60 Seconds with ... Nancy Leveson



Nancy Leveson is Professor of Aeronautics and Astronautics at MIT. She is an elected member of the National Academy of Engineering (NAE). Prof. Leveson conducts research on the topics of system safety, software safety, software and system engineering, and human-computer interaction.

She has received many awards for her work including the IEEE Medal for Environmental and Safety Technologies, the ACM Allen Newell Award for outstanding computer science research and an AIAA Information Systems Award for "developing the field of software safety and for promoting responsible software and system engineering practices where life and property are at stake."

She has published over 400 research papers and is author of three books, "Safeware: System Safety and Computers" [1] published in 1995 by Addison-Wesley, "Engineering a Safer World" [2] published in 2012 by MIT Press, and "An Introduction to System Safety Engineering"[3] published in 2023 by MIT Press. She consults extensively in many industries on the ways to prevent accidents.

What first attracted you to working in System Safety?

To answer this question, I need to start before that time. I first earned a B.A. in Mathematics and then an M.S. in management (operations research). Then I joined the working world as a system engineer for IBM. I didn't like working in industry, so I quit and hitchhiked one and a half times around the world for a couple of years until I could decide what I wanted to do. After returning, I knew that I could not avoid the normal working world without a doctorate so I started a Ph.D. program in Cognitive Psychology. In the end, I got a Ph.D. in Computer Science, specializing in formal methods for software, but I remained interested in human cognition and human factors. At that point I had never heard of safety engineering.

Immediately after finishing my degree, my dissertation was plagiarized. I was pretty bummed out. I started my first academic position, but I had no idea what I wanted to do beyond knowing that I did not want to continue to work on the same topic (database integrity). Within a week of starting my academic position, I got a phone call from someone at Hughes Aircraft Company about a torpedo they were building for the U.S. Navy.

The torpedo had 15 microprocessors on it, which was very unusual at that time. Previously, computers had primarily been used for computation only, while control was done by humans or by simple electronic PID (Proportional–Integral–Derivative) controllers. The engineers were very nervous about the potential for this "heat-seeking" torpedo to turn around 180 degrees and hit the ship that launched it.

They told me that this was a "software safety problem." I told them I had never heard of software safety, but it sounded like a software reliability problem. They were adamant about it not being reliability, but rather involved safety.

This was the first time I had heard about safety engineering. As you can imagine, I wasn't able to help them much with the torpedo, but I learned an enormous amount about safety engineering from these exceptionally qualified engineers.

When the torpedo was finished, they took it to a torpedo testing area and test fired it. Every time it came out of the torpedo tube, it turned itself off, went to the bottom of the ocean, and just laid there. I said, "well, it's safe." They replied that the Navy did not want to pay for this safe torpedo. This was the first time I started to truly understand that safety and reliability were different system properties and, in fact, they could be conflicting—increasing one did not necessarily increase the other and in fact could decrease it. Forty-five years later I am still trying to explain this distinction to computer scientists and to many engineers.

To make the torpedo useful, the engineers took off safety devices, one by one, until it was acceptably reliable and safe. This approach didn't seem great to me, especially as the amount of software and complexity was surely going to increase in the future. I decided I would take six months to figure out the solution and then decide what area of research I really wanted to work in. It took a little longer than I expected (20 years) to find a successful solution and to convince others that it worked.

Software engineers have always, especially at that time, focused on assurance after the design is complete, so I started with trying to enhance traditional test and assurance activities and to extend the standard hazard analysis techniques that existed at that time (FTA, ETA, HAZOP, FMECA, Bow Tie, etc.).

It quickly became clear that after-the-fact assurance was not practical nor cost-effective as software increased in size. Today, embedded control software can have tens of millions or even hundreds of millions of lines of code. Finding that there is a problem after the system and software has been created is too late—the options available after the design is complete are limited, very expensive, and not very effective.

Instead, safety needs to be designed into systems and software from the very beginning. Focusing on assuring it at the end is a dead-end approach and leads to an unnecessarily and often unacceptable level of risk. In addition, the pressure is great at the end to find any argument (whether bogus or not) to demonstrate that the system is safe "enough." The focus is on making a convincing argument that what you have is acceptable rather than on increasing safety in the design and the system.

Besides starting too late in the life cycle, the other problem I encountered in my early work is that traditional hazard analysis techniques were developed to analyze the relatively simple systems existing at the time, which did not include software or cognitively complex human activities. The traditional techniques are based on the assumption that accidents are primarily caused by hardware component failure or by humans not following prescribed procedures. They also assume that "software failure" is just like hardware failure.

But the world of engineering has changed and new causes of losses are becoming increasingly important, many due to flawed software requirements and system design as well as human inability to cope with increasing complexity.

The old hazard analysis techniques, which assume that accidents are caused by component failure alone, are not very useful on these new systems. The focus on component failure leads to techniques that can identify component failures leading to hazards, but nothing else: They cannot be expanded to identify the very different causes of accidents today. That fact doesn't seem to prevent many people from using those techniques, it is just preventing them from providing much useful information to the system designers or assurers. This fact seems to be ignored by those filling out endless forms and tables of information about random component failures and computing their probability while ignoring design flaws.

I believe that the only way to make real progress is to change to solution paradigms that solve the problems of today, not those of yesterday. To accomplish this, I decided that we needed to start from System Theory, and I created a new model of accident causality (STAMP) that focuses on control rather than failures and reliability. On that foundation, very different types of hazard analysis techniques can be created, including STPA (System-Theoretic Process Analysis) and CAST (Causal Analysis based on System Theory). These new analysis techniques seem to be working well and are being used widely in most industries, especially on the most complex and potentially dangerous systems being built today.

What aspect of your career are you most proud of?

I'm clearly proud of my contributions to system safety engineering, but another important aspect has been my work throughout my career mentoring and assisting women to be successful in a profession primarily focused on and catering to men.

What has been one of the key challenges you've experienced in your career?

Being female has far and away been the biggest challenge. The second has been to get people to stop just doing what they have always done—although it no longer works—and to look at new ways of approaching problems. We need more scepticism about what is considered "'truth." I tell my students "You are here not to worship what is known, but to question it" (Jacob Bronowski, The Ascent of Man).

If you could change one aspect of system safety, what would it be?

We need to stop relying on probabilities as the goal in safety engineering or even as a way to reach the goal. They may provide a way to help management make decisions, but the probabilities on which those decisions are based are always wrong so the resulting decisions are not necessarily very good. I've read hundreds of accident reports, and almost every one of them contained a Probabilistic Risk Assessment (PRA) that had showed the accident could not happen—usually exactly the way it did happen. The complacency generated by the numbers was often a major contributor to the loss. There have been only two scientific evaluations of PRA that I have ever found. Both found it to be extremely inaccurate.

Beyond not being accurate, the results do not provide any information to engineers today to design systems to be safer. The risk calculation can be improved by adding a lot of redundancy, but that doesn't have any impact on safety when the problem is a design flaw, the most prevalent contribution today of software and humans to accidents. We need to design systems to be safe, not just convince ourselves that they are safe by producing computations that cannot be validated but make us feel better about the potential risk.

What are the most infuriating misconceptions about the discipline that you still encounter?

Most misconceptions arise from the fact that engineering is very different than it used to be and that old solutions no longer work. Here are a few:

Misconception 1: Risk should be defined as likelihood combined with severity. In fact, likelihood is impossible to determine for most hazards and losses and even severity is more difficult to determine than is usually assumed. The problems in defining severity arise from heuristic biases [4]. There are alternative, improved ways of defining risk, but they require a paradigm change.

Misconception 2: Most accidents are the result of component failures.

Misconception 3: Software "fails" and thus reliability concepts can be applied to software.

Misconception 4: Safety can be assured after the design phase on complex systems today.

What advice would you give to younger people seeking to build a career in systems safety?

- Don't work on "toy" problems. Everything works on such examples, particularly if you ignore all the non-straightforward ("ugly") parts of the problem. Work on real problems at real scale. I make my students apply their new solutions to real systems in their theses and dissertations. The proprietary parts of the design do not need to be made public, but validating new approaches and ideas on real systems is necessary before we encourage people to use those approaches. The important problems in very large systems are different than those in smaller ones.
- Start from the problem, not from solutions. One of my very early papers—that I regret ever writing—applied Petri nets to hazard analysis. Why did I choose Petri nets? I knew them and liked using them. Since that time, I don't know of any real system on which Petri nets have been or could be used for hazard analysis. Amazingly, however, this paper is one of my most highly cited, and I still come across government handbooks, etc., that recommend the use of Petri nets for hazard analysis. They all cite that silly paper I wrote 40 years ago. People are writing lots of conference papers on using Petri nets for safety analysis, citing my paper as a reference. Do not start by applying previous solutions, that you know or like, to a different problem. Instead, start with understanding the problem you are trying to solve. Use that information to find solutions tailored to that problem.
- Learn the history of engineering and of safety engineering—it goes back at least 150 years— and determine what still works and what never worked. Stop reinventing the same things (often with new names) that did not work in the past. We don't make progress that way. At the same time, realize that the world of engineering is changing at lightning pace: We should not robotically keep doing what was done in the past just because it used to work and we know how to do it.

We should not robotically keep doing what was done in the past just because it used to work and we know how to do it.

How do you see the rise of Artificial Intelligence affecting the future of safety engineering?

There are some important engineering problems that will be solved usefully by machine learning and LLMs, but safety is not necessarily one of them

We are living in one of the periodic AI hype bubbles. I've lived through several of them. In the 90's, "expert systems" were the focus of almost all engineering research. They were touted as the solution to all important engineering problems. How many engineering researchers today are focused on using expert systems?

There are some important engineering problems that will be solved usefully by machine learning and Large Language Models (LLMs), but safety is not necessarily one of them. Their use is more likely to increase risk than to decrease it and to create new types of risk. I don't see any way to ensure safety in software when we cannot predict or control the outputs and where the software periodically "hallucinates" (i.e., lies). It may be possible to design the parts of the system around the AI components in some cases to protect against potentially dangerous AI behavior. But I don't see any obvious general solution.

People are also talking about using AI to perform safety engineering tasks (e.g., doing hazard analysis), which seems unethical to me except for the most trivial tasks.

References

[1] Nancy Leveson, Safeware: System Safety and Computers, Addison Wesley, 1995

[2] Nancy Leveson, Engineering a Safer World, MIT Press, 2012.

[3] Nancy Leveson, An Introduction to System Safety Engineering, MIT Press, 2023.

[4] Nancy Leveson, Improving the Standard Risk Matrix, Part 1, unpublished white paper, downloadable from http://sunnyday.mit.edu/Risk-Matrix.pdf