Musings

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

After having worked in one area for so many years, I decided that it is time to choose that area as a topic for musing. That area is writing and writing's relationship to programming.

Stephen King wrote that writing is telepathy [1]. Although King often writes about the metaphysical, in this case he is writing metaphorically. By telepathy, King means that the successful writer takes the ideas in his head and transfers them into someone else's head. There is no magic involved. What is involved is a lot of time, work, and careful consideration.

Technical Writing

I imagine that anyone reading this essay routinely does technical writing of one sort or another: emails, bug reports, proposals, articles, and especially papers. And this is where the telepathy had better happen. If you fail to pierce the fog between you and your readers, your thoughts will not be transmitted or will, at best, be misunderstood.

I ran across a wonderful example of just how technical writing can fail when done poorly and succeed when carefully crafted. I was reading a paper to be presented at a USENIX workshop and was amazed at just how clearly the authors presented their ideas. They explained each concept, which then flowed into the explanation of the following concept, until they had built both a description and an argument about the validity of the research in their paper. Their writing was crystal clear.

That paper received a Best Paper award. At this point you might be wondering why I am not citing that paper, but I have a reason. The first time these authors submitted their paper to the same conference, the paper was rejected because the program committee members didn't understand it. The authors were convinced that their work was good, but they had failed to truly communicate their ideas. The concepts were stuck in the authors' heads.

So they rewrote their paper, then edited it to within an inch of its life. I found this out after I met the authors during the conference, commended them on their wonderful writing, and asked them to reprise their paper for *;login:*. That's when I learned of their tale of failure and redemption.

I often run into very similar problems when editing articles for *;login:*. The authors clearly understand their topic, but leave out so much of *what is familiar to them* that the result is incomprehensible to everyone else. They have failed at telepathy, not because they are not psychic, but because they have not managed to communicate what other people who didn't work on their project couldn't possibly know.

When I run into this I try to coax people into explaining the missing parts. But I often fail for two reasons. There are people who just can't make the leap between ideas being in their heads, perhaps in wondrously beautiful constructs, and the very lack of the foundations necessary for these constructs to appear in the heads of their readers.

The second failing is my own. Once I have read an article many times I begin to understand the authors' points. Sometimes, instead of editing the article, the authors will attempt to explain the subtleties in email communications. So I learn what the authors have been trying

to communicate—I become contaminated with the very ideas that they failed to include in their article. At that point, I can no longer discern that the article has failed to communicate, except to those who already understand the topic and won't be reading the article anyway.

Writing and Programming

At one point I had given up on computing in general when a friend did two things that changed my life. First he asked me to store a box of materials for him while he traveled. The Zilog Z80 CPU manual was at the top of this box, and I realized that the world of computing was about to change: everyone would be able to have their own computer.

The second thing was what that friend, Madison Jones, told me. He said that if I could write, I could program.

I had programmed in college and had even written tools that my advisor used, but I had little faith in my ability to program. In retrospect, I believe that's because I spent a lot of time around people who were incredibly good programmers, and my skills had faded into inconsequence by comparison. Jones' words did serve to inspire me, and I went back to college to refresh my skills and restarted my career.

Dan Kaminsky, in a posting to the langsec-discuss mailing list [2], reminded me of the words of my friend when he wrote:

> Programming languages are about getting intent from human to machine. The human is an inherent part of this equation.

Programming is how we communicate human intent to computers. If you can write text in an organized and understandable way, you should be able to translate that text into a programming language. The translation itself is often faulty, as our mother tongue is not Go or JavaScript. But more often the problem lies in the programmer's inability to have clearly spelled out his ideas first, before attempting the difficult translation into a programming language. Having failed at description, the telepathy, the human-to-machine transmission, fails as well.

There is another way in which this failure to communicate appears in programming that is related to this very same failing. When very smart people write programs, they often begin by writing for themselves. If these people are part of a culture of very smart programmers (think Google, Facebook, LinkedIn, Apple, Microsoft, Red Hat, IBM, and many other organizations), they will also write for the people in their own cultures.

But those people are very poor at representing the rest of humanity. Most humans don't live and breathe advanced algorithms, although where I most often see these problems manifesting is in human-computer interfaces. When you are an insider, the intent of a few pixels in an interface is crystal clear, while outsiders have to guess that right-clicking or swiping over those pixels will produce a desired result and, hopefully, not be the icon for resetting the app to factory defaults, including wiping all the app's data.

Again, the program's authors have failed at their telepathy, keeping in mind that telepathy is not magic. Telepathy in practice is the art of communicating the ideas in your head to someone who doesn't share your head space.

The Lineup

We start off this issue with two articles about programming. While profiling can tell you about areas where your code is running often, causal profiling instead instruments code and tells you where improving the running time of a section of code will improve the overall performance of your code and by how much. Charlie Curtsinger and Emery D. Berger explain their technique, coz, and tell you where you can get the coz code and run your own causal profiling.

Peter Gutmann, a cyberpunk from New Zealand, is the hero of this issue. He has written an article about using AFL, an automated fuzzing toolkit, as well as a book review. Fuzzing your code has been an art form, but AFL along with a compiler like clang or a very recent version of gcc with ASAN support turns fuzzing into something you can do as part of routine testing. I also interviewed Peter because he's well established in several areas I wanted to explore: cryptography libraries, supporting open source software, and the safety of programming languages. I asked him about all of these topics and more.

I approached Michael Backes about his Boxify paper (USENIX Security '15), and he and his coauthors have taken the time to explain how they have expanded on that work to build a privacy-enhancing application for Android. Far too many Android apps want access to everything your smartphone has to offer, including sensitive information like your texts and contacts. With a newer version of Android (4.2+) and Boxify, you can limit the ability of apps to have access to anything the app developers or advertising included by the app might want to suck off your Android device.

Martin Preisler works on OSCAP, software that uses vulnerability information to audit Linux systems. OSCAP only works for some distros, but it doesn't just work on mounted file systems: OSCAP can scan VM images and containers too; OSCAP can also perform remediations, such as fixing permissions or correcting configurations.

I interviewed Nick Weaver, asking him to explain the path that led from designing FPGAs to writing about the first Warhol worm and becoming an informal expert on bulk data collection. Nick spoke on data collection at the first Enigma conference and explained how to de-anonymize Internet traffic.

John Bent, Brad Settlemyer, and Gary Grider share their perspective on how HPC (high performance computing) has changed, and will change, the way distributed storage systems work. HPC has very unusual storage requirements, but these relatively rare installations can teach us a lot about how distributed storage systems will work in the future.

I attended the Linux FAST Summit '16 and took a huge amount of notes. Instead of attempting to transcribe those notes, I have written a summary of the event this time. I also have a note about the FreeBSD Storage Summit that occurred the day before FAST '16 began.

Jonathon Anderson tells us how he is using policy-based routing at the University of Colorado Boulder Research Computing. With iproute2, you can go beyond traditional routing tables by adding policies that override default routes, techniques that really improve performance and latency for services on dual-homed servers.

Mihalis Tsoukalos provides an introduction to MongoDB administration. Like many other NoSQL databases, MongoDB has quirks that you'll want to know about before you start using it.

Peter Salus has collected many of his observations about the history of Linux into an article. The Linux system turned 25 this spring, and Peter tells us about the beginning of Linux and how various distros were born and how various distros died.

David Beazley demonstrates how to improve memory usage in Python. While creating a dict might be the easiest way to import a large amount of data, David demonstrates how using a dict is the least efficient method when it comes to minimizing the memory footprint.

David Blank-Edelman investigates the Spotify API while using Perl. David shows us techniques for exploring the RESTful interface used by Spotify and how to delve into the results returned for a richer understanding of the interface.

Kelsey Hightower explains the `vendor` directories that are now part of Go. Previously, dealing with external libraries required using an additional tool. The `vendor` directory allows you to include and manage external libraries within your Go project.

Dave Josephsen covers Go in his column too. Dave treats us to some neat tricks for instrumenting a service with a view to expanding that monitoring to cover new services with the addition of just a few lines of code.

Dan Geer explains how sometimes your metrics work better when you examine the trajectory of your data rather than their absolute values. Dan focuses on the data he has been collecting monthly about security trends for the past five years.

Robert G. Ferrell takes a deep look into Fuzzy Thinking as applied to cats, politicians, consumers, and fuzzing programs.

We have five book reviews this time, with three by Mark Lamourine, one by Peter Gutmann, and one by me.

Finally, John Yani Arrasjid writes about why he has donated so much of his time and energy to USENIX over the past 30 years.

Coding and Algebra

Like many of my contemporaries, I learned a lot of math before I ever had a chance to program a computer. My high school algebra prepared me for inorganic chemistry and classical physics, classes which I aced because they so nicely correlated to simple algebraic equations. And that appeared to lead me right into college programming classes.

I also tutored fellow classmates in chemistry during high school. While I found the relationship between algebra and chemistry crystal clear, the people I was tutoring did not. These folks were in the same college prep school I was in, equally smart, but my brain and their brains were not wired the same way. What I found simple, algebra, they had great difficulty comprehending.

When I heard of people suggesting that high school students not study algebra, I wondered if you could teach coding without algebra. After a few minutes searching, I ran across Andy Skelton's essay about coding and algebra [3]. Getting to the gist of Skelton's point, first put yourself into the mindset of algebra instead of programming. What do these three statements mean in algebra?

```
a = 2
b = 3
a = b
```

As a programmer, you might optimize this without even thinking about it to:

```
b = a = 3
```

But in algebra, the third statement, `a = b`, is nonsense because `a` does not equal `b`. What the heck! So algebra, even though it is where students learn about abstraction and how to solve word problems, doesn't map very neatly into coding. Not at all.

I also ran across a completely different approach, teaching algebra *through* coding. Bootstrap [4] provides class materials for teaching students about coding through programming a game. Instead of struggling to learn the abstractions of algebra, Bootstrap uses the design of a simple game to teach both programming and math through concrete examples.

I want to end this little essay by encouraging all of you to take the time not just to write, but to polish what you write. Stephen King is famous for his writing, but do you think that he just sits

down and knocks out best-selling novels? No way! King writes a first draft, perhaps 1000 words at a time, then puts that draft away for a month without showing it to anyone. Then he pulls out his draft and begins reworking it until he gets it into a shape where he is willing to show the early stages of a novel to some close friends.

Writing is not something that springs into existence as if a god had struck a rock and, miraculously, clear water pours forth. Just like programming, good writing is something that results from planning, careful work, debugging/editing, and, finally, testing. If you have to explain what you have written to your test audience, you have failed at telepathy. Go back and rework what you have written until a fresh test audience can understand it.

Remember the paper authors? One year their paper gets rejected in the first round, and the next year that same paper, carefully rewritten, wins a Best Paper award. The research was the same, but the exposition of that research wasn't.

Resources

[1] S. King, *On Writing: A Memoir of the Craft*, Scribner, 2000.

[2] Lang-sec discuss: https://mail.langsec.org/cgi-bin/mailman/listinfo/langsec-discuss.

[3] A. Skelton, "Programming Is not Algebra": https://andy.wordpress.com/2012/05/30/programming-is-not-algebra/.

[4] Bootstrap: http://www.bootstrapworld.org.