DOI:10.1145/3563965

The Blood Price of Unrestricted Privacy

Reinhard von Hanxleden ends his May 2022 *Communications* Viewpoint "Information: 'I' vs. 'We' vs. 'They'" (p. 45) by pointing out the unthinking application of unconditional criteria to privacy "seems like a dead end in the long run." It has already proven to be a "dead end" with a Germanwings airliner crash into a cliff in France caused by the copilot. His doctor had given him a sick note, which the copilot threw away, and the doctor was prevented by strict legal prohibitions from communicating his findings of unfitness for flying to the authorities or the airline. The blood price was 199 lives, not including the copilot.

This type of event is entirely foreseeable, as shown by legal requirements in other countries for doctors to communicate their findings if they find a pilot unfit to fly. It is not unreasonable to infer similar tragedies have happened without coming to light or because of a lack of imagination by monomaniacal zealots.

John C. Bauer, Ontario, Canada

Author's response:

Thanks for that comment. The Germanwings crash is indeed a tragic case where the conflict between privacy and other goods manifested itself. There is, of course, the practical concern that one does not want to deter pilots from seeking treatment, but yes, a pilot's fitness to fly is not an entirely private matter. Confidentiality of conversations between patients and doctors is probably one of the most established privacy concepts, going back to the Hippocratic Oath. However, societies have also understood by now that there must be limits to that confidentiality. Thus, there is also a canon of exceptions that keeps being renegotiated, in particular after dramatic cases such as the Germanwings tragedy. However, most of today's "digital privacy" concerns, such as which technologies might be used in education, seem to not have matured yet to that stage of an informed, serious debate.

Reinhard von Hanxleden,
Kiel, Germany

Could Not Make It Plainer

Poul-Henning Kamp's June 2022 *Communications* article "The Software Industry Is Still The Problem" is a great challenge. We have two problems—shoddy technology and poor attitude clinging to outdated technology.

We cannot trust our systems because they have weak security. Security at all levels is to "define boundaries and enforce those boundaries." Boundaries must be enforced at all levels.

We should keep strings and arrays (contents) within the bounds of the assigned memory blocks (container). Where security is not built into the lowest levels we have a weak foundation and security and systems can be undermined. Security is added only as an afterthought with utilities and other software—lipstick on a pig.

Pointer-based languages are weak, and pointers unnecessary. Memory blocks must be defined by more than a base address, but length and other metadata as well. Programmers should think of references to objects (contents) not pointers to memory locations (container)—too many programmers have been taught to think about the container rather than the contents.

Programming languages must define and enforce bounds. But we cannot trust languages, compilers, and runtimes. Hardware must check with descriptor or capability-based architectures. And assembler in 2022? We are still in the dark ages to which assembler should be relegated.

We need separation of concerns that clearly separates system programming handling the container from application programming about the contents. While security is a cross-cutting concern, it must be based on security at the lowest layers of system architecture.

We cannot "trust the programmer" or have faith the "programmer knows what they are doing." Buffer overruns occur either because the programmer does not know what they are doing—a situation we are all in frequently, if we admit it. If the programmer knows what they are doing they are doing it for malicious intent.

Foundations and checks are disdained as "training wheels for beginners." Programming languages at all levels should support modern development in the modern connected environment that needs security. Programmer support, validity checks, and secure bounds should stop being considered "crutches for weak programmers," "hand holding," or against "programmer freedom." One person's freedom is another person's burden. We all pay for weak security.

As Poul-Henning Kamp rightly points out the software industry is still the problem. It is both in technology and attitude. We continue to ignore the elephant in the room, and indeed many defend the elephant with tribal brand loyalty and cult-like fervor.

The software industry often ignores the problems of foundations, preferring to put lipstick on a pig. With that elephant in the room, the woodpecker's job is indeed easy!

Too much teaching and training is being done around the flaws of what is, ancient compromises and constraints, rather than what computing should and must be. Stop defending the status quo. The software industry must change—both its technology and mentality—in profound ways.

Ian Joyner, Sydney, Australia

Author's response:

Poul-Henning nods vigorously.

A Little More Precision Would Be Nice

When I saw the title of the June 2022 *Communications* article "Challenges, Experiments and Computational So-

lutions in Peer Review," I was excited to see that you were going to address one of the more important processes in producing quality software code. Even the subtitle, "Improving the peer review process in a scientific manner shows promise" encouraged me into thinking that you were going to introduce some rigor in expunging bugs from software code under review. Alas, my hopes were dashed when I realized that this was really an article about publishing scientific papers for conferences. In my youth (the 1970s) I campaigned against flow charts and for software peer reviews, and my attitude has not changed in these regards.

Warren Scheinin, Redondo Beach, CA, USA

Voice for Users' Full Control Over Applications

Programs usually perform not what users want but whatever developers consider being good for users. Millions of users try to achieve from programs what these users really need and are involved in the ongoing struggle with applications. The situation was perfectly described in preface to Lieberman.[1] "So-called "applications" software for end users comes with an impressive array of capabilities and features. But it is up to you, the user, to figure out how to use each operation of the software to meet your actual needs. ... You have to translate what you want to do into a sequence of steps that the software already knows how to perform, if indeed that is all possible."

This conflict became obvious long ago, so years ago the adaptive interface was proposed as a solution. After several decades and tons of suggestions, there is no improvement. Why? All achievements of adaptive interface are aimed at softening the conflict but never tackle the problem source.

For possible solution, let's look at a similar situation. We have thousands of items in our household, and not all the time these things around are organized in the best way. Whenever needed, we move and rearrange the surrounding items. Our ability to organize our life in the most effective way is based on two things:

- Everything is movable.
- We easily move anything at any moment without asking permission from anyone.

In programs, everything is controlled by developers, while users can perform only the allowed steps. Historically, and this started before the era of personal computers, developers had absolute control over applications, and throughout years this turned into axiom. But it is not! Give full control to users, and we'll have different programming world.

What is needed to pass full control to users? The mentioned example gives a perfect answer: movability of each and all. The movability of elements automatically sets new rules. While users' control over applications can be rejected as nonsense and an impossible thing, this is not a baseless theorizing. I applied movability to some of the most sophisticated programs—scientific applications. On trying new programs, scientists required further development to be done only in such a way.

There is a book (*User-Driven Applications for Research and Science*) and there are programs that demonstrate the implementation and results with many different examples (https://bit.ly/3qWHVFY). Programs come with codes, so they are for everyone to look at, to try, and to see that any program from the simplest to the most sophisticated one can work under full users' control and for users' great advantage.

Advantages of new design increase with the unpredictability of users' actions or requirements. Science and engineering are areas where the full control in users' hands allows those users to do and check things that nobody before even thought about. This is my understanding of a progress.

Reference

1. Lieberman, H. et al. *End-User Development*. Springer, 2006.

Sergey Andreyev, Moscow, Russia

Communications welcomes your opinion. To contribute a Letter to the Editor, please limit your comments to 500 words or less, and send to letters@cacm.acm.org