The Planning and Care of Data

REARRANGING BUCKETS FOR NO GOOD REASON

Dear KV,

After several years as a startup, we seem finally to have gotten large and popular enough that management and legal are looking at how we store and segregate our user data. To say that this feels like a fire drill would be understating it. Everyone now seems to have an opinion on how we should handle user data. Meetings on this topic would be funny if they weren't so tragic. It's not as if we're a huge company that claims billions of users, and, of course, it's important to protect our customers' data. But all the hand-wringing at this point seems to be very late in the game and is likely to end up causing a huge amount of engineering that ultimately does not add value to the product but is, instead, a way for management—or perhaps legal—to cover their... you know. I can't imagine this is of value, but maybe you have a different opinion? I suspect at the very least, you do have an opinion that will be more fun to read than the emails from legal. I feel as if we're just rearranging buckets for no good reason.

Bucketed for No Good Reason

Dear Bucketed,

In a world that now contains so many rules and regulations around how a company handles user data, I'd like to say that I'm surprised your company managed to go several years before reaching this juncture. But bad news rarely surprises me, even less so than stories of people not thinking about how to handle the data they receive.

KV is not often reflective. However, in rare moments, when he's had a properly made Vesper martini that renders him relaxed enough to reflect, these are thoughts that wander through his addled brain.

It isn't just rules and regs that ought to cause people to think about data engineering and data maintenance; it is the fact that we have now come to a place in computing where data has significant value and significant risk—in equal measure. A wobble down memory lane shows us that the trajectories of engineering efforts through computing have changed significantly over the past 70 years. While 70 years might be considered a short time in some of the traditional sciences, the amount of change over that time in what matters to people working with computers has been dramatic. We have moved from the 1950s and 1960s, where hardware was the dominating cost and the focus of our efforts, to the rise of software in the latter part of the 20th century, to the rise of data in the early 21st century. Why?

Moore's law has a lot to answer for here, as well as the human inability to throw stuff away once we've collected it. Parkinson's law ("Work expands so as to fill the time available for its completion") has a corollary, "Data expands to fill the space available for storage," which has been the case ever since we've had the ability to store data.

As a kid, I remember visiting my uncle's office at his university where he had stacks and stacks of punch cards.

"What are these?" I asked.

"That's all my astrophysical data for my research," he explained.

My uncle had only limited space for his boxes of punch cards, so I encouraged him, as a snot-nosed 16-year-old KV, to switch to tape. I never asked if he did, but I bet if he did, he would have wound up with even more data than would fit in the cubic feet available in his office.

As time progressed, software came to dominate the cost of systems because computers became cheaper and more powerful, and, therefore, we could write larger and more-complex programs, which then became systems of programs, and then distributed systems of programs.

All this increasing complexity forced us to find solutions to a software crisis that was well described by Dijkstra in his 1972 ACM Turing Lecture:

But instead of finding ourselves in the state of eternal bliss of all programming problems solved, we found ourselves up to our necks in the software crisis! How come? ... The major cause ... is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

— Edsger Dijkstra, *The Humble Programmer* (EWD340)
https://www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD340.html

The software crisis has never abated, no matter what. Often, ridiculous trends have appeared to supposedly address it. Modular programming, object-oriented programming, pair programming, Agile, Scrum, whatever, were all meant to address the fact that the hardware—and particularly the software—that we were building was, and continues to be, too complicated for those of us who work with it to understand.

As compute and memory got cheaper, so did storage. In the '80s, the early micros could store a few hundred kilobytes of data on floppies, or, if we were rich, we might have a 10MB drive in a PC. Twenty years later, which is now 20 years ago, that went to many gigabytes of storage, and now it's terabytes—and that's what we can *personally* store. Data centers, of course, went through similar, spectacularly quick growth in storage space.

You really cannot just dump data into a cloud bucket or any other large storage system and expect that everything will work out for the best.

It's not just the amount of data we're storing; it's the relationships among the data. The relationships drive the complexity, just as the explosion of libraries and packages used in modern software drives up the complexity and cost of software systems.

What does all this mean in 2022?

It's well past the time when everyone who even thinks about collecting data, user or otherwise, must first think seriously about data engineering and data maintenance, because the costs of getting it wrong are far too high—both monetarily and societally. I'd like to say that no sane person simply sits down and starts typing code—with just a loose idea in mind—and expects things just to work out in the future. If I did, I'd be lying through my Lillet-soaked teeth. What goes for software engineering goes for data engineering. You really cannot just dump data into a cloud bucket or any other large storage system and expect that everything will work out for the best.

There are people who have thought about this for a long time, but they often don't have much sway anymore. Before the rise of stupidly cheap compute and all the nontransactional database systems, we had people who were specialists in how data should be stored, and these people were necessary for an efficient, data-storage back end. These were the database administrators, but these people are rarely involved in getting startups going because the startups see code first and data second, unless their real go-to market is to get one of the FAANG—Facebook (now Meta), Amazon, Apple, Netflix, Google (now Alphabet)—to buy them for the value of that data. Even then, they're more like vacuum cleaners, sucking up everything they can get a hold of, with little concern for its safety, future value, and risk.

Even when companies start down the right path, they usually fail at data maintenance, just as companies fail at software maintenance. New data is accreted without plans and it piles up everywhere because people figure they will just sprinkle on some machine-learning magic and get more value out of it.

There are no magic bullets in engineering. If you slap an extension on a house without thinking about its effect on the overall structure, your extension, or the whole damn house, is going to be damaged and, in the worst case, come crashing to the ground. Our industry is littered with these data corpses, but a little bit of planning at the start and care throughout the lifetime of the data will pay off handsomely.

Questions such as, "How do we secure this data?" work only if you ask them at the start, and not when some lawyers or government officials are sitting in a conference room, rooting through your data and logs, and making threatening noises under their breath. All the things we care about with our data—security, privacy, efficiency of access, proper sources of truth—require forethought, but it seems in our rush to create *stakeholder value* (a bullshit term used to justify so much) we are willing to sacrifice these important attributes and just act like data gourmands, until, like Mr. Creosote in *Monty Python's The Meaning of Life*, we explode, scattering half-digested data all over the dining room.

Now that data has surpassed most software in size and complexity, it's time to make data engineering and data maintenance first-class topics of study. To do anything else simply invites us to make the same mistakes and put people and our companies at risk.

KV

Kode Vicious, *known to mere mortals as George V. Neville-Neil, works on networking and operating-system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. Neville-Neil is the co-author with Marshall Kirk McKusick and Robert N. M. Watson of* The Design and Implementation of the FreeBSD Operating System (second edition). *He is an avid bicyclist and traveler who currently lives in New York City.*