that the gestation period of Linux coincided with the birth of the World Wide Web, and that Linux left its infancy during the same period in 1993-1994 that saw the takeoff of the ISP industry and the explosion of mainstream interest in the Internet. Linus was the first person who learned how to play by the new rules that pervasive Internet made possible.

While cheap Internet was a necessary condition for the Linux model to evolve, I think it was not by itself a sufficient condition. Another vital factor was the development of a leadership style and set of cooperative customs that could allow developers to attract co- developers and get maximum leverage out of the medium.

But what is this leadership style and what are these customs? They cannot be based on power relationships – and even if they could be, leadership by coercion would not produce the results we see. Weinberg quotes the autobiography of the 19th century Russian anarchist Kropotkin's *Memoirs of a Revolutionist*) to good effect on this subject:

> "Having been brought up in a serf-owner's family, I entered active life, like all young men of my time, with a great deal of confidence in the necessity of commanding, ordering, scolding, punishing and the like. But when, at an early stage, I had to manage serious enterprises and to deal with free men, and when each mistake would lead at once to heavy consequences, I began to appreciate the difference between acting on the principle of command and discipline and acting on the principle of common understanding. The former works admirably in a military parade, but it is worth nothing where real life is concerned, and the aim can be achieved only through the severe effort of many converging wills."

The "severe effort of many converging wills" is precisely what a project like Linux requires – and the "principle of command" is effectively impossible to apply among volunteers in the anarchist's paradise we call the Internet. To operate and compete effectively, hackers who want to lead collaborative projects have to learn how to recruit and energize effective communities of interest in the mode vaguely suggested by Kropotkin's "principle of understanding". They must learn to use Linus's Law.

Earlier I referred to the "Delphi effect" as a possible explanation for Linus's Law. But more powerful analogies to adaptive systems in biology and economics also irresistably suggest

TUG*India*
JOURNAL

Vol. 2 No. 1

themselves. The Linux world behaves in many respects like a free market or an ecology, a collection of selfish agents attempting to maximize utility which in the process produces a self-correcting spontaneous order more elaborate and efficient than any amount of central planning could achieve. Here, then, is the place to seek the “principle of understanding”.

The “utility function” Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers. (One may call their motivation “altruistic”, but this ignores the fact that altruism is itself a form of ego satisfaction for the altruist). Voluntary cultures that work this way are not actually uncommon; one other in which I have long participated is science fiction fandom, which unlike hackerdom explicitly recognizes “egoboo” (the enhancement of one’s reputation among other fans) as the basic drive behind volunteer activity.

Linus, by successfully positioning himself as the gatekeeper of a project in which the development is mostly done by others, and nurturing interest in the project until it became self-sustaining, has shown an acute grasp of Kropotkin’s “principle of shared understanding”. This quasi-economic view of the Linux world enables us to see how that understanding is applied.

We may view Linus’s method as an way to create an efficient market in “egoboo” – to connect the selfishness of individual hackers as firmly as possible to difficult ends that can only be achieved by sustained cooperation. With the fetchmail project

TUG*India* JOURNAL

Vol. 2 No. 1

I have shown (albeit on a smaller scale) that his methods can be duplicated with good results. Perhaps I have even done it a bit more consciously and systematically than he.

Many people (especially those who politically distrust free markets) would expect a culture of self-directed egoists to be fragmented, territorial, wasteful, secretive, and hostile. But this expectation is clearly falsified by (to give just one example) the stunning variety, quality and depth of Linux documentation. It is a hallowed given that programmers *hate* documenting; how is it, then, that Linux hackers generate so much of it? Evidently Linux's free market in egoboo works better to produce virtuous, other-directed behavior than the massively-funded documentation shops of commercial software producers.

Both the fetchmail and Linux kernel projects show that by properly rewarding the egos of many other hackers, a strong developer/coordinator can use the Internet to capture the benefits of having lots of co-developers without having a project collapse into a chaotic gang-bang. So to Brooks's Law I counter-propose the following:

- Provided the development coordinator has a medium at least as good as the Internet, and knows how to lead wit hout coercion, many heads are inevitably better than one

I think the future of free software will increasingly belong to people who know how to play Linus's game, people who leave behind the cathedral and embrace the bazaar. This is

TUGIndia JOURNAL

Vol. 2 No. 1

INDIAN TeX users group

not to say that individual vision and brilliance will no longer matter; rather, I think that the cutting edge of free software will belong to people who start from individual vision and brilliance, then amplify it through the effective construction of voluntary communities of interest.

And perhaps not only the future of *free* software. No commercial developer can match the pool of talent the Linux community can bring to bear on a problem. Very few could afford even to hire the more than two hundred people who have contributed to `fetchmail`!

Perhaps in the end the free-software culture will triumph not because cooperation is morally right or software "hoarding" is morally wrong (assuming you believe the latter, which neither Linus nor I do), but simply because the commercial world cannot win an evolutionary arms race with free-software communities that can put orders of magnitude more skilled time into a problem.

3.11 Acknowledgements

This paper was improved by conversations with a large number of people who helped debug it. Particular thanks to Jeff Dutky who suggested the "debugging is parallelizable" formulation and helped developed the analysis that proceeds from it. Also to Nancy Lebovitz for her suggestion that I emulate Weinberg by quoting Kropotkin. Perceptive criticisms also came from

Joan Eslinger and Marty Franz of the General Technics list. Paul Eggert noticed the conflict between GPL and the bazaar model. I'm grateful to the members of PLUG, the Philadelphia Linux User's group, for providing the first test audience for the first public version of this paper. Finally, Linus Torvalds's comments were helpful and his early endorsement very encouraging.

3.12 For Further Reading

- I quoted several bits from Frederick P. Brooks's classic *The Mythical Man-Month* because, in many respects, his insights have yet to be improved upon. I heartily recommend the 25th Anniversary addition from Addison-Wesley (ISBN 0-201-83595-9), which adds his 1986 *No Silver Bullet* paper.

The new edition is wrapped up by an invaluable 20-years-later retrospective in which Brooks forthrightly admits to the few judgements in the original text which have not stood the test of time. I first read the retrospective after this paper was substantially complete, and was surprised to discover that Brooks attributes bazaar-like practices to Microsoft!

- Gerald P. Weinberg's *The Psychology Of Computer Programming* (New York, Van Nostrand Reinhold 1971) introduced the rather unfortunately-labeled concept of "egoless programming". While he was nowhere near the first person to realize the futility of the "principle of command", he was

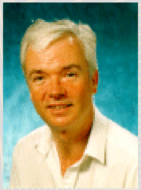
TUGIndia
JOURNAL
Vol. 2 No. 1

probably the first to recognize and argue the point in particular connection with software development.

- Richard P. Gabriel, contemplating the Unix culture of the pre-Linux era, argued for the superiority of a primitive bazaar-like model in his 1989 paper *Lisp: Good News, Bad News, and How To Win Big*. Though dated in some respects, this essay is still rightly celebrated among Lisp fans (including me). A correspondent reminded me that the section titled “Worse Is Better” reads almost as an anticipation of Linux.

- De Marco and Lister’s *Peopleware: Productive Projects and Teams* (New York; Dorset House, 1987; ISBN 0-932633-05-6) is an underappreciated gem which I was delighted to see Fred Brooks cite in his retrospective. While little of what the authors have to say is directly applicable to the Linux or free-software communities, the authors’ insight into the conditions necessary for creative work is acute and worthwhile for anyone attempting to import some of the bazaar model’s virtues into a more commercial context.

4 Book Design for TEX Users Part 2: Practice



Philip Taylor
p.taylor@vax.rhbnc.ac.uk

Abstract: In the predecessor to this paper[2], three fundamental concepts of *uniformity, information* and *structure* were introduced, and general guidance given on each of them. In this paper, more practical advice is given, specifically in two areas: guidance on actual dimensions, proportions and layout; and guidance on implementing some of the ideas through the medium of the TEX language. Finally, some difficult (and even insoluble) problems in layout are discussed.

Keywords: Design, typography, layout.

Every book will have one figure that cannot be seen from its point of reference.

[2]Book Design for TEX Users; Part I: Theory. See TUG*India* Journal Vol. 1, No. 1.

TUG*India* JOURNAL
Vol. 2 No. 1

Contents

TUG*India* JOURNAL

Vol. 2 No. 1

INDIAN TEX users group

4.1 How big is a book?

Just as we are all familiar with the general concept of a book, we are also familiar with practical upper- and lower-bounds on its size; a book that measures 3 centimetres by 2 centimetres is of as little use to most of us as a book measuring 3 metres by 2 metres. Looking at my bookshelves as I write, and ignoring only those volumes whose dimensions lie beyond the 3 σ points of the distribution, I can safely suggest that the majority of 'normal' books lie in the range 18 cm $\times$ 10 cm to 35 cm $\times$ 25 cm. In terms of more traditional printers' units (picas), we can re-express this range as 42 pc $\times$ 24 pc to 80 pc $\times$ 64 pc (in all cases I have approximated rather than taking any exact measurements). What is more interesting, however, is the aspect ratio of each these books: almost without exception they are in portrait orientation rather than landscape. Why should this be?

There are, I suggest, two answers to this; one intensely practical, the other slightly theoretical. The practical answer is easily demonstrated: take any book that is *not* in portrait orientation (*i.e.*, one that is in landscape orientation), hold it in one hand and attempt to open it: if the book is small, or tends to square rather than being overtly landscape, it will be reasonably stable in the hand, but if it is large, or markedly landscape in aspect ratio, it will tend to fold back on itself as the centres of gravity of the two halves fall outside the span of the opened hand. For certain classes of book (*i.e.*, those intended to be read from a desk or lectern, or perhaps opened on the reader's lap), this is of little consequence; but for those books which are most likely to be read whilst being held in the hand (which includes the vast majority of books published), such instability would render them almost unreadable, and therefore such combinations of size and aspect ratio are generally avoided.

The theoretical reason hearkens back to material covered in the predecessor to this paper, and is concerned with the optimal length of line. In that paper it was suggested that between 40 and 70 characters per line is the target, with the ideal somewhere near the upper end of that range. Given that most normally sighted people can read without difficulty 9 point to 12 point typefaces at the normal distance associated with reading books, but find anything much smaller somewhat difficult to read (and tend to regard anything much larger as 'insulting', in the sense that it appears to have been intended for children), this suggests that most books will tend to have a *measure* somewhere in the approximate range 12 picas to 30 picas, but will tend to cluster nearer the upper end of that range. When we compare this with the range of book sizes cited above, these figures seem reasonable; the smallest book encountered was 24 pc in width, measured across the cover, whilst the largest was 64 pc, similarly measured. Allowing for trimmed pages fitting comfortably inside the cover, and 'sensible' margins (as yet to be defined), we find that the smallest book has a measure of 17 pc whilst the largest has a measure of 48 pc (and is set in an

TUG*India*
JOURNAL
Vol. 2 No. 1

abnormally large font; it would be more usual to find a book of this size set in double-column format). Clearly there is a reasonable correspondence between theory and practice.

In practice, some sizes are more 'desirable' than others; traditionally, books were printed in a restricted range of sizes, and some of the terms used are still extant today; examples include 'quarto', 'folio', etc. Others, for example 'elephant' and 'royal' have fallen into disuse, and there is today far greater freedom in choosing the final size of a book. However, practical realities intrude here, as everywhere else, and ultimately the printer will have to produce the pages of the book by sub-dividing a much larger sheet of paper; as such large sheets of paper are produced in a fixed range of sizes, it is obvious that some final page sizes will result in much less wastage than others, and such sizes are therefore to be preferred; your printer will give you advice on 'ideal' page sizes if asked, and will almost certainly tell you if your preferred size leads to gross wastage.

In determining the dimensions of a book, there are essentially three variables: the overall area of the text, including headers and footers; the margins; and the trimmed dimensions of the final page. Clearly at most two of these can be arbitrarily determined, and the third must follow by the simple rules of arithmetic and geometry. In practice one tends (if given total freedom) to determine the final page size and the text area first, and then to calculate the margins based on the difference; but

TUGIndia JOURNAL
Vol. 2 No. 1

in so doing it is important to remember that the margins are just as important as every other element of the made-up page, and cannot simply have arbitrary size. 'Sufficient, but not too much' is an excellent axiom to bear in mind when determining the size of margins; for example, a small book whose trimmed width is 23 pc might have an outer margin of 3 pc and a measure of 17 pc; the actual inner margin will therefore also be 3 pc, but the *perceived* inner margin will be somewhat less, as some portion of it is taken up by the binding. In general, the thicker the book the greater the apparent loss of inner margin, but binding technique is even more significant, and a well bound thick book may lose less space on the inner margin than a poorly bound thin book.

As the overall dimensions of the book increase, so may the margins; but they do not increase in direct proportion to the increase in page size: rather, if anything, they increase quite slowly, perhaps in proportion to the square root of the increase in page size, or to its logarithm. Once again, 'sufficient but not too much' is the key.

So far we have concentrated on the inner and outer margins, and it is worth pointing out before considering the top and bottom margins that, if symmetric perceived margins are required, this inherently requires asymmetric actual margins; but the asymmetry alternates between verso and recto pages. That is, in order to allow for the binding loss, the right margin on the verso page and the left margin on the recto page must each be

TUG*India*
JOURNAL
Vol. 2 No. 1

increased by the binding loss. This is achieved automatically in the 'book' style of LaTeX, but plain TeX users will need a modified output routine. In order not to need any knowledge of the existing output routine, the following code hooks into the `\shipout` primitive, and can therefore be used in conjunction with *any* output routine, no matter how complex, unless it, too, adjusts `\hoffset` on the fly (in which case more sophisticated code would be required).

```
\newdimen \rectohoffset
\newdimen \versohoffset

\def \bindingloss {2 pc}
      %%% adjust to suit actual book
\let \Shipout = \shipout
      %%% need an alias so as to steal primitive
\let \then = \relax
      %%% just syntactic sugar (sorry, Kees!)

\rectohoffset = \hoffset
      \advance \rectohoffset by \bindingloss
\versohoffset = \hoffset
       \advance \versohoffset by -\bindingloss

\def \shipout
     {\ifodd \count 0
             %%% can't use \pageno in LaTeX
       \then
             \hoffset = \rectohoffset
        \else
             \hoffset = \versohoffset
```

TUG*India*
JOURNAL
Vol. 2 No. 1

```
\fi
\Shipout
}
```

Before considering actual dimensions for the vertical margins, it is worth considering the simpler question of proportion, and here, as in many elements of book design, two schools of thought obtain: the first would advocate that the top margin should be less than the bottom, the second just the converse! The argument in each case is based on visual balance: those who would place the text block asymmetrically towards the top of the page claim that, visually speaking, it 'sinks down under its own weight', whilst the alternative school claim that unless it is set asymmetrically towards the bottom of the page, it makes the page look top-heavy and therefore unstable. My own belief is that once the effects of head- and footlines are considered, the two schools can to a certain extent be reconciled; if, however, there are no head- and footlines, then my sympathies incline more towards the 'lower-is-better' school than towards its opponents.

The reason for considering the head- and footlines whilst discussing the margins is that whereas the left and right margins are what I will term 'simple' (that is, they each occupy a single band of white space), the top and bottom margins are effectively composite: there is white space above the headline, white space below the headline, and similarly white space above and below the footline (if present; if not, then the bottom margin is simple).

TUGIndia
JOURNAL
Vol. 2 No. 1

But in terms of visual density, the footline is usually very light—frequently no more than an unornamented page number—whilst the headline is frequently quite dense (see the predecessor to this paper for a fuller discussion on the possible contents of a headline). The effect of this is that the two lower margins are perceived by the eye/mind as being a single band of white space, whilst the two upper margins are perceived as separate entities. The eye/mind therefore takes the sum of the two bottom margins as representing the white space at the bottom of the page, whilst more or less ignoring the lower of the two upper margins and seeing only the upper component as representing white space.

We must now attempt to summarise the preceding discussion and to come up with some firm recommendations. In general the space above the headline is significantly greater than the space below, and is of the same order of magnitude as the mean of the left and right margins (assuming for the moment that these are not exaggerated; discussion on exaggerated margins occurs later in this section). The space below the headline is fairly small: perhaps 1 pica or thereabouts. At the bottom of the page, the situation is reversed: there is relatively little space above the footline, but rather more space below. But here caution must prevail: if we were to leave the same space above the footline as below the headline (e.g., 1 pica), we would overconstrain the page makeup process, for although any page could still run one line light, it could not run one line over without interfering with the footline (or, worse, displacing the footline vertically downwards); it is therefore necessary to leave additional white space above the footline on a normally made-up page, so that an overrun of a single line can be permitted *in extremis*. Thus a gap above the footline of perhaps 2 picas is appropriate, with an additional margin of 3 or 4 picas below. Bear in mind that these figures represent only a first-order approximation, but that only relatively small adjustments would be needed for a fairly wide variation in page size.

All the discussion on margins up to this point has reflected a fairly traditional, orthodox and conservative perspective. But the size and symmetry of margins is one of those areas in which *avant garde* designers feel obliged to express their individuality. Until the advent of the so-called 'DTP revolution', most books had conservative margins of the order of magnitude suggested above; but at about the time when DTP was becoming widespread, a new generation of designers suddenly found the need to adopt quite enormous margins, sometimes out of all proportion to the other material on the page.[3] The reasons for this sudden interest in wide margins are probably quite interesting, but I suspect not well understood. I can think of several possible reasons: (1) Each generation of designers feels obliged to express its creativity in some overt manner; simply to follow the guidance of its predecessors is felt at best to be pastiche, and at worst plagiarism. (2) The liberating effect of what I will term 'Design through DTP'[4] allowed designers to experiment with designs that might previously have been consigned to the

TUGIndia JOURNAL
Vol. 2 No. 1

dustbin, either because the wasteful nature of their extremes became only too apparent as real paper models were made of the design, or because the time which elapsed between the creation of a design and its first physical realisation allowed the designer time for retrospection; many, I am sure, toned down their own excesses during this cooling-off period. (3) Many of the realisations of these designs were accomplished using early DTP systems, which were themselves fairly limited in their page makeup ability; having large margins into which oversize elements could flow allowed the designers additional flexibility to work within the constraints of the DTP system.

But there is a fourth consideration, quite independent of the DTP revolution, which may also dictate the use of large margins, and this final discussion on margins concentrates solely on the page makeup problems associated therewith. Text, tables, graphics, equations and formul all have different, and sometimes conflicting, requirements—text, as we have seen, will normally fit best into a measure somewhere in the range

[3] It is a sad reflection of our times that this also occurred during a period when awareness of the ecological effects of the loss of the world's forests was becoming increasingly widespread; thus on the one hand we had the environmentalists urging us to save trees, whilst on the other we had a generation of designers apparently hell-bent on destroying the world's forests purely to provide large asymmetric white borders for their books. . .

[4] by which I mean the use of an Apple Macintosh or similar system to produce an on-screen mock-up of a proposed design without any need for a physical realisation to become available.

TUGIndia JOURNAL

Vol. 2 No. 1

INDIAN TEX users group

12 pc to 30 pc; tables possessing multiple columns may well not fit into such a restricted measure, a problem that also can affect complex graphics (which although generally scalable can become illegible if over-reduced); equations and formul may also require a measure well in excess of 30 pc if they are not to be split over more than one line. With the exception of equations and formul, the problems are not insoluble, or even difficult: where it is known in advance that a measure well in excess of 30 pc will be required, the text can be set in two columns whilst overwidth tables and graphics can be allowed to span both columns; as tables and graphics can be allowed to span both columns; as tables and graphics are generally regarded as 'floating' entities (that is, they can migrate in the text without causing the reader difficulty, as reference to them is almost invariable by name or by number rather than by implicit physical association), they can appear on a page in their own right, or at the top or bottom of the page on which they are referenced, without interrupting the flow of the text. But equations and formul (and similar entities, such as program fragments and algorithms) frequently *cannot* be allowed to float: the author will almost invariably write the text on the assumption that the equation/formula will always occur exactly where it does in the manuscript, and will simply allow his or her text to 'fall through' to the equation or formula; if such an equation/formula is overlong and cannot be wrapped, then both columns of the two-column text will need to be

TUGIndia
JOURNAL
Vol. 2 No. 1

Title Page
Who's who
Editorial
The Cathedral and the Bazaar
Book Design for TEX Users–II
A Review of TUG98
Quit

interrupted, to the great inconvenience of the reader, for it will not necessarily be at all apparent whether the text is to be read up to the equation/formula and then continued below in the same column, or the text is to be read up to the point of the equation/formula and then continued from the top of the next column. Worse, if the equation/formula occurs not in the first column of text but the second, as the reader progresses down the first column he/she is suddenly stopped dead in his/her tracks by a completely irrelevant equation/formula; not only does the reader now not know from where to continue, he/she also does not know why the interruption occurred in the first place. Only on reading down the second column does the reason for the interruption become clear.

Therefore, in such works, an alternative approach is required, and one such approach is the use of oversized margins: the text is set to a fairly wide single-column measure, but the trim dimensions of the page are such as to allow the longest equation or formula to extend out into the (usually right) margin as necessary. The designer is then faced with another problem: how to justify to the reader the presence of these margins on pages where no such equations or formul occur. It is by no means unusual to find section heads pushed out into the margins in these circumstances, nor to find marginal notes which might otherwise occur as foot- or even endnotes. Anything which can justify the presence of the anomalous margin is regarded as fair game!

Finally gutters: the internal 'margins' that separate columns from each other in multi-column formats. Generally speaking, a gutter should be no wider than the mean of the left and right margins; if anything, it can be somewhat narrower. Some designers prefer to divide their gutters vertically by a narrow rule; I would tend to avoid this unless rules were used elsewhere in the design. Here, as in many places, the desire for uniformity provides excellent guidance.

4.2 The elements of a book

Having established guidelines for the overall dimensions of our book, it is now appropriate to consider the various elements which make up that book. At the most superficial level (and ignoring the covers, spine and dustjacket), a book consists of the *front matter* (also referred to as 'prelims'), the *text*, and the *back matter* or *end matter* (the last is clearly ambiguous, as a book has two ends, but traditionally 'end matter' is used in preference to the less ambiguous 'back matter').

The front matter is composed of such elements as the half and full title pages; the copyright and cataloguing-in-publication data page; a table of contents (and sometimes other analogous tables); and perhaps a preface. Also frequently included in the front matter (particularly with the advent of the DTP revolution, since which we have all become far more aware of typefaces and typography in general) is a 'colophon', which strictly speaking should occur as the very last element of the book, but now more usually occupies space on the copyright and cataloguing-in-publication page; the colophon contains details of the typefaces and leading used, and may also give details of designer, printer, etc.

Amongst the end matter are found appendices; one or more indexes; a bibliography (if such is not associated with each chapter, or if an overall bibliography is desired as well as one per chapter); and perhaps a glossary or similar.

Finally, the text is composed of the body of the book; usually divided into chapters, it may also be divided at a higher level into parts.

It is fair to say that the boundaries between these three zones are not entirely rigid: an author may choose to regard a preface as a part of the text, rather than as a part of the front matter, and this will need to be reflected in the page numbering, as we shall see. Similarly some writers may regard their appendices as forming a part of the text; this may affect their page numbering but is less likely so to do. Indeed, an author may choose to write a preface, a prologue, an introduction, a conclusion, an epilogue, and one or more appendices; the designer and author will need to liaise carefully to ensure that each is appropriately classified.

The primary reason for this division concerns page numbering: front matter is traditionally numbered in *roman* style, using lower-case roman numerals (i, v, x, l, c, d, m) which are often set as dropped folios, whilst the text proper is usually numbered using *arabic* numerals (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Appendices and other end matter usually continue in the same sequence and style as the main text, but it is permissible to re-start the numbering for the appendices and prefix the page number with a letter 'A'; if this latter course is taken, the index (assuming that the index forms the very last element of the end matter) will need to have unnumbered pages, as it would clearly be inappropriate to continue using 'A'-style numbering whilst it would be equally inappropriate to resume the main numbering scheme. Fortunately indexes are not required to be self-referential (although I confess to once padding out an index that would otherwise not balance with an entirely spurious reference to 'loop, infinite', whose sole page number was that of the entry for 'loop, infinite' in the index. . .).

There are also conventions as to which elements are required to occur recto, which verso, and which require to be preceded or followed by a blank page. A typical book might be numbered as followed (remember that odd numbers indicate recto, whilst even numbers indicate verso):

- half title;
- blank;
- full title;

TUGIndia JOURNAL

Vol. 2 No. 1

INDIAN TEX users group

- cataloguing-in-publication, copyright, colophon;
- preface to the edition;
- general preface;
- ditto, continued;
- blank;
- table of contents;
- ditto, continued;
- glossary;
- blank;
- first chapter.

Of these, the half and full titles are required to occur recto, (whence the blank page between, which also affords a nice contrast to the complexity of the full title page); the copyright and c-i-p page frequently occurs on the reverse of the full title page; the preface is not required to start recto, but it may be the designer's wish that it should so appear; the table of contents is normally recto, as here; the first chapter invariably opens recto, and except in the most casual of styles all subsequent chapters must open recto as well. The page number of the first chapter

TUG*India* JOURNAL

Vol. 2 No. 1

page could equally well have been ‘13’; it is a design decision as to whether to continue the numbering sequence from the prelims or whether to start afresh with the main text.

There are fewer conventions concerning the end matter, but it would be normal for the *first* appendix to start recto; subsequent appendices may start recto or verso as necessary; and the index would also normally start recto.

4.3 Laying out the pages

Although by far the majority of pages in a book are ‘normal’ pages, it makes a certain amount of sense to start by considering the opening chapter pages, since these contribute a great deal to the book’s visual identity and allow a fair degree of artistic licence in their creation. (It is also fair to say that one can waste an enormous amount of time trying to design them!)

When designing one’s first book, it is by no means uncommon for people to align the main chapter header (be it ‘Chapter 1’ or ‘Introduction’) with the top of a normal page. For some books, particularly those with very short (less than two pages) chapters, this makes enormous sense, for otherwise one can run to far more pages than are strictly necessary (there are also sthetic reasons why such a design is to be preferred in these circumstances). However, the vast majority of books have chapters whose page count often runs into double figures, and for such books it is customary (although not essential) to

TUGIndia JOURNAL
Vol. 2 No. 1

INDIAN TEX users group

start the opening chapter heading some way down the page. Typically a quarter to a third of the page depth may be reserved for the above-heading space.

There next comes the question of what to put in the heading. If chapters are numbered, one has to decide between 'Chapter 1', 'One', '1' or some similar variant; and if named, whether to also number or just to use the name (and if one uses both names and numbers, then which numbering style to use). It is thought that 'Chapter 1' is a little old-fashioned, but I do not hold to this view. If both numbers and names are used, and if just the arabic number is chosen, then there is also the option of placing the two on the same line, perhaps separated by a colon and the space of the line; if they are put on separate lines, then it is customary for the number line to precede the name line.

Next the question of font: in which font(s) are these headings to appear? In almost all cases, a large bold font will be used, but 'large' is very much in the eye of the beholder; it is probably safe to say that LaTeX uses rather larger fonts for this purpose than more conservative designers might choose. The use of a *sans serif* font for such headings is most certainly justifiable, but not essential.

Placement: should the headings be centered or ranged left (or even ranged right)? Generally speaking, centered headings are either slightly old-fashioned or are more suitable for works in the arts; modern scientific publications frequently adopt a ranged-left theme which runs throughout the book, including headings such as these. Ranged-right probably shrieks *avant garde*, but cannot be discounted on that score; if used, there should probably be other elements in the design which echo the ranged-right theme, or there should be a contrasting ranged-left theme to balance. If an epigram is used, it is probably better to have the headings ranged left and the epigram ranged right, as the converse would over-emphasise the epigram to the detriment of the chapter title.

There is another element to placement which also requires discussion: is the white space above the heading to be regarded as belonging to the heading or to the page? By this I mean the following: if the chapter title normally occupies n lines (typically one or two), but a pathologically long title for a particular chapter requires one or more additional lines, from where should the space for these lines be taken? Should the title be allowed to extend *up* the page, encroaching on the reserved white space, or *down* the page, displacing the starting point of the main text downwards? Neither is ideal, but if authors insist on writing pathologically long titles, one or other solution must be taken. Although the following is not cast in stone, it is perhaps worthy of consideration: if the opening chapter page starts with a line containing only the *number* of the chapter (or with the word 'Chapter' followed by the number), then that should *always* occur in the same vertical position (and thus the main text will get displaced downwards); but if the page starts with the *title* of the chapter, then that title may be allowed to extend upwards, thereby ensuring that the main text always starts at exactly the vertical position on the page.

And rules: should the headings be set off from the text by a horizontal rule? Here we probably need to return to the theme of uniformity: if rules form a recurring theme throughout the book, then a rule between heading and text is probably fine; if not, then it may seem intrusive.

Finally, before leaving the subject of opening chapter pages completely, it might be worth recapitulating on the advice given in the predecessor to this paper concerning running heads and folios: generally speaking, a running head has no place on an opening chapter page; the white space above the title should merge imperceptibly into the top margin. This means that the folio, if normally on the outer edge of the running head, must (on an opening chapter page) either be omitted completely, or must be relegated to the footline. Omitting the folio is highly undesirable, as it renders the table of contents virtually useless (and also reduces the usefulness of the index, if any entry in the index refers to an opening chapter page); the solution is therefore to set the page number as a dropped folio, centered in the footline. Sometimes such folios are given a little additional ornamentation, for example en-dashes on each side set off by a thin space; although this convention is taken directly from typewriter practice it does, in the opinion of the present author, render the folio a little more obvious, and therefore has something to commend it.

Having completed opening chapter pages, the next most significant element in the design of the book is the normal text page; such pages usually make up over 90% of the book, and it is therefore worth expending considerable effort ensuring that they look 'right'. We have already dealt with margins, gutters, head and footlines, so we may concentrate on the text proper, and in particular on the fonts and leading to be used.

4.4 Fonts and leading

As suggested above, the text will normally be set in a 10 pt *serif* font, often on a 12 pt leading (here, at least, plain TEX gives sensible defaults, except in the excessive measure used). There appears to be a widespread belief that Times Roman is the font of choice, yet this font, designed as it was for use in the exceptionally narrow measure of newspaper columns, has little to commend it apart from widespread availability. The font is too narrow for the generous measure of most books, and if it *must* be used can benefit enormously from being anamorphically scaled by a factor of 24/25 in the vertical direction. Such scaling, whilst anathema to purists, converts the somewhat narrow letterforms of Times Roman into rounder, softer, shapes, and enables a near optimal combination of font size and leading to be used on measures up to 27 pc and beyond. 11/12.5 Times Roman, when anamorphically scaled by a factor

24/25, yields 10.56/12 which in the opinion of the present author results in a highly readable text.

But far better than anamorphically scaling Times Roman is to select a font which already has the appropriate properties (rounded letterforms, suitability for use with wide measures, etc.); examples are legion, but amongst the most obvious candidates are Baskerville, Bembo, Caslon, Garamond, and Palatino. To be avoided are fonts which are highly idiosyncratic: it is to be remembered that the *sole* purpose of the font is to convey information; if the reader is distracted by the idiosyncratic nature of the font, information transfer will be less than optimal and the book's value reduced as a result.

It may be worth digressing at this stage to discuss briefly one particular book which I first encountered on being asked to review it, Knuth's *3:16 Bible Texts Illuminated*. My first reaction on opening this book was to ask myself rhetorically "why on earth did he set it in Computer Modern?". I was familiar with Computer Modern from the *Computers and Typesetting* quintology, and had, of course, set much of my own material in Computer Modern whilst learning about TEX; but I had reached the point where I felt that other fonts had much more to offer, and had not, for some time, typeset anything in Computer Modern at all; it therefore came as a nasty shock to find a book on Bible Study typeset entirely in Computer Modern, particularly by someone whose opinions I value so greatly.

TUGIndia
JOURNAL

Vol. 2 No. 1

And yet, the strange thing is that having read no more than half a dozen pages of *3:16* I suddenly discovered that I was no longer seeing the font at all; it had, to all intents and purposes, ceased to exist as a typeface, and become purely a medium for the communication of facts. Now Computer Modern, based as it is on Monotype 8a, is not everyone's ideal font; and particularly when rendered on low resolution devices such as laser printers can be quite unpleasant indeed, with the thin strokes breaking up or disappearing completely and the thick strokes somehow seeming out of proportion. Yet when rendered on a high resolution typesetter, the contrast between thick and thin contributes much to the sthetics of the font, and the overall effect is to yield an unintrusive design, pleasantly devoid of idiosyncrasies, which suppresses its own personality and allows the information to shine through. Perhaps there is no such thing as a bad font; what we perceive as bad may simply be a good font used inappropriately, or rendered using inappropriate technology.

But to return to the question of design, and in particular to the design of the normal text pages of a book. Having selected our primary font and leading, we will need to select appropriate variants of that font for particular purposes (we may also need to select one or more other fonts for special purposes, but as a general rule the fewer fonts used in a document, the better the document will be). For emphasis, and for foreign words and phrases within the text, it is customary to use an italic variant of the font; the use of bold for emphasis is to be strongly deprecated, with such fonts being reserved for headers and similar. Italics may also be used for book titles, for the names of ships, and for other analogous entities. It goes without saying that underlining, too, has no place in the running text of a book, and very little place anywhere else either; just as the use of bold for emphasis is an artifact of early word-processing systems (which were incapable of italics and therefore had to create an alternative convention for achieving stress), underlining is an artifact of handwritten and typewritten text, and has no place in a typeset document.[5]

If it is necessary to stress a word or phrase within a longer structure that is already being typeset in italics, it is customary to revert to a roman font for the stressed section; but the present author can find no reason why in these circumstances the stressed section should not be set in bold italics, if such a font variant is available (and with the advent of PostScript fonts, such variants are usually to be found); if the bold stressed section is being compared or contrasted with another section

[5] Of course, like almost every rule, these rules too admit of exceptions, and it would be a brave author indeed who wrote that *every* instance of underlining, or of the use of bold within running text for emphasis, was categorically wrong; the most that can be said is that generally speaking such (ab)uses are regarded as infelicitous or inappropriate, and that should the designer none the less decide to adopt such a convention, he or she should be aware of the 'rules' that are being flouted, and take a conscious decision to flout them rather than simply being unaware of their existence.

TUG*India*
JOURNAL

Vol. 2 No. 1

of text in the book which is physically nearby, then it may be necessary to set that section too in bold italics, even if it occurs in a context in which italics are *not* being used; in that way, the reader will be given appropriate typographic cues as to which two sections are being compared or contrasted.

Italics (which are a highly stylised variant of a font) should not be confused with *slanted* or *oblique* variants, both of which involve no original design but result from a simple geometric transformation of the roman form of the font. Whereas italics and oblique forms both have an honourable ancestry (oblique normally being reserved for *sans serif* fonts whilst italics are normally a variant of a *serif* form), slanted fonts appear to be another artifact of the DTP revolution. In the opinion of the present author they have little to offer in the way of sthetics, and even though they are sometimes used where it is deemed desirable to differentiate typographically between two entities which would otherwise both have to be rendered in italics, as a general rule I would caution against their use. Designers have managed for centuries to convey considerable amounts of information without having recourse to slanted fonts; it is to be hoped that future generations of designers will conclude that they represent no more than what Fowler might have termed 'elegant variation', and are therefore a luxury without which we can all happily do.

It is sometimes necessary, particularly in books on linguistics or other subjects in which language is both used and discussed,

TUG*India* JOURNAL

Vol. 2 No. 1

to differentiate typographically between the two uses. Sometimes simple quotations marks will suffice; sometimes italics; but there are also times when both of those forms are already reserved for other typographic differentiation, and some third form is needed to clarify which text is being discussed and which text is performing the discussion. In these circumstances (and in very few others), it is justifiable to introduce a new font which may be used as a part of the running text. If the main text is set in a *serif* font (as it almost invariably will be), then a second *serif* font would *not* be suitable; even though two *serif* faces may be as different as chalk and cheese, the risk of confusion is still too great (and the sthetic clash too severe) to permit two distinct *serif* faces to appear in juxtaposition. The second font must therefore be a *sans serif* face, chosen to blend in with, whilst being clearly differentiable from, the main text face. The second font will need to be matched for weight (visual density), ex-height and caps-height; and because of the variation in the semantics of *design size*, will probably need to be loaded at a fractional size.

4.5 Headings

The motto for the predecessor to this paper was “There can never be too little space below headings, only too much!”, and in those few words can be summarised the bulk of the received wisdom concerning headings. As previously pointed out, a

TUG*India*
JOURNAL

Vol. 2 No. 1

heading *must* be tied to the text with which it is associated, and that text is invariably the text which immediately follows. Headings are frequently hierarchical in nature, and lower-level headings are more closely bound to the following material than higher-level; thus the white space which separates low-level headings from the text is usually less (and never more) than the white space which separates higher-level headings and text. In the limiting case, the heading is *run in*, that is to say literally forms a part of the text and does not occupy a line in its own right. For run-in headings, it is essential that the author be consistent in usage, since such headings can either participate in the grammar of the text or remain a distinct grammatical entity; in the former case it is customary to indicate the extent of the heading by a change of font (italics, or bold, or even caps and small caps), but by no extra horizontal white space or punctuation; for headings which are grammatically distinct from the text which they introduce, a change of font is also indicated, but punctuation (e.g. a colon) or additional white space (e.g. one quad) is also frequently used. Such a heading might be set off by as little as 1 ex additional white space from the preceding text, and certainly by not more than one blank line.

At the next level in the hierarchy, the heading usually occurs on the immediately preceding line, and occupies a line in its own right. It is not set off by any additional vertical white space, but simply separated from the text by the normal leading for the paragraph. Again a change of font is indicated, and the font options applicable to run-in heads are equally applicable here, although the use of caps and small caps would be unusual. The extra vertical white space above the heading is of the same order of magnitude as for run-in headings.

A level higher and perhaps a larger font is indicated. Assuming a base setting of 10/12, a 12 pt font might be suitable for such a heading. If a bold font has been used for any lower level, then this font too must be bold, otherwise ambiguity will result (the same is true at *all* levels in the hierarchy: once a bold font has been used at a lower level, bold fonts must be used at all higher levels. In the same way, no font used in a higher level heading may be smaller than a font used in a lower level heading; it may be the same size, but only if it is bold and the lower level is not, or if there is other clear typographic indication of the hierarchy). Above such a heading a little extra white space might be allowed, perhaps between one and one-and-a-half blank lines.

Beyond this point, simple extrapolation is sufficient: as we move up the hierarchy, headings get bigger, bolder, more distinctive. The white space below them may increase, but only very slightly; the white space above increases, but not to ridiculous limits. Anything in excess of three blank lines is almost certainly excessive, and two blank lines are normally more than sufficient.

At this point it is appropriate to consider the implications of the above set of rules on TEX implementations. In order to allow successful page makeup in TEX, it is customary to allow the vertical white space associated with headings to be flexible (*i.e.*, 'rubber lengths', in LATEX's quaint terminology); but TEX has two quite distinct rules when dealing with flexibility: if a dimension is given a negative flexibility (*i.e.*, is allowed to shrink), then TEX will take advantage of the stated shrinkability if necessary to achieve optimal page makeup, but will never attempt to shrink it by more than the permitted amount; however, if a dimension is given *positive* flexibility (*i.e.*, is allowed to stretch), then TEX will first of all take advantage of that flexibility to achieve optimal page makeup, and if that flexibility is insufficient, *will continue to stretch it until optimal page makeup has been achieved*, even if this involves stretching it by many times its stated stretchability. Of course in these circumstances TEX issues a warning, but by then it is too late: the evil deed has been done.

The implications of this behaviour for successful implementations of design are quite severe: TEX must *never* be given positive stretchability to use if it is required to exercise any automatic control over the upper bound by which white space will be stretched; shrinkability can be used, but TEX is noticeably asymmetric in this respect, and whereas `\vfil` and its friends can be used to pad out underfull pages whilst preventing embedded . . . `plus $n$ pt` constructs from contributing white

TUGIndia
JOURNAL
Vol. 2 No. 1

space, there is no equivalent which can be used to negatively pad pages whilst preventing . . . `minus $n$ pt` constructs from shrinking (the reason is that TEX will not allow what it terms ‘infinite glue shrinkage’ to occur in unrestricted horizontal or vertical modes). Thus there are severe problems in inhibiting TEX from taking excessive advantage of permitted flexibility, and in the end only careful observation of the log file, and manual intervention where TEX has exceeded its brief, will be sufficient to keep matters under control.

But recalling for a moment the discussion on grid-based layouts which took place in the predecessor to this paper, it will be appreciated that simply preceding and following header lines by `\vskip` commands will not necessarily have the desired effect. A far more satisfactory method of placing headers, whilst ensuring that they occupy an integral number of blank lines (*i.e.*, an integral multiple of `\baselineskip`) relies on a technique which I refer to as a ‘pseudobox’: this is a TEX construct which is in reality a box whilst behaving like glue; the following code fragment illustrates the technique in use.

```
\newbox \headerbox
\newdimen \headerheight
\newdimen \headerdepth
\def \header #{\afterassignment \afterheader
               \setbox \headerbox = \vtop}
\def \afterheader {\noindent
                 \aftergroup \reallyafterheader}
\def \reallyafterheader
```

TUG*India* JOURNAL

Vol. 2 No. 1

INDIAN TEX users group

⏪ ◁ ▷ ⏩ ●

```
 {\headerheight = \ht \headerbox
  \headerdepth  = \dp \headerbox
  \advance \headerheight by \headerdepth
  \headerdepth = \headerheight
  \ht \headerbox = 0 pt
  \dp \headerbox = 0 pt
  \advance \headerheight by 0.5\baselineskip
  \divide \headerheight by \baselineskip
  \multiply \headerheight by \baselineskip
  \ifdim \headerheight < \headerdepth
   \advance \headerheight by \baselineskip
  \fi
  \vskip 0 pt
  \box  \headerbox
  \vskip \headerheight
  \noindent
  \ignorespaces
  }
```

If this code is used to typeset a large bold header within the text of this paragraph, as in `\header {\Huge Header}`, the effect *should* be to leave the remainder of the paragraph set on its natural grid;

Header

whether or not it has achieved this effect is left to the reader to see! Perhaps a brief explanation of the code is in order, as so far as the author is aware the technique has not previously been published. The `\header` macro takes no parameter, but

TUG*India* JOURNAL

Vol. 2 No. 1

INDIAN TEX users group

the terminal hash of its parameter list causes it to require an open brace to immediately follow its use; on the assumption that the open brace is the open brace of a brace-delimited parameter (which it should be, if the macro has been properly used), the macro sets `\headerbox` to a `\vtop` containing the parameter. However, an additional token is introduced into the `\vtop` just prior to the parameter by means of the `\afterassignment`, that token being `\afterheader`. This token itself expands into three further tokens, `\noindent` (to prevent the parameter from being indented within the box), `\aftergroup` (to allow the following token to be expanded not within the box but outside it, once it has been set), and `\reallyafterheader`, which is the macro that does all the real work. Thus the combined effect of the `\afterassignment` and the `\aftergroup` is to inhibit any indentation of the parameter, and to cause `\reallyafterheader` to be expanded once the box has been set. `\reallyafterheader` commences its work by saving the height and depth of the box in which the header has been set, and then computes their sum; the height and depth are set to 0 pt. Using Knuth's algorithm from A15.8, the combined height + depth is rounded to the nearest integral multiple of `\baselineskip`, and if the result of this rounding is less than the original sum, a further increment of `\baselineskip` is added. The result of this computation is the smallest integral multiple of `\baselineskip` within which the entire contents of the box can be set. A vertical skip of 0 pt is carried out (to force TEX into vertical mode), and then the box is typeset (remembering that it has zero apparent height and depth, and therefore occupies no space), after which a further `\vskip` of the calculated integral multiple of `\baselineskip` is carried out to leave room for the contents of the box whilst not disturbing the regularity of the baseline grid, Finally `\noindent` and `\ignorespaces` ensure that the first paragraph following the header is typeset correctly.

A real-life instance of this code would require parameterisation to indicate the level of header, from which it could ascertain (by means of a look-up table) how to distribute any required additional space around the header; in addition, it would enable ragged-right setting within the header box, and would need to deal correctly with a header immediately followed by another header (the spacing should not be additive). Many other refinements are possible.

4.6 Paragraphs

In trying to make practical recommendations for real-life book design, it is necessary to alternate between those entities which occur fairly rarely (opening chapter pages, headers, etc.) and those which form the bulk of the book (regular pages, paragraphs, etc). Here we consider material which makes up the vast bulk of the book, to whit the paragraph.

TUGIndia JOURNAL

Vol. 2 No. 1

Fortunately the 'rules' for paragraphs are fairly straightforward, but as so many examples may be seen which either blatantly ignore the rules or are simply unaware of them, some discussion is none the less necessary. It should be noted, however, that these rules are inherently culturally based, and I am advised by one eminent French authority[6] that the rule stated below concerning the first paragraph of any new section would be incorrect were it to be applied to material published in French.

- The first paragraph of a new section is not indented. This rule is so often more honoured in the breach than in the observance that I sometimes wonder whether its existence is widely known at all. For reasons entirely unclear to me, LaTeX whilst doing its best to honour this rule indents abstracts, which seems to me at best inconsistent and at worst inexcusable. I am very pleased to see that these proceedings avoid that error.

- A paragraph is either indented, or is set off by vertical white space from preceding material. It is normally considered infelicitous to do both; it is a gross error to do neither. The reason why the latter is so severe a crime is that if paragraphs are neither indented nor set off by vertical white space, then any text in which a paragraph just happens to end

[6] Bernard Gaulle, past and future President of GUTenberg, the French-speaking TeX Users' Group.

TUGIndia
JOURNAL
Vol. 2 No. 1

flush with the right margin will be followed by a paragraph whose existence can barely be guessed at. There will be *no* typographic clues to indicate that a new paragraph has started.

- The leading and font within a paragraph are uniform. This may seem to go without saying, but if a document is set with the minimum leading necessary for unadorned text, then an accented capital letter may well be enough to force down the entire line on which it occurs. In such circumstances either the leading must be increased for the entire document, or special steps taken to conceal the height of the accented letter (whilst ensuring that it does not unfortunately co-incide with a descender from the line above). By 'uniform', when applied to the font, I do not suggest that every glyph in the paragraph must be set in the same font; clearly there may be a need for italics, or even for a *sans serif* font at points, as indicated above. But all the glyphs within the paragraph should *appear* uniform, and must therefore come from closely related or well chosen fonts. For example, the first phrase of each paragraph in a book may be set with an initial full cap and then small caps; provided that these blend in with the main text font, there can be no objection to this. Similarly the first letter of the paragraph may be a dropped cap; provided that it too blends in with the main text font,

TUG*India*
JOURNAL
Vol. 2 No. 1

that is a perfectly valid design decision (and sometimes very stylish, if I may express a personal opinion).

- A paragraph should not end with only a part-word on the last line. Assuming that hyphenation is permitted at all (which it will need to be if fully justified text is specified), then the last line of a paragraph should end with at least one full word and preferably more. Plain TEX's (and LATEX's) setting for `\parfillskip` do not encourage this; a more felicitous setting might be `\parfillskip=0pt plus 0.7\hsize`, which encourages longer last lines at the expense of setting some such lines slightly loose.

4.7 Graphics, figures, and other 'floating' entities

Although there is much more than can (and should) be said about book design in general, I feel that there is one area which must be treated before I close, and that is the whole area of insertions, or as LATEX terms them, 'floats'. These are, in some general sense, graphic entities, although they may turn out to be purely textual in content. What really typifies them, however, is that they are invariably indirectly referenced; that is, they are referenced by the author in terms of *see Fig. 1* or *See Table 2.4*, rather than being implicitly referenced by position in the text as in, for example, *as shown below*. By virtue of the indirect

TUGIndia JOURNAL

Vol. 2 No. 1

nature of their reference, they can be physically remote from the point of reference, but one of the major skills of page makeup is the careful placement of such entities. The cardinal rule for these insertions is that *they must be capable of being seen from the point of reference*. One of the little appreciated strengths of TEX is how well it carries out this task for footnotes, which are a very simple instance of insertions; if you look carefully at TEX-set material which has many footnotes, you will probably be surprised at the number of times that a footnote reference occurs on the very last line of the page (before the footnotes themselves appear, that is). If you have not thought about this problem before, you may casually remark to yourself "that's lucky; another line and the footnote marker and its text would have appeared on different pages". But now try to find an instance where that has happened; try as you might, I suggest that you won't. And that surely suggests that it is more than luck that causes that particular juxtaposition of footnote marker and start of footnotes to occur so regularly, so reliably, and so consistently. And of course it *is* more than luck; all the while that TEX is accumulating material in galleys, it is carefully tracking how much space is occupied by footnotes and how much by the main text; and as soon as the combination of the two exceeds the available space on the page, TEX knows that it must cut the galley at or near that point and start a new page.

Now footnotes are, as I said, a particularly simple instance of such insertions; no-one minds if the text of a footnote is started

TUG*India* JOURNAL

Vol. 2 No. 1

INDIAN TEX users group

on its page of reference but continued on the next (no-one but a pedant, that is). But figures, tables, graphics, etc., are a very different kettle of fish; they are essentially indivisible entities, and can therefore either appear on a given page or not appear on that page; there are no half measures which would allow a part of the figure/table/graphic to appear, and the remainder to appear on the next page.

So now put yourself in the position of TEX, this time not accumulating text and footnotes, but accumulating text and (say) figures. TEX continues to accrete material in its galley as before, and encounters a reference to a figure; say that the page is only a third full. If the figure is less than two-thirds the depth of the page, there is no problem: TEX simply adds the figure to the list of things that appear on that page and carries on. But now let there be a second figure reference, may be two-thirds down the page: TEX looks to see how big the figure is, and discovers it needs half a page to itself. What does TEX do? The first choice is trivially ruled out; you can't have the reference to the figure followed by the figure, because (a part of) the figure would fall off the bottom of the page. OK, what's the next choice? Remember that the figure can *float*. So, let's try floating the figure to the top of the page on which it was referenced: no problem there, the figure appears at the top of the page, pushing the textual material down. Some of the textual material will fall off the bottom of the page, of course, because we already know that we have 2/3 of a page of text, and 1/2 of

TUG*India*
JOURNAL
Vol. 2 No. 1

INDIAN
TEX
users
group

◁◁ ◁ ▷ ▷▷ ●

a page of graphics, so 1/6 of a page of text falls off the bottom. But that's no problem, because textual material can normally be split at almost any point: so TeX chooses the nearest valid breakpoint and carries the remaining material over to the next page.

Then what happens? Well, think about what is *on* the material that has been carried over: the reference to the figure that caused the trouble in the first place! So now we have the figure on page n, and the reference to the figure on page $n + 1$. If $n + 1 \equiv 1 \pmod 2$ (sorry, if $n+1$ is odd!), then there is no real problem, for the reference to the figure occurs on the recto half of the spread, and the figure itself occurs on the verso half of the spread, so all is well. But if $n + 1$ is *even*, then all h@ll breaks loose: because the figure is on the recto half of a spread, and the reference to the figure is on the verso half of the next spread; and when the reader finally encounters the reference to the figure, the figure itself can no longer be seen. And no matter what TeX were to do in those circumstances, it would not be able to solve the problem without assistance.

So there are some problems in page makeup that simply *cannot* be solved by naively applying rules; rules are all very well, but eventually the time will come when the author's text and the rules of design are simply incompatible, and in those circumstances you will have little option but to liaise with the author and attempt to persuade him or her to re-write the offending portion of the text. If the author is dead, and the text is cast in tablets of stone, then you will have to do a lot of work by hand, may be setting a whole series of paragraphs one line looser than ideal, just to force a reference onto a more appropriate page. But when you've done it, and the finished book is printed, and you look at it and *know* that there are no further improvements that you could have made, then a great warm glow will fill your body and you'll know that it's all been worthwhile. Good luck!

5 TUG98 – A Report

Kaveh Bazargan
kaveh@focal.demon.co.uk
Philip Taylor
p.taylor@vax.rhbnc.ac.uk

This document is generated using pdfLaTeX and hyperref.sty of Sebastian Rahtz. Therefore, you need to download it separately by clicking the download button below. If you have already got the file, simply click view button.

View   Download

TUGIndia JOURNAL
Vol. 2 No. 1

INDIAN TEX users group

⏪ ◁ ▷ ⏩ ●