

Deception and Intuition in Software Engineering

Phil Laplante^{1b}, IEEE Fellow

Deception poses threats but also opportunities in software engineering, and we must learn to recognize and manage these. Our intuition can save us from these threats and unlock the opportunities, but takes time and practice.

I have come to realize that much of my writing over the years reflects a (healthy) distrust of “reality.” My distrustful nature comes from my background in working on mission-critical software systems. I also have a paranoid disposition from my Sicilian mother’s influence. But skepticism is healthy, particularly in software engineering, where uncertainty abounds and not everything is what it seems to be. So, what does the smart software engineer do? Rely on intuition borne of experience.

Digital Object Identifier 10.1109/MC.2023.3328508
Date of current version: 5 January 2024

DECEPTION, UNCERTAINTY, AND INTUITION

Nature demands that we live with danger: violent weather, accidents, disease, human and other animal threats. We should expect and prepare for all of these. But other forms of danger are more uncertain: the lurking danger of disguised and hidden threats. Intuition is often the only thing that saves us from these hazards since they are difficult to prepare for.

In the animal kingdom certain species have evolved to use the uncertainty of danger as a survival strategy. This evolutionary strategy, called *mimicry*, is found in many species. For example, deep-sea anglerfish have a bioluminescent lure that dangles in front of their mouths, resembling prey to draw unsuspecting victims. The alligator snapping turtle has a tongue-like appendage that

DISCLAIMER

The author is completely responsible for the content of this article. The opinions expressed here are his own.



resembles a worm. Fish are attracted to this “worm” and become prey when they venture too close to the turtle’s mouth. There is defensive mimicry too: for example, the dead leaf butterfly, native of Southeast Asia, has wings that resemble dried, dead leaves, making them indistinguishable from actual dead leaves. There is also an order of insects that resemble dead branches or sticks, and there are other examples.

Human interactions commonly employ deception. Con artists and other criminals, Internet sites, movies and television programs, advertising (and sometimes, fake news) all use deception. In the case of criminals and con artists, our experience and intuition can protect us. Similarly, with the ultrarealistic computer-generated images found in movies and other media, our intuition informs us that some situations can’t be true. For example, one widely distributed video shows Bruce Lee using num-chucks instead of paddles to defeat a professional in a ping pong match. This never happened, but the video is so realistic that you can only intuit the fraud from an understanding of the game of ping pong and the use of num-chuks, a rare combination of skills.

As software engineers, we also deal with uncertainty and deception in our practice, and we must learn to recognize it and cope with it or use it to our advantage. Let’s take a look at some of these situations.

THE THREATS OF DECEPTION TO SOFTWARE ENGINEERS

Spoofing is a type of cyberattack in which a person or system deceives the victims by impersonating someone or something else, just as the alligator snapping turtle dangles its false worm in front of unsuspecting victims. There are various forms of spoofing, such as e-mail, IP address, and domain

name system. Spoofing attacks can have various motivations, including stealing personal information, conducting financial fraud, launching cyberattacks, or spreading malware.

Another form of deception-based threat is *malware*, including viruses, worms, trojans, ransomware, spyware, etc. These all pose unique threats to

plagiarized, leading to a formal investigation and eventual proof of the crime.

Finally, a curious form of deception that I have hypothesized is called *surreptitious problem solving*. This spoof involves using large-scale game-playing networks to trick players into solving complex problems using brute force for nefarious purposes. Surrep-

As software engineers, we also deal with uncertainty and deception in our practice, and we must learn to recognize it and cope with it or use it to our advantage.

each system and their data. But recognize that malware exists because it works. Various forms of malware deceive victims who are either inexperienced or naïve, but also experienced persons who ignore their intuition.

Software plagiarism deals with the wholesale copying of portions of code, either identically or in disguised form, such as using variable name substitution algorithms, insertion of dummy code, and juxtaposition. Design or architectural duplication centers on the copying of a high-level structure of a program, such as a unique modular decomposition, in the case of procedural code, or in object-oriented design, class definitions, inheritance hierarchy, and method invocation sequence. Other aspects of a software program that could be copied include unique algorithms, elements of the user interface, and any other proprietary functionality that might be unique to the application domain. The deception of plagiarized software is hard to defeat but tools for clear identification of the lineage or origin of software (whether in original, rewritten or disguised form) are gaining more attention.¹ And I know of legal in cases where intuition suggested that software was

titious problem solvers are a crowdsourced approach to large, intractable problems. Think of Amazon’s Mechanical Turk but with a layer of deception. While surreptitious crowdsourcing through gameplay could be used to solve important societal problems, this has never been done as far as I know. But there are cases of surreptitious problem solvers used to defeat captchas, and I am sure that there are more to be discovered.² As someone once addicted to a particular free game app, my intuition told me that somehow these free clicks were being monetized in some way.

PURPOSEFUL DECEPTION IN SOFTWARE ENGINEERING

Sometimes deception in software engineering is desirable. Software engineers sometimes work on uncertain applications or disguised software. This is common in defense applications, where the engineer may work on some software components of the system but not know the total system purpose. For example, long ago I worked on navigation software for a military aircraft whose purpose was unknown to all but a few of the engineers on the project.

A digital twin is another form of deception. A digital twin is a virtual representation of a real-world thing and the associated process around within. Digital twins are used to help gain experience with the real-world thing

television programs and movies sometimes the doubles are so mismatched or badly disguised that the deception is comically obvious). Auto manufacturers use high-tech mannequins (ironically called *dummies*) to double

fall for a spoof or click a bad link that installs malware, even when your intuition tells you not to do so.

I find that young software engineers tend to lack intuition. As with the Bruce Lee ping pong game, often only experience can inform our intuition sufficiently. The best software engineers have expert knowledge and expert intuition. Perhaps that is why Ken Jennings, a software engineer, is the all-time *Jeopardy* champion of champions.

Personal security expert Gavin de Becker recommends that in dangerous situations we should trust our intuition, even when doing so defies convention and logic and experts' recommendations.⁶ Millions of years of evolution have made our brains powerful pattern-matching machines, helping us to recognize danger. Even with purpose-built AI we should trust our brain's pattern-matching ability over a computer or any expert, especially when the consequences could result in great harm to us or society. It is particularly in those instances where our "zero trust" brains tell us something is wrong that we must listen to them,⁷ You already use your intuition to override technology: When a navigation assistant tells you to drive into the lake or through a red light, you ignore that instruction because your intuition, borne from experience, tells you to ignore it.

Not every person's intuition is the same, however, due to the "*Rashomon* Effect." *Rashomon* (1950) is a celebrated Japanese film centering on the disparate viewpoints of the murder given by four diverse witnesses to the crime. Each viewpoint, which is truthfully told by each witness, is vastly different from the others. The different viewpoints are based on each witness' lived experience, unconscious bias, and stake in the outcome of the investigation. Therefore, their "truths," although authentic to them, are different, and self-serving.⁸ So, we should not judge when our intuitions contradict someone else's.

Digital twins are powerful tools, but trust in them must be tempered with reality, and intuition must override recommended actions in dangerous situations.

(in safety and at low cost) to improve qualities of that thing and its uses. For example, a digital twin of a production plant can be used to train operators, improve efficiency, and reduce power consumption. By definition, a digital twin is a fake, and the fake is only as good as its design allows it to be. Digital twins are powerful tools, but trust in them must be tempered with reality, and intuition must override recommended actions in dangerous situations.³

Significant advances in human-computer interfaces and computational power have led to another form of deception: the development of very realistic virtual worlds leading to virtual reality and a more fully developed virtual universe, often called the *metaverse*. Virtual worlds, such as Second Life, emerged from the gaming community, but virtual reality applications are now used in many domains for training, testing, inspection, and more. Virtual worlds are a deliberate form of deception, and if one is fully immersed in them, it takes some effort and strong intuition to detach from the illusion. I speak from experience playing certain virtual reality games.⁴


Recently, colleagues and I have suggested using proxy systems for testing critical artificial intelligent (AI) systems in dangerous and tricky scenarios. A proxy is a stand-in or substitute for something, but it is also a form of deception. Actors have stunt doubles for dangerous scenes (in older

for humans in crash tests. We already use proxies in other areas of software engineering: for example, simulated (proxy) data for personally identifying information during testing.

Similarly, we can use a noncritical AI system that had a similar (or identical) application, operational profiles, user characteristics, underlying AI algorithm, etc. If the "stand-in" is similar enough to the critical system, then we could test even the most dangerous scenarios. For example, a robot vacuum may have significant operational and implementation similarities to function as a testing proxy for certain scenarios for autonomous land vehicles, so we could test the robot vacuum in situations involving deciding between crashing or avoidance of pedestrians.⁵ I believe that it will be proven that the best way to do proxy testing is with a disguised proxy: that is, one in which the real system being tested is unknown to the tester. This approach can be used to remove any kind of tester bias.

TRUST YOUR INTUITION

Defending against spoofing attacks or malware typically involves implementing security measures, such as e-mail authentication protocols, network filtering, and encryption to verify the authenticity of communications and data. Detecting plagiarized software is difficult but possible with certain tools and protocols. But no amount of protection is absolute. Everyone can

My Sicilian mom would say the only people you can really trust are family. In our increasing virtual experiences, as we encounter increasing disinformation, deception, and fakes, I would add that your intuition is the only other thing you can really trust. Therefore, you should practice using your intuition, evaluate its accuracy, and find ways to improve it. Remember that good judgment comes from experience and experience comes from bad judgments. 

REFERENCES

1. P. Laplante, "Software labels," *Computer*, vol. 54, no. 11, pp. 82–86, Nov. 2021, doi: 10.1109/MC.2021.3102360.
2. P. A. Laplante, "Ender Wiggin played mafia wars too," *IT Prof.*, vol. 13, no. 4, pp. 6–8, Jul./Aug. 2011, doi: 10.1109/MITP.2011.60.
3. P. Laplante, "Trusting digital twins," *Computer*, vol. 55, no. 7, pp. 73–77, Jul. 2022, doi: 10.1109/MC.2022.3149448.
4. P. Faraboschi, E. Frachtenberg, P. Laplante, D. Milojevic, and R. Saracco, "Digital transformation: Lights and shadows," *Computer*, vol. 56, no. 4, pp. 123–130, Apr. 2023, doi: 10.1109/MC.2023.3241726.
5. P. Laplante, M. Kassab, and J. DeFranco, "Proxy verification and validation for critical autonomous and AI systems," in *Proc. IEEE 29th Annu. Softw. Technol. Conf. (STC)*, Oct. 31, 2022, pp. 37–40, doi: 10.1109/STC55697.2022.00014.
6. G. de Becker and T. Stechschulte, *The Gift of Fear*. New York, NY, USA: Dell Publishing, 1997.
7. P. Laplante and J. Voas, "Zero-trust artificial intelligence?" *Computer*, vol. 55, no. 2, pp. 10–12, Feb. 2022, doi: 10.1109/MC.2021.3126526.
8. P. A. Laplante and M. Kassab, *Requirements Engineering for Software and Systems*, 4th ed. New York, NY, USA: Taylor & Francis, 2022.

PHIL LAPLANTE is a computer scientist and software engineer, a Fellow of IEEE, and an associate editor in chief of *Computer*. Contact him at plaplante@psu.edu.

**SUBMIT
TODAY**

IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING

► SUBSCRIBE AND SUBMIT

For more information on paper submission, featured articles, calls for papers, and subscription links visit: www.computer.org/tsusc

