Realizing the impact of the developments of integrated circuits, my research group shifted focus from developing better electronic devices to designing micro-devices that could sense and interact with nonelectric variables such as force, pressure, and photons. That first micro-motor and other devices like it called out for a place to house the partnerships and entrepreneurial applications of the lab work. To fill this need, a colleague, Dick White, and I founded the Berkeley Sensor & Actuator Center (BSAC) in 1986, with the support of NSF and five industrial sponsors. A breakthrough achievement occurred in the early years of BSAC when an industrial sponsor (Analog Devices, which had joined while Analog was solely an IC manufacturer) applied the BSAC-invented poly-Si mechanical-device processing technology to produce a breakthrough accelerometer for automobile airbags. Over the 30-plus ensuing years, scores of advances were carried out at BSAC as the MEMS/NEMS industry grew from negligible size to its present tens-of-billion-dollar worldwide product size. BSAC itself has likewise grown, until today it is directed by 13 research faculty, who typically supervise more than 100 graduate students in programs associated with roughly 35 industrial sponsors. In 2013, the impact of BSAC was honored by the presentation of the IEEE/Royal Society of Edinburgh (RSE) James Clerk Maxwell Medal to BSAC—Dick and myself, as the founders.

**ENGIneous: For a scientist, you have had a heavy focus on communication and language skills throughout your roles in teaching and research. Why is that?**

**Muller:** I discovered this passion early, when I founded the first newspaper at my grammar school. I went on to run the newspaper in high school. I was the editor-in-chief of the Stevens newspaper at my undergraduate college. In those early years, I had a strong inclination toward a career in journalism—but the appeal of engineering caused me to decide on study at an institute of technology. An important part of my research career has been service as editor-in-chief of the IEEE/ASME Journal of MicroElectromechanical Systems from 1997 to 2012. These years of editing were valuable in learning to evaluate ideas and to express them with clarity and impact. With these perspectives, the researcher is equipped to recognize frontiers in a problem area and to decide on ways to push back these frontiers. Charting frontiers is the skill of pioneers. 📚

Richard S. Muller is Professor Emeritus in the Department of Electrical Engineering at the University of California, Berkeley, and Co-Founding Director of the Berkeley Sensor & Actuator Center.

**ENGIneous sat down with Dan O’Dowd (BS ’76, engineering) and his son Richard O’Dowd (BS ’13, computer science) to discuss their Caltech journeys, their dedication to computer science, and the field’s impact on the world.**

**Dan O’Dowd** discovered his passion for fixing bugs in code as a Caltech undergraduate student. Less than 10 years later, he founded Green Hills Software, and he is still serving as its president and chief executive officer. Before founding Green Hills Software, he was manager of compiler and operating system development at National Semiconductor, where he designed the architecture for the NS32000 32-bit microprocessor. Prior to that, he was at APIC Technologies, developing some of the first embedded development tools for microprocessors. These were used for developing the first handheld electronic games for Mattel and Mattel’s line of home video games. O’Dowd has crossed paths with many luminaries in computer science, including Steve Jobs, and his mathematically proven secure software, called INTEGRITY, has been essential to the aerospace industry. His son Richard works at Green Hills Software as a senior software engineer, developing the next generation of real-time operating systems.

**ENGIneous: How did you both come to choose computer science and Caltech?**

**Dan:** Caltech is the top school for computer science—it was then and it is now. I actually started out in mathematics, but I took some computer courses along the way, which I liked. Knowing that programming would mean guaranteed employment, I switched to computer science, which is still sort of math, just a more practical application. Of course, there was no computer science department at the time. That didn’t start until the year after I left, so I got an engineering degree, but with a focus in computer science.

**Richard:** My dad didn’t talk much about his time at Caltech. He did share some, but more of it was after I came to campus and we realized I was his legacy in Dabney. I wasn’t even certain which house he was in when I got here. I had an inkling it was Dabney, but I didn’t know for sure. As it turns out, we are both keepers of the Dabney jai alai tradition and were both and players of the “ball against the wall.”

**Dan:** I came to Caltech to visit Richard at Dabney shortly after Richard got here, and I had this weird déjà vu where I felt it was 37 years earlier.

**ENGIneous: Was it a coincidence that you both ended up at Dabney House?**

**Richard:** I was steeped in my dad’s interests. I was always a math and science guy, so Caltech was a natural fit. My dad did computer science, so I dabbled in it and in programming. But when I came to Caltech I gave myself two choices, biology or computer science, since they seemed like the two emerging fields going forward. I did try biology, but computer science proved much more interesting to me.

**ENGIneous: Was it a coincidence that you both ended up at Dabney House?**

**Richard:** My dad didn’t talk much about his time at Caltech. He did share some, but more of it was after I came to campus and we realized I was his legacy in Dabney. I wasn’t even certain which house he was in when I got here. I had an inkling it was Dabney, but I didn’t know for sure. As it turns out, we are both keepers of the Dabney jai alai tradition and were both and players of the “ball against the wall.”

**Dan:** I came to Caltech to visit Richard at Dabney shortly after Richard got here, and I had this weird déjà vu where I felt it was 37 years earlier.
It was at the same time familiar and unfamiliar, and I had the feeling of being a freshman coming in for the first time. I felt he had to come to that himself; that he ended up in Dabney was by way of his own path that happened to intersect with mine.

**ENGenious:** What Caltech experiences do you still carry with you?

**Dan:** I found a memory recently. I stumbled on a Manila envelope in an old box marked “September 1976.” Inside were my student records of fixing bugs for one month. I recorded everything I did, every bug I had, the mistakes I made, and how long it took me to fix them. I was studying my own process even then. Fixing bugs has been my professional career, but the original notion started at Caltech.

**Richard:** Caltech students are a lively bunch, up to all sorts of random shenanigans. Everyone is smart and coming to new ideas and new conclusions. Your mind is changed and you change minds in routine conversation about interesting topics. And then there is jai alai, of course.

**ENGenious:** Why is your collaboration with the aerospace industry unique?

**Dan:** What we do, for the most part, at Green Hills Software is solve the basic problems of software. Following traditional methodologies, software invariably has bugs and security vulnerabilities, which are simply not acceptable. The task is building something that people’s lives are dependent upon. And yet software is still being created the traditional way, so how do you fix it? The aerospace industry is a place where they take this question seriously. It’s the one industry where they really go to the trouble to do it well, because planes do crash, and hundreds of people die when they crash. There is major incentive to study and fix problems. I was interested in the problem of fixing software, the aerospace industry needed that problem solved, and we were equally dedicated to the solutions. That combination created opportunity, and now most modern aircraft use our INTEGRITY software and our tools for debugging the software.

We worked on this project together, maintained a friendship, and shared knowledge for a long time. He left Apple shortly thereafter, but I continued to interact with him for some years after that. I developed a product for him to use the happenstance of a single overlapping connection.

**ENGenious:** What advice do you have for the next generation of Caltech students?

**Dan:** I think you should do something you like, something you’re good at and you really like, because that makes it fun and then you accomplish things. I went from math to computer science because I liked it, but I liked computer science better. It felt like a big decision to make that switch, but it was the right one for me, and all of these years later I still know it was right.

**Richard:** I would advise dabbling in computer science. Actually, almost everybody at Caltech now do coding as a core skill of science and engineering. Many of my Caltech friends are now coding in addition to being engineers. We are the coding experts among our science and engineering students. The combination created opportunity, and now most modern aircraft use our INTEGRITY software and our tools for debugging the software.

**ENGenious:** How do you encourage engineers of your generation to put this notion of safety into practice?

**Richard:** You can lead by example. I feel like some engineers have a problem with constrained thinking. They often receive constraints mainly from management, of time and money. So the engineer says, “Okay, I’ll do the best that I can do.” And then they manage will say, “Well, now you have even less time and less money.” And the engineer will say, “I’ll do the best that I can do.” It’s going to be a good job, but it’ll be the best that I can do.” What they don’t say is, “No, we can’t do the right job with that much money and that much time.” The engineers need to push back in these cases, because this culture can be dangerous. It can kill lots of people. We have to think it all the way through and push back.

**ENGenious:** How did Caltech help you integrate this base layer of safety?

**Dan:** I work on technology, but I also have to get people to understand the fundamental ethical problem so that the decision makers will make the right decisions. When I was a student at Caltech, there was a course on engineering ethics, which taught us that we could design a product that wasn’t very good, or of low quality, but that we must say no at the point where we realize a device is an outright danger to humanity. To refuse to make a bad product is an acceptable moral choice but a potentially insufficient one. At some point you have to call it out, become the whistleblower, and foster support from other people if you believe there is a tragedy or catastrophe in process.

**Richard:** How are design integrity and logic having their meta-battle moment?

**Dan:** We need to ask why it is that we think we need the things we’re building. What is their value? If the answer is conveniences, cost savings, or entertainment, then follow that question with another: Is it worth it if that thing can kill millions of people? A prime example is the internet-connected home. Once you come to this concept and you identify cost savings or convenience opportunities, then you suddenly think when everything in the home should be connected as the next logical conclusion. “The stove is in the house, so it must be connected.” You can’t just push a high-level concept down to the lower levels without thought. At the point where you get to the stove, it becomes too dangerous. If someone can hack into your stove and cause an explosion in your home, that is unacceptable.

**ENGenious:** How do you make code reliable and safe?

**Dan:** We need to start by separating the critical code from the non-critical code, which greatly reduces thedebugging task. Often code serves to optimize, improve, or entertain, all of which takes a lot of code but doesn’t control the actual device. It’s okay if the non-critical part of software doesn’t work. It might be annoying, but it is not critical. We can’t make all code reliable—it’s just too much. We find a way to get it down to the few percent that is critical and make sure we do that part correctly. This can cost $1,000 per line of code. And before anyone even thinks about cost, consider this: If your device is such that it has the ability to kill lots of people, almost by definition is a big enough business to support this level of software scrutiny. The competition is huge. Cars are a trillion-dollar business. Trains are a hundred-billion-dollar business. Our exploding store? Tens of billions of dollars are spent on stores annually. Guaranteeing the safety of critical software in these products is a small reinvestment of profit that provides huge payoffs in the overall health of the consumer and the company.

**Richard:** How do you make code reliable and safe?

**Dan:** We need to start by separating the critical code from the non-critical code, which greatly reduces the debugging task. Often code serves to optimize, improve, or entertain, all of which takes a lot of code but doesn’t control the actual device. It’s okay if the non-critical part of software doesn’t work. It might be annoying, but it is not critical. We can’t make all code reliable—it’s just too much. We find a way to get it down to the few percent that is critical and make sure we do that part correctly. This can cost $1,000 per line of code. And before anyone even thinks about cost, consider this: If your device is such that it has the ability to kill lots of people, almost by definition is a big enough business to support this level of software scrutiny. The competition is huge. Cars are a trillion-dollar business. Trains are a hundred-billion-dollar business. Our exploding store? Tens of billions of dollars are spent on stores annually. Guaranteeing the safety of critical software in these products is a small reinvestment of profit that provides huge payoffs in the overall health of the consumer and the company.