Engineering for Chaos: Lessons Learned from COVID-19

Steven Fraser, Innoxec, Santa Clara, CA, USA
sdfraser@acm.org

Dennis Mancl, MSWX Software Experts, Bridgewater, NJ, USA
dmancl@acm.org

ABSTRACT

The global COVID-19 pandemic has transformed the way we live, learn, and teach – impacting both "how we learn" and "what we learn." Software system resilience has emerged as a critical concept, a departure from historical system objectives obsessed with high performance. In practice, there are many situations when development focused on efficiency, creates a system that is not very resilient. Fortunately, some technology companies have prioritized stability and availability over efficiency in order to deliver to customers a more consistent experience. Governments also value resilience to reliably serve their communities in the face of crises like cyber hacking and COVID-19. System resilience is a topic often neglected in computer science curricula. This paper reports on a recent virtual ACM SPLASH-E Education Symposium panel session held in November 2020 that discussed resilience, efficiency, and the impact of COVID-19 on computer science education. The panel featured Steven Fraser (panel impresario) and panelists Rebecca Mercuri, Landon Noll, Aleš Plšek, and Moshe Vardi.

Categories and Subject Descriptors
C.4 **[Performance of Systems]**: Reliability, availability, and serviceability; K.3.2 **[Computers and Information Science Education]**: Computer science education K.4.3 **[Computers and Society]**: Computer-supported collaborative work

General Terms
Management, Economics, Reliability

Keywords
Resilience, efficiency

1. RESILIENCE TO COMBAT CHAOS

Since March 2020, the world has been changed by the pandemic, and all of society has been struggling to improve the resilience of the systems that support our economy, employment, education, communication, and everyday life. As part of the online virtual ACM SPLASH 2020 conference in November 2020, the Educators session (SPLASH-E) sponsored a panel session titled "Engineering for Chaos: Lessons Learned from COVID-19." The panelists shared personal perspectives on the general problem of improving reliability and resilience, both of our computer systems and our personal lives, with examples of how our lives have changed as a result of the coronavirus crisis.

Resilience is a system property much in the news of 2020. SpaceX gave the name “Resilience” to their first human staffed space capsule as a way of promoting the reliability and flexibility of their commercial space launch system. The resilience of election systems to support vote-by-mail have been put to the test in the 2020 US presidential election. The year 2020 has seen increased scrutiny of the resilience of medical supply chains, the processes used to triage pandemic victims, and the development, trials, and distribution of effective vaccines and treatments.

Resilience is the ability of a system to continue to operate successfully in the presence of unplanned failures – it is an increasingly important topic in software systems given the current public health crisis. Unfortunately, many software systems are not very resilient, and resilience is often underappreciated by developers and product managers. The panelists explained four different viewpoints on the resilience crisis.

First, software developers are influenced by engineering and economic forces that lead them to neglect resilience as they try to optimize system performance and efficiency. Second, software designers often undervalue resilience because they believe that resilience is only important for extreme “corner case” situations, not for everyday systems. Third, resilience properties in a big system are often hidden by system complexity, so a lack of resilience may not surface until disaster strikes. In this situation, resilience can be improved if developers try a promising set of testing approaches (chaos testing and resilience experiments), where developers perform extra system stress testing. These automated experiments run through a series of tests that trigger a system’s key failure and recovery scenarios and collect information about the impact of partial failures. Fourth, staffing and training shortfalls may result in a loss of system resilience, particularly when experienced staff leave a company through downsizing or retirement. Staff reductions have become a serious problem in today’s software industry because there are fewer mentoring opportunities for new development staff to learn from the previous generation of experts.

The panelists shared their observations on how well we are dealing with today’s chaos in education and industry.

2. RESILIENCE VERSUS EFFICIENCY

Moshe Vardi (Rice University) has spoken and written throughout 2020 on the dichotomy between efficiency and resilience [1][2]. Moshe explained that there are engineering and economic forces that drive systems developers to focus on efficiency. But developers who focus solely on efficiency also need to consider multiple national and world crises – the current chaos that should push developers to try for more flexibility, adaptability, and resilience. Moshe specifically mentioned four crises in 2020: public health (the COVID-19 pandemic), economic (joblessness and recession triggered by the pandemic), social (social equality movements such as Black Lives Matter), and political (increased political polarization in the wake of a contentious 2020 Presidential election campaign in the United States).

Moshe cited a March 2020 Wall Street Journal article on the pitfalls of efficiency written by William Galston of the Brookings Institution [3]. Galston explained that although businesses always look at

efficiency as an “economic virtue,” efficient systems often have serious weaknesses. Galton’s article asks, “What if the relentless pursuit of efficiency, which has dominated business thinking for decades, has made the global economic system more vulnerable to shocks?” Moshe went on to explain how the attempts to optimize a system can reduce resilience: “When we are efficient, we try to be optimally adapted to the [current] environment. But resilience means we can adapt to rapid changes in the environment.”

Moshe described the tragedy of the early response to COVID-19 as the result of over-focusing on efficiency. Hospitals were deluged with COVID patients, and as a result had inadequate supplies of Personal Protective Equipment (facemasks, plastic gowns, and other gear used by medical professionals treating infectious patients). Moshe blamed some of the chaos on cost efficiency as a consequence of healthcare providers’ focus on cost containment. Under those constraints, medical facilities kept their staff size lean, emphasized outpatient care, and reduced stockpiles of unneeded supplies. This frugal economic approach left many hospitals unprepared for the first surge of COVID-19 infections, and it led to chaos in nursing homes that did not have the equipment and staff to meet the crisis [4]. Moshe lamented the consequences: “In many countries, a huge fraction of the COVID deaths have been in nursing homes. And you see the same thing: it was efficiency at the expense of resilience.” Lean supply chains, just in time manufacturing, and other modern innovations may have reduced resilience of the economy [5].

Are there any big software systems that have been truly resilient? Moshe points to the Internet, which has performed heroically in 2020 as many business and schools moved their daily work online. By design, the Internet is a resilient communications infrastructure. The Internet has enormous built-in redundancy, and it has continued to run smoothy in a stressful time. The Internet has also evolved to provide support for new services, from social media to video meetings. The lesson of resilience through redundancy can be applied to other problems, Moshe added. “In finance, how do you make sure the banks are resilient? They must have redundant capital. It’s very inefficient, but it gives you resilience.”

3. RESILIENCE FOR SURVIVAL

Landon Curt Noll (planetary astronomer) pointed to personal responsibility as a factor in assuring a high level of system resilience. Landon drew on his expertise in system security and reliability to explain how resilience is related to security. “Resilience is part of security. It’s not just a convenience.”

Landon suggested that the reactions of many individuals and companies to the pandemic crisis could raise new security issues as more activities move online. Landon described how perimeter security and firewalls for corporate data centers are weakened: “We drill in all kinds of tunnels because everyone is working from home. Soon you find that this thing you thought was a wall of Troy is actually Swiss cheese, nice and tasty for the people who want to eat into your data center.”

Landon’s main point was the importance of resilience in everyday life: “There is an unfortunate assumption that this resilience stuff is only for special circumstances, that for everyday stuff you don’t need to be resilient.”

Landon gave an example of everyday resilience: data backups. Landon's personal backup policy is to ensure that there are three copies of the data on his personal devices in at least two locations, and that he has tested the backups. He observed that if we practice what we preach, then it is easier to make complex systems more resilient, be it a stock market, an election system, a deep space probe, or pandemic response.

The panelists discussed the difference between "unknown unknowns" (unpredictable events that could affect system resilience) and more mundane "predictable events" (future failures that developers sometimes ignore for time and space efficiency reasons). Y2K is an example of a predictable event. In the 1990s, significant effort with software updates was required to ensure that computer systems handled dates correctly. Did we learn anything from Y2K?

Moshe pointed out that economics still seems to be more important than resilience, even when you can predict a future crisis. "Why do people refuse to believe in climate change? Because if you believe in climate change, it is going to cost a lot of money to prepare for it. And people don't want to spend the money." Landon liked building resilient systems as a matter of habit. "I think part of it is practicing what we preach and doing it in everyday life. If you start doing it there, it is easier to apply to more complex systems, such as a stock market or an election system."

4. RESILIENCE AND LEARNING

Rebecca Mercuri (Notable Software) has worked in a wide range of industry and academic roles. Her current professional work is in the field of computer forensics, and she is also an expert in security for voting machines and elections.

Rebecca reflected on the just-concluded presidential election in the US. Some people had predicted chaos, but the process of casting and counting ballots was fairly smooth. She explained that election security and election resilience has been the product of a 30+ year effort – a citizens' movement that started with a small cadre of computer scientists who complained about the poor security design of the first generation of electronic voting machines in the 1980s. She asserted that in the opinion of the experts, these early electronic voting machines "were just junk." None of these experts was thinking about a pandemic. But they were thinking about how we could make these voting systems better.

By the early 2000s, many of these computer scientists had coalesced around the proposal to replace electronic voting machines without audit capabilities with voter verified paper ballots [6]. The proposal gained traction over time, and by 2020, most US states had systems in place to process paper ballots efficiently and accurately. The November 2020 US election wasn't perfect, but every state was ready to process a tidal wave of mail-in ballots, and 100 million people voted using hand-prepared voter verified paper ballots.

However, Rebecca continued to worry about the correctness and security of election systems. Most election systems depend on proprietary code, so governments are unable to perform adequate security audits. Rebecca has included analysis of voting systems in her university classes on forensics,

to show the potential vulnerability of source code. She asks her students to experiment with an open-source voting system: first they load the system into a test environment, then attempt to hack it.

Other panelists joined the discussion of how software developers can learn more about resilience. Should resilience be part of the undergraduate computer science curriculum? Which existing courses should include resilience as a topic? Giving the example of bubble sort as inherently more resilient, Moshe hoped that computer science education would go beyond the standard "analysis of algorithms" material: "It's always about run time, memory consumption, it's never about how resilient the algorithm is." Moshe observed that knowledge is fleeting and often lost when experienced developers retire.

Rebecca and Landon expressed opinions about "what we teach" and "what we learn" in computer science. Rebecca said that she has learned many practical things from experienced mentors throughout her career, but today, the combination of retirements and downsizing has reduced the amount of intergenerational learning. This learning cycle is an important input into system resilience. Landon lamented that too many students have focused narrowly on developing programming expertise at the expense of breadth and "out of the box" thinking. Landon explained that when he interviews new graduates and early career candidates, he always asks "Tell me something you did in the university that was not the result of an assignment." Becoming a better C or Java programmer is more than just coding skills. He said he advises early career computer scientists to learn skills other than coding, for example, to play a musical instrument or to paint. By exercising your mind in other domains, you will become a better problem solver.

The panel moderator, Steve Fraser (Innoxec), added to Landon's comment. For Steve, collaboration skills are a key part of the education of a software researcher. When he interviews job candidates, Steve inquires about collaboration experiences. Steve's favorite interview question for new grad PhDs is: "I see you co-authored many papers. Let's chat a bit about one of them. On this paper, tell me about co-author number two." Most people are eager to talk about their relationship with their teammates. But it can be a warning sign if the candidate's response is "I always work alone – my co-authors didn't contribute."

5. RESILIENCE EXPERIMENTS

The fourth panelist, Aleš Plšek (Netflix), described his Netflix experience and their Resilience Engineering Team. Aleš's team develops and supports tools that enable Netflix software engineers to run chaos experiments and resilience experiments in production environments, with automated execution of these experiments [7]. These experiments uncover Netflix vulnerabilities and expand developer knowledge. Aleš added that because the system is so complex, key parts of the system's behavior only become visible by running resilience tests in production.

The goal of his team is to keep the system stable and available, even in the face of changing demand and features. Efficiency is important, but it isn't the primary focus. COVID-19 has increased demand for Netflix: many Netflix customers are isolated at home and streaming more videos. The value of a resilient and highly available system is obvious.

Resilience is part of the culture at Netflix. All developers are responsible for testing new features and changes, and then re-run resilience experiments to measure the impact of any changes. Aleš also explained the importance of "people resilience" based on a consistent set of company values. The two main values in Netflix [8] are "context, not control" (managers let teams make their own technical decisions) and "freedom and responsibility" (fewer rules but everyone is responsible for doing what is best for the company).

6. LESSONS LEARNED

Moshe expressed his disappointment with the current "hybrid model" of education found in US schools and universities. Educational institutions had to do their best in spring 2020 to react to the first wave of the pandemic. But it should have been possible to do more in-person education in the fall. Moshe complained: "I consider this a shameful moment for higher education, because it was driven not by pedagogical values but by financial values." But Moshe believed that students are under a lot of stress, and he will be understanding: "This is not a semester where I am going to insist on being a tough grader."

Rebecca shared mentoring experiences – in spring 2020 she coached students who were writing scripts to create COVID-19 dashboards for a high school competition. She helped them to think out of the box when they displayed their data: "We all realized at the same time that we should be using a log scale."

An audience participant, Karl Stolley (Illinois Institute of Technology), joined the discussion and shared some of his teaching experiences. Karl explained that instructors need to demonstrate resilience to their students. Resilience is a principle that should guide day-to-day work as a professional. He presented two different ways that he shared this principle with students: student projects with evolving requirements (to increase chaos) and live coding demonstrations by the instructor (to show how even smart people make mistakes). Landon immediately agreed with Karl's points: "Giving your students twists and turns, and having them pivot, is something that is quite important."

Karl explained one approach to giving students experience in "design for resilience and flexibility." He would always have one group project where he announced to the students at the beginning that he would "modify one project requirement" at some point before the deadline. Two weeks before the deadline, he would tell them "this old requirement, it's gone, here's the new one instead." Students had to adjust their designs to match the modified requirements. Although this was not a popular approach among current students, Karl says that he had some students come back years later and tell him that this exercise was helpful to them, because in the real world, requirements churn was an essential part of their jobs.

Elisa Baniassad (University of British Columbia), SPLASH-E co-chair, pointed the audience to a famous ICSE paper on computer science education that described similar strategies to get students to think out of the box: "Twenty Dirty Tricks to Train Software Engineers" by Ray Dawson [9].

Elisa noted that there are two challenges to using these "dirty tricks." First, they are difficult to use for a large class with hundreds of students. Grading becomes more difficult and more students may need

one-on-one assistance. Second, it can be difficult to provide the same educational experience for all students, to ensure parity across the entire class and judge students fairly. Moshe agreed with this point, and he claimed that if he followed Karl's approach at his university, students would probably file complaints with the university administration. But everyone agreed that students need to be exposed to the challenges of changing requirements and developing flexible designs.

Karl explained another unusual teaching practice: he runs live coding demonstrations for his students. "I love to live-code in class because I will inevitably make a mistake. And sometimes it's a mistake that will end class." He would model the behavior of a good resilient developer: consult the API documentation and work through the code to find the problem. He would sometimes resort to dismissing class early with a promise to record and post a 10-minute video from home with the correct code. "It's a risk as an instructor, because you look like an idiot for at least a little while." Students must learn that "reading the documentation" is a key part of their job. And it is not essential to memorize everything, because a real expert is someone who knows where to look things up.

Moshe concluded the panel with an observation about what we need to do next. He will start to think about the educational angle of resilience: "If we want the world to become resilient, we need to start with students. They will build the systems of tomorrow."

7. SUMMARY

Remote work and remote learning have accelerated the demand for resilience, both in computer systems and in our day-to-day lives. Resilience will be an increasingly important subject for software developers and educators. The panel proposed four actions for the future:

- Developers must take a balanced approach between efficiency and resilience.
- Resilience is not just for exceptional systems, it is our professional responsibility to make everyday operations resilient.
- Resilience Engineering techniques should be considered as part of project development plans, and reliability experiments will be especially valuable when building highly available systems.
- Companies and universities should address some learning gaps: increasing intergenerational knowledge sharing, improving the effectiveness of distance learning and collaboration, and encouraging out of the box thinking.

As it turned out, 2020's pandemic has helped all of us to recognize the value of resilience in the systems that will meet future environmental and public health challenges, such as those posed by climate change and new viruses.

8. REFERENCES

[1] Vardi, M. 2020. Efficiency vs. Resilience: What COVID-19 Teaches Computing. *Communications of the ACM* 63, 5, (Mar. 2020), 9. DOI= https://doi.org/10.1145/3388890

[2] Vardi, M. 2020. Lessons from COVID-19: Efficiency vs Resilience. ACM TechTalk Webinar, Oct. 14, 2020, https://learning.acm.org/techtalks/covid

[3] Galston, W. 2020. Efficiency Is Not the Only Economic Virtue. *Wall Street Journal*. March 10, 2020.

[4] Terlep, R.G. and Evans, M. Why Did Covid Overwhelm Hospitals? A Yearslong Drive for Efficiency! 2020. *Wall Street Journal*. September 17, 2020.

[5] What Covid-19 toilet paper shortages tell us about supply chains. *Financial Times*. June 7, 2020, https://www.ft.com/video/6e5acf3e-511b-48d1-948c-ff7c94f3ba1b

[6] Mercuri, R. 2002. Explanation of Voter-Verified Ballot Systems. *The Risks Digest* 22, 17, (July 24, 2002). http://catless.ncl.ac.uk/Risks/22/17#subj4.1

[7] Basiri, A. et. al. 2016. Chaos Engineering. *IEEE Software* 33, 3, (May-Jun. 2016), 35-41. DOI= https://doi.org/10.1109/MS.2016.60

[8] "Netflix Culture," https://jobs.netflix.com/culture

[9] Dawson, R. 2000. Twenty Dirty Tricks to Train Software Engineers. Proc. 22nd International Conference on Software Engineering (Jun. 2000). 209-218. DOI=https://doi.org/10.1145/337180.33720

This paper appeared in the April 2021 issue of ACM SIGSOFT Software Engineering Notes (Vol. 46, No. 2, pp. 25-27, DOI: 10.1145/3448992.3448998)