DOI:10.1145/3106633

Article development led by acmqueue
queue.acm.org

Why the Bell curve hasn't transformed into a hockey stick.

BY THOMAS A. LIMONCELLI

Four Ways to Make CS and IT More Immersive

I was lucky. I learned IT in an incredibly immersive way. My first two jobs were in organizations that followed the very best practices for their day. Because it was all I knew, I considered that to be normal. I had no idea how unique those organizations were. I didn't know at the time that the rest of the industry would not adopt these techniques for a decade or more.

My next career moves brought me in contact with organizations that did not adhere to the same best practices, nor any others. In fact, they were unaware that such best practices existed at all. I considered this to be a bug and went about fixing it, dismayed that anyone would settle for anything else. I was re-creating what I considered "normal."

The environment I was trying to reproduce, however, was not normal, or more accurately, it was not typical. A typical IT organization is, in comparison, in utter disarray. The quality of IT organizations follows a bell curve: A few percent run like fine-tuned machines, a few percent look like toxic waste dumps on fire, and the vast majority are somewhere in the middle.

Fortunately for me, I won the IT career lottery. Early in my career I saw what the best in class looked like and considered it normal. Later, this high standard made me look like a visionary. The truth is I just didn't know any other way.

Most IT practitioners are not so fortunate. They are not blessed with the same experience I was afforded, and they literally do not know any better.

This, I believe, is why the bell curve has not transformed into a hockey stick, or is even a lopsided blob. This is why we cannot have nice things.

How Did We Get Into This Situation?

Students certainly are not learning best practices in the classroom. In fact, students are more likely to learn the best-of-breed DevOps practices through extracurricular involvement in open-source projects than from their university professors.

Most large open source projects use Git for source-code control, use Jenkins for CI/CD (continuous integration/continuous deployment), and have a fully automated testing procedure because it enables them to scale to large numbers of participants with minimal overhead. Smaller open source projects tend to use these tools, too, because they lack resources and using these tools makes managing the project significantly easier.

Yet, how many universities require CS homework to be turned in via Git commit? How many universities have an IT department that is a showcase for best DevOps practices? How many universities have CS departments and IT departments that collaborate to push the boundaries of best practices? I assure you the number is very low. It is no

PHOTO BY JESSE THE TRAVELER/FLICKR (CC BY-NC 2.0)

surprise that the innovations that led to the DevOps transformation did not come from academia. This is a disgrace that should make every CS department bow their heads in shame.

How Can We Turn This Around?

How can we ensure students are exposed to the best of the best practices from the start so that they consider anything else a bug?

How can we make curricula more immersive?

Here are a few small and big things that universities could do.

1. **Use DevOps tools from the start.** Students should use source-code repositories such as Git, and CI/CD tools such as Jenkins, as they do their CS homework. These processes should be established as the normal way to work. Professors should expect homework assignments to be turned in by linking to a Git commit and a Jenkins output log.

Some instructors will undoubtedly feel it is difficult enough to teach first-year computer science without adding the complexities of Git. Most IDEs, however, make simple check-in/check-out operations a breeze, especially for single-person projects with no branches. By the time projects get more collaborative, the students will be ready for the more advanced Git features.

2. **Homework should generate a Web page, not text to the console.** I recently spoke to a roomful of third-year CS majors and was shocked to learn that most didn't know HTML. The curriculum was fairly standard—undergraduate algorithms and such. HTML was something you learned in the art department; the computer science department was for serious students.

I think there is a middle ground between serious computer science theory and accidentally turning into a Web application boot camp.

It isn't a radical statement to say that most software engineers write code that is somehow part of a Web-based application. At companies like Squarespace and Google, software engineers' IDEs supply default templates for new programs. Such a template is for a self-contained Web server that directs output to a Web page. Even a simple "Hello World!" program is a Web server that outputs the greeting as a result of an HTTP request and, by default, generates logging information, monitoring metrics, and so on.

Yes, that is a bit much for an introductory student's "Print your name 10 times" program. But after that, generate a Web page!

3. **IT curricula should be immersive.** How could formal education better emulate the immersive experience that

I was lucky enough to benefit from?

Most IT curricula are bottom up. Students are taught individual subsystems, followed by higher levels of abstractions. At the end, they learn how it all fits together. Toward the end of their college careers, they learn the best practices that make all of it sustainable. Or, more typically, those sustainability practices are not learned until later, when the new graduate has a job and is assigned to a coworker who explains "how things work in the real world."

Instead, an IT curriculum should start with a working system that follows all the best practices. Students should see this as the norm. They can dissect the individual subsystems and put them back together, rather than building them from scratch.

The Masters of System Administration curriculum at the University of Oslo includes a multiweek immersive experience called the Uptime Challenge.[1] Students are divided into two teams, and each team is given a Web-based application, including multiple Web servers, a load balancer, a database, and so on. The application is a simple social network application called BookFace.

Once the system is running, the instructor enables a system that sends an ever-increasing amount of simulated traffic to the application. Each team's system is checked for uptime every five minutes. The team receives a certain amount of fake money (points) if the site is up, and a small bonus if the page loads within 0.5 seconds. If the site is down, money is deducted from the team. This simulates a typical website business model: you make money only if the site is up. Faster sites are more appealing and profitable. Customers react to down or slow sites by switching to competitors; thus, those lower-performing sites lose money.

The challenge lasts multiple weeks, during which the students learn to perform common web-operation tasks such as software upgrades, bug fixing, task automation, performance tuning, and so on. Inspired by Netflix's Chaos Monkey,[2] individual hosts are randomly rebooted to test the resiliency of the overall system.

The Uptime Challenge enables students to understand IT's value to the organization and to identify the IT processes that impact this value and permit continuous improvement. As a result, students are more motivated and better able to assess their own work. This leads to improved engagement and fosters more practical class discussions. It creates a direct feedback loop between a student's actions and the value they create. Most importantly, it better prepares students for the real world.

4. Be immersive from the start. IT projects usually involve some kind of legacy system. The most apt analogy is being asked to change the tires on a truck while it is being driven down the highway.

Software engineers spend more time reading other people's code than writing their own. We evolve existing systems. Green-field or "fresh start" opportunities are rare. Many people I have met have never been in a situation where they designed a new network, application, or infrastructure from scratch. Why can't education better prepare students for this?

Could something like the Uptime Challenge be introduced even earlier in the educational process?

Perhaps on the first day of class students should be handed not only copies of the syllabus, but also the username and passwords to the administrative control panel of a working system. Instruction and labs could be oriented around maintaining this system. Students would have their own wikis to maintain documentation and operational runbooks.

Each student would have their own working system, but I suggest that every few weeks students be randomly reassigned to administer a different system. Seeing how their fellow students had done things differently would be educational. Also, the best way to learn the value of a well-written runbook is to inherit someone else's badly maintained runbook.

Institutions are developing more immersive educational strategies. In cooperation with industry, Bossier Parish Community College[a] has created an Associate of Applied Science in Systems Administrator degree, which is highly immersive and covers core DevOps principals, including automation technologies (infrastructure as code and software-defined networking), Lean/KanBan ideas, cloud fundamentals, and even DevSecOps.

a https://www.bpcc.edu/catalog/current/technologyengineeringmathematics/aas-system-administration.html

Conclusion

Education should seek to normalize best practices from the start. Working outside these best practices should be considered a bug. Students should not struggle to learn best practices after graduation, and they should be shocked if potential new employers do not already have these practices in place.

Both IT and CS curricula could be structured to be more immersive, as immersive education more reliably reflects the real world. It prepares students for industry and better informs the research of those who choose that path. Seeing the forest, and then understanding the trees, helps students understand why they are learning something before they learn it. It is more hands-on and therefore more engaging, and lends itself to gamification.

Our first experiences cement what becomes normal for us. Students should start off seeing a well-run system, dissect it, learn its parts, and progressively dig down into the details. Don't let them see what a badly run system looks like until they have experienced one that is well run. A badly run system should then disgust them. C

Related articles on queue.acm.org

Undergraduate Software Engineering
Michael J. Lutz, et al.
http://queue.acm.org/detail.cfm?id=2653382

A Conversation with Alan Kay
http://queue.acm.org/detail.cfm?id=1039523

Evolution of the Product Manager
Ellen Chisa
http://queue.acm.org/detail.cfm?id=2683579

References

1. Begnum, K. and Anderssen, S.S. The Uptime challenge: A learning environment for value-driven operations through gamification. *Usenix J. Education in System Administration 2*, 1 (2016); https://www.usenix.org/jesa/0201/begnum.
2. Tseitlin, A. The antifragile organization. *Commun. ACM 56*, 8 (Aug. 2013), 40–44.

Thomas A. Limoncelli is a site reliability engineer at Stack Overflow Inc. in NYC. He blogs at EverythingSysadmin.com and tweets at @YesThatTom