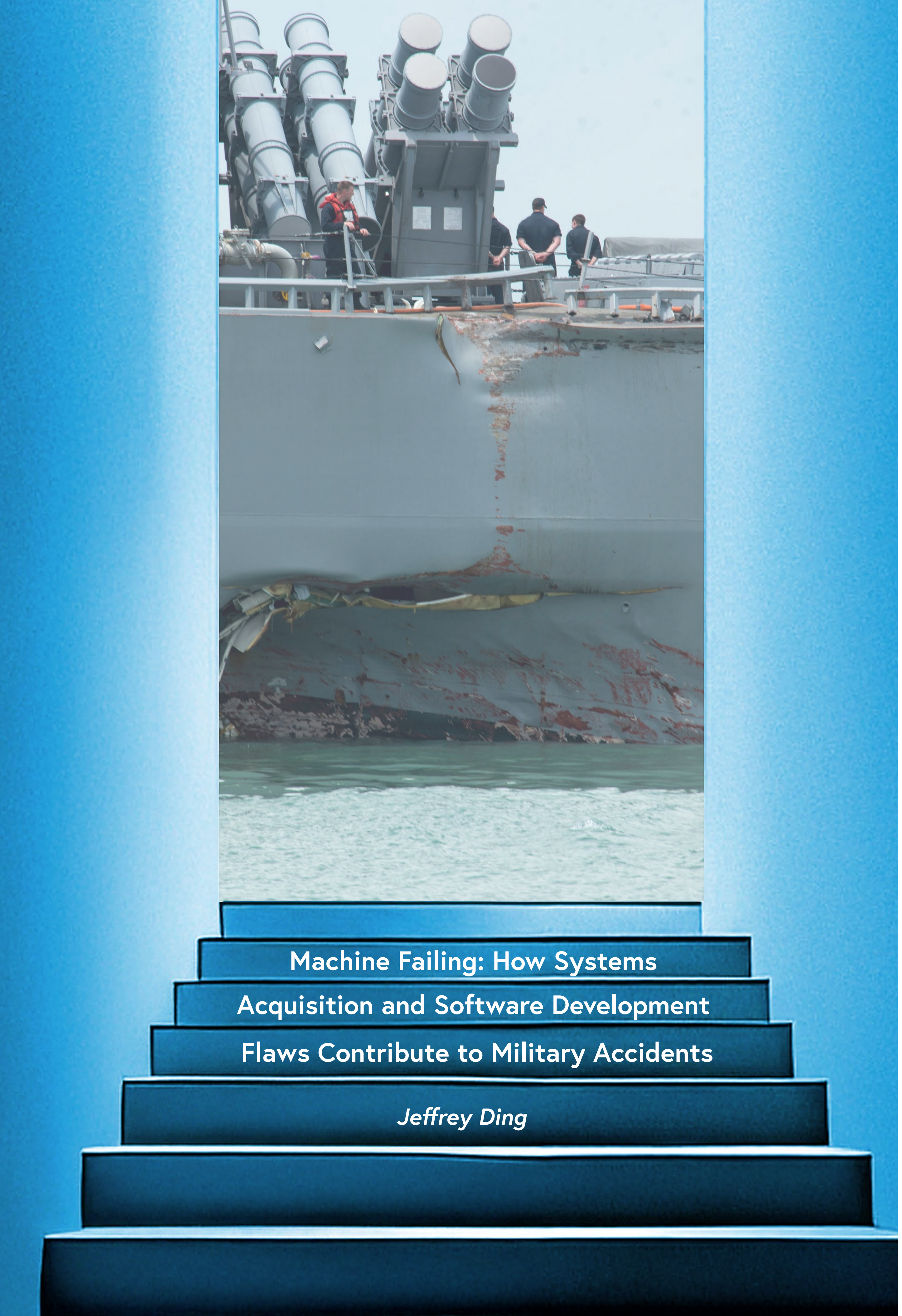


# Machine Failing: How Systems Acquisition and Software Development Flaws Contribute to Military Accidents

*Jeffrey Ding*

How does software contribute to military accidents? The stakes are high. During the Cold War, computerized early warning systems produced "near-miss" nuclear crises. In the future, military AI applications could fail with devastating consequences. To illuminate the causes of military accidents, existing studies apply "normal accidents" and "high reliability organizations" theories. While these frameworks are helpful, they neglect the military's system acquisition process, which often outsources software development to contractors and limits input from military end-users. By contrast, the software development lifecycle theory expands the causal timeline of accidents beyond decisions made on the battlefield to those made decades earlier in software design, serving as an antecedent account of how software contributes to military accidents. Illuminating dynamics overlooked by the two dominant approaches, this theory is supported by four cases: the 1988 USS Vincennes shootdown of an Iranian airliner; the 2003 Patriot fratricides; the 2017 USS McCain collision; and software upgrades in the 2021 Kabul airlift.

On Aug. 21, 2017, the USS John McCain crashed into an oil tanker near the Strait of Malacca, resulting in the death of 10 sailors and marking the Navy's worst accident in four decades. While the Navy initially blamed the incident on the McCain's crew, later investigations pointed out problems with the ship's navigation software.[1] In fact, the Navy's own review of the crash stated, "There is a tendency of designers to add automation ... without considering the effect to operators who are trained and proficient in operating legacy equipment."[2]

Safety-critical software systems come with high stakes. At present, maritime forces rely on navigation software like the one used on the McCain, and an accident at sea could be one of the most likely triggers of a military conflict between the United States and China.[3] During the Cold War, accidents involving computerized early warning systems produced numerous "near-miss" nuclear crises. In the future, military systems that incorporate new advances in AI and other emerging technologies could also fail, bringing devastating consequences.[4]

How does software contribute to the risk of military accidents? Existing scholarship on the safety risks of military technology systems draws on debates between normal accidents theory and high relia-

1 T. Christian Miller, Megan Rose, Robert Faturechi, and Agnes Chang, "The Navy Installed Touch-Screen Steering Systems to Save Money. Ten Sailors Paid with Their Lives," *ProPublica*, December 20, 2019, https://features.propublica.org/navy-uss-mccain-crash/navy-installed-touch-screen-steering-ten-sailors-paid-with-their-lives/.

2 U.S. Fleet Forces Command, "Comprehensive Review of Recent Surface Forces Incidents," Department of the Navy, October 26, 2017, https://news.usni.org/2017/11/02/document-navy-comprehensive-review-surface-forces.

3 Daniel Yergin, "The World's Most Important Body of Water," *The Atlantic*, December 15, 2020, https://www.theatlantic.com/international/archive/2020/12/south-china-sea-us-ghosts-strategic-tensions/617380/. On debates over whether conflicts can be "accidental," see Dan Reiter, "Exploding the Powder Keg Myth: Preemptive Wars Almost Never Happen," *International Security* 20, no. 2 (1995): 5–34, https://doi.org/10.2307/2539227; Andrew A. Szarejko, "Do Accidental Wars Happen? Evidence from America's Indian Wars," *Journal of Global Security Studies* 6, no. 4 (December 1, 2021), https://doi.org/10.1093/jogss/ogaa030.

4 Paul Scharre, *Army of None: Autonomous Weapons and the Future of War* (New York: W.W. Norton & Company, 2018); Paul Scharre, "Debunking the AI Arms Race Theory," *Texas National Security Review* 4, no. 3 (Summer 2021): 121–32; Michael C. Horowitz, Paul Scharre, and Alexander Velez-Green, "A Stable Nuclear Future? The Impact of Autonomous Systems and Artificial Intelligence," arXiv:1912.05291 [Cs], December 13, 2019, http://arxiv.org/abs/1912.05291.

bility organizations theory.[5] The normal accidents approach argues that the causes of accidents in highly complex and tightly coupled technological systems are deeply embedded in the systems themselves. Tightly coupled systems require centralized authority because small mishaps can rapidly escalate into major disasters, but problems in complex systems demand responses by local decision-makers who understand how the system works. The tension between these two imperatives results in unavoidable accidents.[6] Under normal accidents theory, accidents are inevitable in software-intensive military systems because they are very complex and tightly coupled. For instance, one expert on AI governance predicts that normal accident problems will be "particularly acute in military AI applications."[7]

The high reliability organizations literature, in contrast, posits that certain organizations can effectively manage the risks of hazardous technologies. Studies in this tradition emphasize the importance of organizational culture, such as deference to expertise and dedication to learning from failures, as well as flexible organizational structures that permit authority to be centralized and decentralized depending on the situation.[8] As evidenced by studies of the U.S. Navy's nuclear aircraft carrier community and submarine community, certain military organizations have demonstrated excellent safety records with complex, interdependent technology systems.[9] Scholars have proposed the high reliability organizations model as a way to manage the risks of autonomous weapons.[10]

To be sure, these two approaches have produced valuable insights on the causes of military accidents. Yet, the applications of normal accident and high reliability organizations theories to software-intensive military systems share a drawback: their scope of analysis is confined to the actions of military organizations only after software systems have been fielded. This means that they neglect the initial phase of software acquisition and development — when critical safety decisions are made. It also means that the existing literature does not account for the activities of the defense contractors that develop most of the military software.

Taken together, these considerations point to the importance of the military's software acquisition process. If this process limits feedback from military operators to end-stage testing and evaluation (e.g., linear "waterfall" models), when it is too late to change fundamental system designs, accidents are more likely. These acquisition pathways often yield confusing human-machine interfaces and limit adaptability to hidden vulnerabilities that emerge from operators using the system in the field. In this article, I propose software development lifecycle theory as an alternative way to explain how software contributes to military accidents.

By focusing on patterns of software procurement and development, software development lifecycle theory sheds light on causal factors that affect military accidents often overlooked by the normal accident and high reliability organizations perspectives. Consider,

5 The seminal text on this subject is Scott D. Sagan, *The Limits of Safety* (Princeton: Princeton University Press, 1993), https://press.princeton.edu/books/paperback/9780691021010/the-limits-of-safety. See Scott D. Sagan, "Rules of Engagement," *Security Studies* 1, no. 1 (September 1, 1991): 78–108, https://doi.org/10.1080/09636419109347458; Bart Van Bezooijen and Eric-Hans Kramer, "Mission Command in the Information Age: A Normal Accidents Perspective on Networked Military Operations," Journal of Strategic Studies 38, no. 4 (June 7, 2015): 445–66, https://doi.org/10.1080/01402390.2013.844127; Scharre, *Army of None*; Scott A. Snook, *Friendly Fire: The Accidental Shootdown of U.S. Black Hawks over Northern Iraq* (Princeton: Princeton University Press, 2000). On the organizational politics of accidents in autonomous weapons and cyberspace, see Avi Goldfarb and Jon R. Lindsay, "Prediction and Judgment: Why Artificial Intelligence Increases the Importance of Humans in War," *International Security* 46, no. 3 (February 25, 2022): 7–50, https://doi.org/10.1162/isec_a_00425; Michael C. Horowitz, "When Speed Kills: Lethal Autonomous Weapon Systems, Deterrence and Stability," *Journal of Strategic Studies* 42, no. 6 (September 19, 2019): 764–88, https://doi.org/10.1080/01402390.2019.1621174; Michael C. Horowitz et al., "Policy Roundtable: Artificial Intelligence and International Security," *Texas National Security Review*, June 2, 2020, https://tnsr.org/roundtable/policy-roundtable-artificial-intelligence-and-international-security/; Jon R. Lindsay, "Stuxnet and the Limits of Cyber Warfare," *Security Studies* 22, no. 3 (July 1, 2013): 393–94, https://doi.org/10.1080/09636412.2013.816122; Paul Scharre, "Autonomous Weapons and Operational Risk," Center for a New American Security, 2016; Jacquelyn Schneider and Julia Macdonald, "Looking Back to Look Forward: Autonomous Systems, Military Revolutions, and the Importance of Cost," *Journal of Strategic Studies* 47, no. 2 (January 24, 2023): 1–23, https://doi.org/10.1080/01402390.2022.2164570. On other contextual factors that shape the risks of accidents, see Ingvild Bode, "Practice-Based and Public-Deliberative Normativity: Retaining Human Control over the Use of Force," *European Journal of International Relations,* April 10, 2023, https://doi.org/10.1177/13540661231163392; Patricia Owens, "Accidents Don't Just Happen: The Liberal Politics of High-Technology 'Humanitarian' War," *Millennium* 32, no. 3 (December 1, 2003): 595–616, https://doi.org/10.1177/03058298030320031101; Rebecca Slayton, "The Fallacy of Proven and Adaptable Defenses," Public Interest Report, Federation of American Scientists, 2014.

6 Charles Perrow, *Normal Accidents: Living with High Risk Technologies* (New York: Basic Books, 1984); Sagan, *The Limits of Safety.*

7 Matthijs M. Maas, "Regulating for 'Normal AI Accidents': Operational Lessons for the Responsible Governance of Artificial Intelligence Deployment," in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society,* 2018, 223–28; John Borrie, "Safety, Unintentional Risk and Accidents in the Weaponization of Increasingly Autonomous Technologies," *UNIDIR Resources* 5 (2016); Scharre, "Autonomous Weapons and Operational Risk"; Horowitz, "When Speed Kills"; Horowitz, Scharre, and Velez-Green, "A Stable Nuclear Future?"

8 Todd R. LaPorte and Paula M. Consolini, "Working in Practice But Not in Theory: Theoretical Challenges of 'High-Reliability Organizations'," *Journal of Public Administration Research and Theory* 1, no. 1 (January 1, 1991): 19–48, https://doi.org/10.1093/oxfordjournals.jpart.a037070; Andrew Hopkins, "The Limits of Normal Accident Theory," *Safety Science* 32, no. 2 (1999): 93–102.

9 Karlene H. Roberts, Denise M. Rousseau, and Todd R. La Porte, "The Culture of High Reliability: Quantitative and Qualitative Assessment Aboard Nuclear-Powered Aircraft Carriers," *The Journal of High Technology Management Research* 5, no. 1 (1994): 141–61; Scharre, "Autonomous Weapons and Operational Risk."

10 Thomas G. Dietterich, "Robust Artificial Intelligence and Robust Human Organizations," *Frontiers of Computer Science: Selected Publications from Chinese Universities* 13, no. 1 (February 1, 2019): 1–3, https://doi.org/10.1007/s11704-018-8900-4; Scharre, "Autonomous Weapons and Operational Risk," 51.

as an illustration, the failure of a Patriot missile defense system to intercept a Scud missile that struck a U.S. barracks at Dhahran, Saudi Arabia, in the last days of the Gulf War. The missile's impact caused the deaths of 28 soldiers, more than one-third of all U.S. servicemembers killed in the war.[11] Army investigators attributed the breakdown to a timing error in the computer software designed by Raytheon. Long, continuous operation led to loss of precision in tracking incoming missiles. Crucially, technical specialists were aware of the issue and had even developed a software patch, but the upgrade was not prioritized because they discounted the possibility that operators would keep the computer running for long periods without a reboot.[12] The accident was not "normal" — despite tight coupling and complexity, the cause was well understood, and a fix was available. Nor could it have been prevented by the Army improving its organizational culture and structure. The computer malfunction was a product of a considerable disconnect between Army users and software contractors in the Patriot's development process.

This theory is supported with four case studies: the 1988 Vincennes incident, in which a U.S. naval ship accidentally shot down an Iran Air civilian airliner; the 2003 Patriot fratricides at the beginning of the U.S. war in Iraq; the 2017 USS McCain collision; and software performance in the 2021 Kabul evacuation. In each case, the choices about software development by senior procurement officials and defense contractors were central to how software contributed to accident risks. This is not to say that human error played no part; rather, the way in which these military systems were developed set end-users up to fail.

This article makes two main contributions. First, scholarship on military accidents has been preoccupied with the clash between the normal accidents and high reliability organization frameworks. Since Scott Sagan's formative application of these frameworks to nuclear command and control accidents, this literature has seen very few theoretical innovations.[13] Departing from the normal accidents/high reliability organization dichotomy, this article presents a novel approach to exploring the sources and limits of safety in military technology systems. In doing so, it connects political science scholarship to a shift in the systems engineering, human-computer interaction, and risk management fields, all of which increasingly emphasize the need for "moving beyond normal accidents and high reliability organizations."[14] By focusing on participation by end-users in software development, software development lifecycle theory builds on recent scholarship on the military adoption of automated technologies, which underlines the significance of tactical-level operators' trust in new innovations.[15]

**While AI will bring novel risks, learning from past software-intensive military systems should serve as a foundation for comprehending the risks of military AI applications.**

Second, this article has direct implications for the risks of emerging technologies such as AI. In recent years, leading scholars and policymakers have likened the safety hazards of autonomous military systems to the Cold War's nuclear close calls, many of which were linked to false alarms produced by technological systems.[16] For example, Michael C. Horowitz, a University of Pennsylvania professor who serves as the director of the Department of Defense's Office of Emerging Capabilities Policy, reexamined the Cuban Missile Crisis with autonomous naval ships in the mix, giving significant attention to the accident risks of uncontrollable systems.[17] These analyses appropriately highlight specific features of AI-ena-

11 J.C. Humphrey, "Casualty Management: Scud Missile Attack, Dhahran, Saudi Arabia," *Military Medicine* 154, no. 5 (May 1999): 322–26, https://pubmed.ncbi.nlm.nih.gov/10332169/.

12 U.S. General Accounting Office, "Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia," February 1992, https://apps.dtic.mil/sti/citations/ADA344865. For more on Patriot malfunctions in the Gulf War, see Theodore A. Postol, "Lessons of the Gulf War Experience with Patriot," *International Security* 16, no. 3 (1991): 119–71, https://doi.org/10.2307/2539090.

13 Sagan, *The Limits of Safety*; Eric Schlosser, *Command and Control: Nuclear Weapons, the Damascus Accident, and the Illusion of Safety* (New York: Penguin, 2009).

14 Nancy Leveson, Nicolas Dulac, Karen Marais, and John Carroll, "Moving Beyond Normal Accidents and High Reliability Organizations: A Systems Approach to Safety in Complex Systems," *Organization Studies* 30, no. 2–3 (February 1, 2009): 227–49, https://doi.org/10.1177/0170840608101478.

15 Nina A. Kollars, "War's Horizon: Soldier-Led Adaptation in Iraq and Vietnam," *Journal of Strategic Studies* 38, no. 4 (June 7, 2015): 529–53, https://doi.org/10.1080/01402390.2014.971947; Julia Macdonald and Jacquelyn Schneider, "Battlefield Responses to New Technologies: Views from the Ground on Unmanned Aircraft," *Security Studies* 28, no. 2 (March 15, 2019): 216–49, https://doi.org/10.1080/09636412.2019.1551565.

16 Vincent Boulanin et al., "Artificial Intelligence, Strategic Stability and Nuclear Risk," SIPRI, June 2020, https://www.sipri.org/publications/2020/policy-reports/artificial-intelligence-strategic-stability-and-nuclear-risk; Horowitz, Scharre, and Velez-Green, "A Stable Nuclear Future?"

17 Horowitz, "When Speed Kills."

bled military applications, such as enhanced levels of autonomy, but they gloss over the simple fact that these applications will be implemented as software programs.[18] While AI will bring novel risks, learning from past software-intensive military systems should serve as a foundation for comprehending the risks of military AI applications.

The article proceeds as follows. It first outlines my argument about how software development practices influence the risk of military accidents. Next, the article explains the empirical method by distilling each of the three theories' expectations about the impact of software technology on military safety. Evaluations of the Vincennes, Patriot, and McCain cases trace the sources of these accidents back to the initial software requirements phase and the weak ties between software developers and military operators. In addition, evidence from the Afghanistan evacuation's use of Kessel Run[19] software further supports my theory by showing that an iterative approach to software development can reduce safety risks. The article concludes by discussing broader implications and policy recommendations, including defense software acquisition reforms that shift away from waterfall models and toward more agile approaches.

## Software Development Lifecycle

How does software affect military accidents? Typically, a software failure is defined as the inability of code to meet performance requirements. When post-accident investigations assign blame to the crew by noting that software systems performed flawlessly — as was the case with the Navy's report on the McCain crash — they rely on this narrow definition of software failure. In contrast, the normal accidents and high reliability theories emphasize that software can contribute to accidents, even when it performs how it is supposed to, by influencing the connected structures and organizations tasked with managing hazardous technologies.[20] These two prevailing organizational theories of safety in technological systems, therefore, offer a broader conception of software failure in military accidents.

The first approach contends that normal accidents occur in software-intensive military systems due to tightly coupled and highly complex structural elements. When problems — even seemingly trivial ones — arise in systems in which many events happen simultaneously and interact with each other, it is difficult for managers and operators to identify fixes.[21] Even if software works as coded, novel and unexpected interactions between battlefield conditions and such systems can snowball into unavoidable crises. For instance, in a 1979 false alert involving missile warning computers, the United States initiated retaliation measures based on mistaken reports of a major Soviet nuclear attack. This "near-miss" was produced by coincidences that would have been difficult to reasonably predict, including the insertion of training tape data at the same time as a momentary circuit failure in a ground station.[22]

The second approach, rooted in high reliability organizations, also posits that accidents in software-intensive military systems are rooted in the organizational structures that manage these systems. High reliability organization scholars claim that certain organizations can reliably prevent system accidents if they maintain certain qualities, such as deference to experienced operators, devotion to learning from failures, and commitment to safety.[23] Studies of the U.S. nuclear Navy, for instance, have highlighted the experience level of operators and cultural commitments to safety.[24] Notably, the autonomy of experienced front-line operators to "circumvent" certain bureaucratic procedures is one way high reliability organizations maintain system safety.[25]

While these two schools of thought are helpful for understanding how software affects military accidents, their focus is on unpredictable interactions or operator inadequacies that trip up software systems after they are in operation. Less attention is paid to the system acquisition and development phases before militaries even field these technologies. Yet, software safety specialists have identified that the bulk of safety-critical decisions are made during the initial phase of software design and requirements specification. Based on one study of military aviation mishaps, the concept development step accounts for 70 to 90 percent of safety-relevant decisions.[26]

Without accounting for military software acquisition and development, any explanation of accidents in software-intensive military systems is incomplete. In these early stages of system design, heavy reliance on contractors entails additional communication steps between software programmers and operators of deployed systems. Drawing on Nancy Leveson's seminal work on software-linked accidents, one review of decades of research on this subject concludes, "The source of most serious problems with software relates to outsourcing software development."[27] Regarding military accidents, the strength of these feedback channels between operators and software developers is crucial because they link military and civilian organizations that adhere to very different standards on system performance and reliability.

**Even if operators are sufficiently involved in the software development process, they still might not be able to anticipate all the scenarios under which the system will be used.**

This third approach, which I call software development lifecycle theory, highlights the impact of software acquisition patterns on the development of accident-prone systems. Military software development typically follows a linear "waterfall model," which begins with system requirements specification and then progresses sequentially through system design, development, testing, and deployment. Within this process, evaluation and feedback from operators are limited to the late stages of development, by which time it is difficult to rework system concepts.[28] As a 2010 National Research Council report notes, the Defense Department's acquisition practices for information technology are hampered by a "serial approach to development and testing (the waterfall model)," in which "end-user participation often is too little and too late."[29] Like water and waterfalls, operator input does not flow back up the chain of software development.

These limited feedback channels between end-use operators and software developers produce more accident-prone military systems through three interrelated pathways (see figure 1). First, the waterfall method tends to produce human-machine interaction issues such as difficulty accessing critical data and unwieldy interface designs. Consider, as an example, one of the early interfaces for the Patriot system, which Dr. John K. Hawley, an engineering psychologist with the U.S. Army Research Laboratory, recalls was an error-prone interface based on his user tests: "An insect could land on it even and change the settings." Yet, the interface was not fixed, in part due to the rigid sequential software development process established by Army senior leaders and the prime contractor Raytheon. As Hawley recounts, "But they had already spent all this money on it, and so they fielded it. The soldiers are put into the position where they have to use it whether they like it or not."[30]

Therefore, software can "fail" in the sense that it does not meet the needs of users, even if it passes all the technical requirements established in the system acquisition phase. Studies by human-computer interaction specialists have identified confusing interfaces as contributing to inadvertent military launches, fratricide, and other safety hazards. For instance, the

18 One exception is Schneider and Macdonald, "Looking Back to Look Forward," which discusses the impact of different acquisition strategies on managing the risks of autonomous systems, including operational trade-offs between control/safety and cost.

19 Kessel Run is a U.S. Air Force organization that develops command and control and targeting software capabilities. See https://kesselrun.af.mil.

20 Veronica L. Foreman, Francesca M. Favaró, Joseph H. Saleh, and Christopher W. Johnson, "Software in Military Aviation and Drone Mishaps: Analysis and Recommendations for the Investigation Process," *Reliability Engineering & System Safety* 137 (May 1, 2015): 102, https://doi.org/10.1016/j.ress.2015.01.006; Scott Shappell and Douglas Wiegmann, "The Human Factors Analysis and Classification System—HFACS," Embry-Riddle Aeronautics University Publications, February 1, 2000, https://commons.erau.edu/publication/737. The human factors analysis and classification system, used by the U.S. military for aviation accident investigations, also considers human errors as a product of broader organizational influences. I thank an anonymous reviewer for raising this point.

21 Perrow, *Normal Accidents*; Sagan, *The Limits of Safety.*

22 Another coincidence that no one could have reasonably predicted was that the last block of sequential numbers on messages coming into the multiplexor of the system computer (the 427M) had been "001" and the first block of numbers in the mistakenly inserted training data was "002." Aerospace Defense Command, "History of ADCOM/ADC, 1 January–31 December 1979," n.d., Secret, excerpts, excised copy January 1, 1980, newly declassified.

23 LaPorte and Consolini, "Working in Practice but Not in Theory."

24 Roberts, Rousseau, and La Porte, "The Culture of High Reliability"; Scharre, "Autonomous Weapons and Operational Risk."

25 Leveson et al., "Moving Beyond Normal Accidents and High Reliability Organizations," 228. For a similar argument in a very different policy area, see Susanna P. Campbell, *Global Governance and Local Peace: Accountability and Performance in International Peacebuilding* (Cambridge: Cambridge University Press, 2018), https://doi.org/10.1017/9781108290630.

26 Nancy G. Leveson, *Engineering a Safer World* (Cambridge, MA: The MIT Press, 2012), 51; F.R. Frola and C.O. Miller, "System Safety in Aircraft Acquisition" (Washington, DC: Logistics Management Institute, January 1984).

27 Roel I.J. Dobbe, "System Safety and Artificial Intelligence," in *The Oxford Handbook of AI Governance*, ed. Justin B. Bullock et al. (Oxford: Oxford University Press, 2022), https://doi.org/10.1093/oxfordhb/9780197579329.013.67. Spanning engineering, risk assessment, and science and technology studies, scholars from a wide range of disciplines have studied how to safeguard software-based systems over the past few decades. See also Madeleine Clare Elish, "Moral Crumple Zones: Cautionary Tales in Human-Robot Interaction," *Engaging Science, Technology, and Society* 5 (March 23, 2019): 40–60, https://doi.org/10.17351/ests2019.260.

28 John K. Hawley, "Patriot Wars: Automation and the Patriot Air and Missile Defense System," Center for a New American Security, January 25, 2017, https://www.cnas.org/publications/reports/patriot-wars. See also Scharre, "Debunking the AI Arms Race Theory."

29 National Research Council, *Achieving Effective Acquisition of Information Technology in the Department of Defense* (Washington, DC: National Academies Press, 2010), https://doi.org/10.17226/12823. Related to the waterfall model, the DOD has historically relied on block development, in which each development phase is completed once according to unchanging software requirements established at the outset (cf. literature on evolutionary acquisition). David N. Ford and John Dillard, "Modeling the Performance and Risks of Evolutionary Acquisition," *Defense AR Journal* 16, no. 2 (2009): 143. I thank Jackie Schneider for her insights on this topic.

30 Interview with John K. Hawley, May 9, 2022.

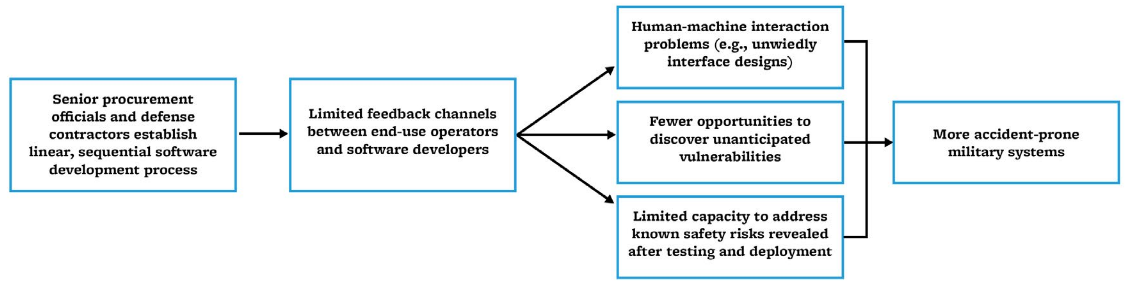

Senior procurement officials and defense contractors establish linear, sequential software development process
Limited feedback channels between end-use operators and software developers
Human-machine interaction problems (e.g., unwiedly interface designs)
Fewer opportunities to discover unanticipated vulnerabilities
Limited capacity to address known safety risks revealed after testing and deployment
More accident-prone military systems

*Figure 1. Causal Graph for Software Development Lifecycle Theory*

lack of visual contrasts between button interfaces for different settings has caused drone crashes.[31] These human-machine interaction problems are intensified in high-stress scenarios when military operators must rapidly interpret data and make decisions.

Second, limited operator involvement in the software design process leads to fewer opportunities to discover unanticipated vulnerabilities. As the Scud missile attack discussed above demonstrates, there are some contingencies that could be identified in the development process if software developers better understood how operators would likely employ a system.[32] To address this issue, the Department of Defense has pushed for agile software engineering practices that involve rapid prototyping and early engagement with operators.[33]

Even if operators are sufficiently involved in the software development process, they still might not be able to anticipate all the scenarios under which the system will be used.[34] Rebecca Slayton has shown that missile defense systems require vastly different software settings for field tests versus real-world use.[35] Regardless, iterative software development practices can uncover more potential safety hazards than rigid, sequential ones.

Third, waterfall models of software development limit the capacity of military organizations and defense contractors to ameliorate issues revealed by the testing and deployment process. Unlike the hidden vulnerabilities issue discussed above, these safety risks are *known*. However, they remain unresolved because, at this phase of the software development lifecycle when end-use operators point out hazards, it is too late, costly, and difficult to rework system designs. As one expert on human-machine integration in military systems notes, “By that time, most degrees of freedom for concept reevaluation or design changes have been lost.”[36]

Due to the inherent complexity and adaptability of software, the effect of system procurement practices on safety is especially salient for software-intensive military systems. As the three pathways illustrate, military software development presents unique challenges because it necessitates continuous adaptation to unanticipated vulnerabilities as well as end-use operator understanding of the system. For hardware development, sequential procurement approaches may be more suitable for managing safety issues since potential vulnerabilities and human-machine interaction effects are more predictable.[37]

To reinforce this point, consider the differences between “smart” and normal (or “dumb”) refrigerators. When accounting for hardware failures in a normal fridge, manufacturers tend to focus on a limited set of physical or chemical mechanisms (e.g., a puncture that results in gas leakage). Software-based smart fridges, on the other hand, present a wider range of failure paths connected to unpredictable human-machine interactions (e.g., user forgets to connect fridge to new wi-fi system, which causes temperature control software to malfunction).

Software development lifecycle theory builds on the normal accident and high reliability organization approaches, which have broadened our perspective on software’s role in military accidents, but also unearths overlooked causal factors. Debates between the normal accidents and high reliability organizations camps tend to concentrate on whether accidents are inevitable in complex technological systems.[38] Likewise, this clash has been adjudicated in military technologies by focusing on how military organizations operated and managed such systems after they had been fielded. Software development lifecycle theory brings the causal timeline of military accidents back to the initial phases of software design and requirements specifications. In doing so, it also expands the range of actors responsible for military accidents to include the defense contractors that build the software.

The ramifications of software development lifecycle theory are present in other contexts where national governments rely on contractors to develop technological systems.[39] On the one hand, the Defense Department’s efforts to work with non-traditional contractors (commercial firms with limited defense sales) could mitigate software failure scenarios, as these entities have developed significant experience with agile methods in large-scale software development projects such as a traffic management system.[40] On the other hand, the costs of incorporating operational feedback on integrating commercial-off-the-shelf software products in military systems may be prohibitive.[41] Like the Defense Department’s acquisition process, the National Aeronautics and Space Administration’s contracting process has sometimes limited the ability of astronauts to provide input on safety issues.[42] As a result, contractors aggressively pursued short-term fixes to avoid significant delays in delivery, which created unaddressed safety hazards.

As for the root causes leading militaries to adopt certain patterns of software acquisition, political and bureaucratic barriers often hamper transitions to iterative software development approaches. To maintain key programs and lucrative contracts based on waterfall models, the prime defense contractors exploit their close ties with politicians and defense officials as well as their deep understanding of military requirements built up over many years of doing business with the government.[43] Since more agile development might entail losing out to more open-minded competitors, there is little incentive for long-time defense contractors to abandon the waterfall model, as shown by resistance to past acquisition reforms.[44] Senior procurement officials, who “grew up with” the waterfall approach, may not invest in building an acquisition culture that supports agile methods.[45]

## Empirics

Illuminating dynamics often overlooked by the two dominant approaches of normal accidents and high reliability organizations theories, I assess software development lifecycle theory across four historical case studies: the Vincennes accident, the Patriot fratricides, the USS McCain collision, and Kessel Run software’s performance in the 2021 Afghanistan evacuation. Adopting a process-tracing approach, in each of the cases, I evaluate the observable implications of software development lifecycle theory, and then compare them to those derived from the two more established theories (see table 1).[46] While the three theoretical approaches agree that there is a causal relationship between the introduction of software and an accident, they offer different interpretations of *how* this process occurs, in particular as it relates to principal actors, timing of most relevant decisions, precipitating events, and conceptions of software failure. The last case, in which the U.S. Air Force

31 Mary L. Cummings, "Automation and Accountability in Decision Support System Interface Design," *Journal of Technology Studies* 32, no. 1 (2006); Thomas B. Sheridan, *Humans and Automation: System Design and Research Issues* (New York: Wiley, 2002).

32 Eric Schmitt, "AFTER THE WAR; Army Is Blaming Patriot's Computer for Failure to Stop the Dhahran Scud," *The New York Times*, May 20, 1991, https://www.nytimes.com/1991/05/20/world/after-war-army-blaming-patriot-s-computer-for-failure-stop-dhahran-scud.html; Eliot Marshall, "Fatal Error: How Patriot Overlooked a Scud," *Science* 255, no. 5050 (March 13, 1992): 1347, https://doi.org/10.1126/science.255.5050.1347.

33 Defense Innovation Board, "Software Acquisition and Practices (SWAP) Study," May 3, 2019, https://innovation.defense.gov/software/.

34 Alan Borning, "Computer System Reliability and Nuclear War," *Communications of the ACM* 30, no. 2 (1987): 112–31.

35 Slayton, "The Fallacy of Proven and Adaptable Defense." These unanticipated contingencies correspond to the novel and unexpected interactions under normal accidents theory.

36 Hawley, "Patriot Wars," 13.

37 Defense Innovation Board, "Software Acquisition and Practices (SWAP) Study," viii–ix. I am grateful to an anonymous reviewer for guidance on this passage.

38 In Sagan's words, these discussions center around "conflicting visions about what could be called the degree of perfectibility that is possible in complex organizations." Sagan, *The Limits of Safety*, 14.

39 These theoretical claims also apply to other militaries. On different models of defense software development in France, the United Kingdom, Germany, and China, see Simona R. Soare, Pavneet Singh, and Meia Nouwens, "Software-Defined Defence: Algorithms at War," The International Institute for Strategic Studies, February 2023, https://www.iiss.org/research-paper//2023/02/software-defined-defence.

40 Mary Ann Lapham et al., "Considerations for Using Agile in DoD Acquisition," Software Engineering Institute, 2010, https://insights.sei.cmu.edu/documents/2180/2010_004_001_15155.pdf; Michael P. Fischetti, "The Challenges Facing 'Non-Traditional' Contractors," *Georgia Tech Contracting Education Academy* (blog), April 10, 2020, https://contractingacademy.gatech.edu/2020/04/10/15587/.

41 Nancy G. Leveson, "Using Cots Components in Safety-Critical Systems," in *RTO Meeting on COTS in Defense Applications*, 2000, http://sunnyday.mit.edu/papers/cots.pdf.

42 Diane Vaughan, "Autonomy, Interdependence, and Social Control: NASA and the Space Shuttle Challenger," *Administrative Science Quarterly* 35, no. 2 (1990): 241, https://doi.org/10.2307/2393390.

43 Peter Dombrowski and Eugene Gholz, *Buying Military Transformation: Technological Innovation and the Defense Industry*, 1st ed. (New York: Columbia University Press, 2006); John A. Alic, "The Origin and Nature of the U.S. 'Military-Industrial Complex'," *Vulcan* 2, no. 1 (2014): 63–97.

44 Eugene Gholz and Harvey M. Sapolsky, "Restructuring the U.S. Defense Industry," *International Security* 24, no. 3 (1999): 34–35.

45 Lapham et al., "Considerations for Using Agile in DoD Acquisition"; Daniel E. Schoeni, "Long on Rhetoric, Short on Results: Agile Methods and Cyber Acquisitions in the Department of Defense," *Santa Clara High Technology Law Journal* 31 (2015): 385.

46 Joachim Blatter and Till Blume, "In Search of Co-Variance, Causal Mechanisms or Congruence? Towards a Plural Understanding of Case Studies," *Swiss Political Science Review* 14, no. 2 (2008): 315–56, https://doi.org/10.1002/j.1662-6370.2008.tb00105.x.

| | Software development lifecycle | High reliability organizations | Normal accidents |
|---|---|---|---|
| *Principal actors* | Software developers (contractors) and senior procurement officials | Military organizations responsible for operating software systems | Military organizations responsible for operating software systems |
| *Timing of most relevant decisions* | Initial contracting phase, software requirements specification, preceding decade(s) | After the system has been fielded (emphasis on continuous operations and training) | Combination of system design and organizational operations after system has been fielded |
| *Conception of software failure* | Software development process does not incorporate feedback from end-users | Operators not adequately trained to manage software | Software contributes to tight coupling and high complexity |
| *Precipitating events* | Predictable issues that remain unaddressed due to software development lifecycle | Mistakes that escalate because operators fail to maintain system safety | Novel and unexpected interactions between system components |

*Table 1. Three Perspectives on Software's Contribution to Military Accidents*

experimented with a different approach to software acquisition and development, further probes software development lifecycle theory by incorporating variation on the type of software acquisition pathway.

All these accidents are the product of multiple, overlapping factors. In many cases, precipitating conditions based on all three perspectives are present, making it difficult to evaluate their relative explanatory power. In addressing this issue, I first analyze whether the evidence matches up with predictions unique to software development lifecycle theory, which do not intersect with the other mechanisms.[47] For instance, if an accident can be traced to known vulnerabilities rooted in software development patterns, which neither made the system more complex nor could have been addressed by more training, then this would constitute strong evidence for my argument. Second, I examine the historical record for evidence that components of the software development lifecycle mechanism served as antecedent conditions for the high reliability organizations and normal accident mechanisms. Indications that limited feedback between operators and software developers produced systems that either were highly complex and tightly coupled or that constrained the ability of operators to manage safety risks would illustrate the value of software development lifecycle theory.

To ensure a rigorous test of software development lifecycle theory, I selected cases that are representative illustrations of the normal accidents and high reliability organizations models. Analyses of the McCain, Vincennes, and Patriot accidents often pinpoint either complex software or inadequate operator training as key causal factors.[48] Paul Scharre, for example, writes, "The causes behind the Patriot fratricides illustrate how normal accidents also can occur in military systems."[49] This test's generalizability is enhanced because these cases also differ in many ways, including their operational context, level of complexity, the defense contractor involved, type of safety hazard, as well as the relevant military organizations.[50]

Insights from this article's cases, therefore, enable us to better understand the risks of human-software interaction in conflict settings where reducing accident risk is especially challenging.

The focus on software applications in navigation, operational planning, and weapons control systems — which aid platforms in tracking, targeting, and shooting their targets — provides two further advantages. First, these are reference classes that are similar in many relevant aspects to how AI could be incorporated into weapons platforms.[51] Second, compared to those linked to nuclear weapons systems, command and control issues in these military software applications are relatively understudied, and they provide a new universe of cases to explore the effects of technology on military accidents.[52] Thus, they provide fertile ground for evaluating whether software development lifecycle theory can offer insights not fully captured by the normal accidents or high reliability organizations perspectives.

Lastly, these cases all feature systems in military settings where safety must be balanced against operational effectiveness. Lessons learned from software-related accidents in the civilian sector where safety considerations are paramount, such as airline transport or air traffic control, may not translate. Similarly, in many of the military operations studied by high reliability organization researchers, such as non-combat aircraft operations, safety goals were protected from other competing priorities.[53] Insights from this article's cases, therefore, enable us to better understand the risks of human-software interaction in conflict settings where reducing accident risk is especially challenging.

I drew from three additional sources to enrich the case study analysis: recently declassified U.S. government documents, archived discussions in the Forum on Risks to the Public in Computers and Related Systems,[54] and interviews with contractors and military officials who developed and tested these systems.[55] Taken together, these sources fill important gaps in the official post-accident investigations. Typically staffed by military operators who want to avoid implicating senior procurement decisionmakers, boards of inquiry for military accidents tend to avoid investigating the system design process.[56] Ideally, the scope of the analysis would include non-U.S. cases. However, this decision was shaped by practical considerations, including access to interviewees and archival records.

## USS Vincennes Shootdown of Iran Air Flight 655

On July 3, 1988, the USS Vincennes was passing through the Strait of Hormuz, within Iranian territorial waters, to investigate reports of Islamic Revolutionary Guard Corps speedboats attacking neutral merchant ships. The Vincennes boasted an Aegis system, a highly sophisticated combat information center that automated functions such as target classification and target-weapon pairing.[57] That morning, radar operators on the Vincennes misidentified an Iranian passenger airliner as an Iranian F-14 Tomcat, and the U.S. warship fired two surface-to-air missiles at Iran Air flight 655, killing all 290 civilians on board. Before the July 2014 MH17 shootdown over Ukraine, it stood as the deadliest civilian airliner shootdown in history.[58]

How does a billion-dollar warship, equipped with state-of-the-art software for tracking and classifying aircraft, end up shooting down a civilian passenger plane? The official investigation, known as the Fogarty report, revealed that the ship's Aegis system supplied accurate data to the Vincennes crew.[59] The Aegis provided altitude information that the plane was ascending (like a commercial plane), not descending (like a hostile military plane). The crew, however, reported that the aircraft was descending as it approached the ship.[60] There is much to be mined from the high-stakes calls made in these critical minutes, and the ensuing investigations and Congressional hearings identified human error (including stress and psychological issues) as the primary cause.[61] Yet, it is equally, if not more, valuable to trace problems with the Aegis further back to the decisions made about software design in the initial procurement phase. This would also cast attention on the companies that built the system, namely the RCA Corporation, which was

47 Derek Beach and Rasmus Brun Pedersen, *Process-Tracing Methods: Foundations and Guidelines* (Ann Arbor: University of Michigan Press, 2013), 100–5.

48 Bode, "Practice-Based and Public-Deliberative Normativity"; Scharre, "Autonomous Weapons and Operational Risk"; Miller et al., "The Navy Installed Touch-screen Steering Systems to Save Money."

49 Scharre, "Autonomous Weapons and Operational Risk."

50 The three main branches of the U.S. military are represented: the Air Force (Afghanistan evacuation), the Navy (the Vincennes and the McCain), and the Army (Patriot fratricides). While the Patriot fratricides involve three friendly fire incidents, if my argument holds, these three episodes should trace back to software development lifecycle issues that are common to all Patriot systems. Thus, I treat these incidents as one case.

51 Perrow, the pioneer of normal accidents theory, also cites software as a neglected area. Perrow, *Normal Accidents*, 354.

52 Gene I. Rochlin, "Iran Air Flight 655: Complex, Large-Scale Military Systems and the Failure of Control," in *Responding to Large Technical Systems: Control or Anticipation*, ed. Todd R. LaPorte (Berkeley: University of California Press, 1991), 95–121.

53 Leveson et al., "Moving Beyond Normal Accidents and High Reliability Organizations," 239.

54 Sponsored by the Association for Computing Machinery, this forum brings together experts to discuss computer-related mishaps.

55 This study was declared exempt by the George Washington University Institutional Review Board under Department of Health and Human Services regulatory category 2 (IRB# NCR245704).

56 Interview with C.W. Johnson, author of handbook on military accidents, March 13, 2023.

57 Samuel Cox, "H-020-1: USS Vincennes Tragedy," July 2018, http://public1.nhhcaws.local/content/history/nhhc/about-us/leadership/director/directors-corner/h-grams/h-gram-020/h-020-1-uss-vincennes-tragedy--.html.

58 Nick Danby, "How the Downing of Iran Air Flight 655 Still Sparks U.S.-Iran Enmity," *Responsible Statecraft* (blog), July 2, 2021, https://responsiblestatecraft.org/2021/07/02/how-the-downing-of-iran-air-flight-655-still-influences-us-iran-enmity/.

59 William M. Fogarty, *Investigation Report: Formal Investigation into the Circumstances Surrounding the Downing of Iran Air Flight 655 on 3 July 1988* (Washington, DC: Department of Defense, 1988).

60 R. Jeffrey Smith, "Decision on Vincennes Echoes Precedent," *Washington Post*, August 20, 1988, https://www.washingtonpost.com/archive/politics/1988/08/20/decision-on-vincennes-echoes-precedent/069c5094-670b-42ed-b2ae-d10b904354cb/.

61 Kristen Ann Dotterway, "Systematic Analysis of Complex Dynamic Systems: The Case of the USS Vincennes," Master's thesis, Naval Postgraduate School, 1992.

awarded the prime contract for the Aegis in 1969.[62]

If software development lifecycle theory holds, the case evidence should show that the system development process factored into the combat information center operators' struggles with managing Aegis-related risks. One of the key issues is whether the Aegis' program managers and RCA gave sufficient attention to user interface considerations in tense, combat settings. Matt Jaffe, a systems engineer at RCA in the mid-1970s, pushed for the Aegis display to include a rate-of-change indicator for altitude. On debates over this issue, Jaffe recalls, "I wound up literally screaming at my boss, 'You hired me for my technical experience and combat experience. If you're not going to listen to me, why don't you fire me?!'"[63] Having previously served in the Vietnam War on ships equipped with forerunners to the Aegis system, Jaffe was one of the few people involved in Aegis software development with experience operating similar systems in high-stress environments. Without this rate-of-change indicator, controllers had to "compare data taken at different times and make the calculation in their heads, on scratch pads, or on a calculator — and all this during combat."[64] This increased the likelihood of misreading whether a ship was descending or ascending.

These issues could be traced back to the extent, timing, and influence of feedback from military operators to software contractors in the system development process. The Aegis system was developed based on the Department of Defense's "1679A" software standard, which heavily relied on the waterfall model and even restricted the ability of designers and users to collaborate and revise initial specifications.[65] As Jaffe states, "When Aegis was being developed, it was being developed with a waterfall (model). ... we drew up a software requirements specification document, sent it to the Navy for review, and then they would come back for one meeting of a few hours. It's hard to do human-machine interface that way."[66]

Furthermore, many actors had raised concerns that Aegis systems proceeded into production without adequate attention to testing and operator feedback. A General Accounting Office investigation reported a "long list of testing limitations" to the Aegis system at RCA's facility in Moorestown, New Jersey, such as the on-land location, lack of actual missile firing capability, and differences between the tested and fielded versions of the system.[67] The realism of these tests has also been questioned. The test ranges were set up such that threats would only come from a predictable area. Moreover, certain events, such as aircraft leaving the immediate test area, tipped off crews that a test event would soon occur. Together, these conditions allowed "crews to deduce the general direction, timing, and type of the test threats."[68] Sources familiar with a classified version of the General Accounting Office report confirmed that these so-called sea tests did not approximate a realistic, challenging combat environment.[69]

Later examinations of the Vincennes case have undermined this account, arguing that the crew's mistakes were likely due to a combination of information overload and unwieldy user displays, rather than collective bias.

62 Philip J. Hilts, "Aegis System Has Been Controversial from the Start," *Washington Post*, July 7, 1988, https://www.washingtonpost.com/archive/politics/1988/07/07/aegis-system-has-been-controversial-from-the-start/eb4d2ab1-b3a4-40d0-8496-82b1541d924f/. Through a series of acquisitions and sales, beginning with General Electric's purchase of RCA in 1986, Lockheed Martin has taken over this business line.

63 Interview with Matt Jaffe, March 8, 2023. Jaffe also emphasized that RCA managers provided some valid pushback to this indicator, including: the limited display space, risks of information overload, and issues related to the vertical beam width of the radar (which could muddle rate-of-change calculations). Ultimately, Jaffe's supervisor claimed that the Navy never requested such an indicator. For Jaffe, the root issue remained that the input RCA received from the Navy was from senior officers on the project management team who did not have operational experience with these systems.

64 Eric J. Lerner, "Lessons of Flight 655," *Aerospace America* 27, no. 4 (1989): 18.

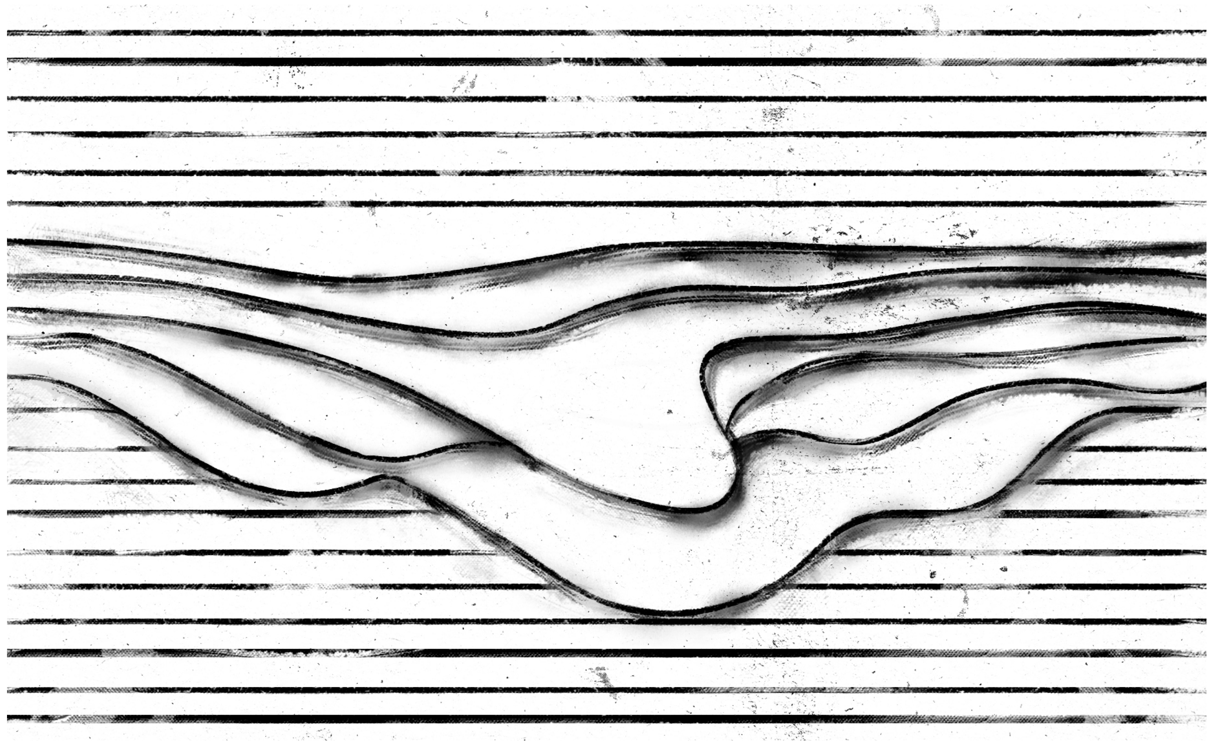
65 Harvey Lyon, "Navy Military Standards for Technical Software Documentation of Embedded Tactical Systems; a Critical Review," Naval Postgraduate School, 1985, https://apps.dtic.mil/sti/citations/ADA161238, 13–14, 26–27; National Research Council, "Achieving Effective Acquisition of Information Technology in the Department of Defense," 48.

66 Interview with Matt Jaffe, March 8, 2023.

67 I am very grateful to Shelby Oakley, at the Government Accountability Office, for helping me locate this report. U.S. General Accounting Office, "Weapons Testing: Quality of DOD Operational Testing and Reporting," July 26, 1988, https://www.gao.gov/products/pemd-88-32br.

68 U.S. General Accounting Office, "Weapons Testing."

69 Ahern, Tim. "Aegis System Got Poor Marks in GAO Report." Associated Press, July 9, 1988. Regarding these sea trials, a detailed *Newsweek* investigation stated that "the navy could not afford to risk failure in the trials for fear that Congress would stop funding the Aegis program." John Barry and Roger Charles, "Sea of Lies," *Newsweek*, July 12, 1992, https://www.newsweek.com/sea-lies-200118.



Unlike many post-incident investigations, this General Accounting Office report — released just three weeks after the Vincennes accident — identified problems with the Aegis originating from *before* the accident, thus limiting the risk of hindsight bias in shaping its conclusions. The report drew on interviews done between September 1987 and March 1988. It also scrutinized the testing and evaluation processes for not just the Aegis but also five other systems.[70]

Finally, even accounts that focused on the tactical judgements in the final minutes before the accident eventually landed on problems stemming from the early stages of system development. As noted above, Congress and the Navy's investigations of the Vincennes accident pointed to the problem of "scenario fulfillment," which leads highly trained organizations to unconsciously ignore and misinterpret evidence that does not conform to a preconceived scenario, such as an Iranian air attack.[71] Later examinations of the Vincennes case have undermined this account, arguing that the crew's mistakes were likely due to a combination of information overload and unwieldy user displays, rather than collective bias.[72] Moreover, some psychologists stressed that operator errors were entrenched in the Aegis system design and development process. Richard W. Pew, in testimony to a Congressional hearing on the Vincennes accident on behalf of the American Psychological Association, stated, "Part of the problem is that automation decisions are made at the time the fundamental architecture of a system is being defined. We need more extensive methods of analysis to understand how to integrate human operator performance with system performance *during the conceptual design state of new weapon systems*."[73]

What can normal accidents and high reliability organizations theories reveal about the Vincennes case? At the time, the Aegis was one of the most complicated weapons systems in action.[74] It could also operate at high levels of automation from target tracking to missile firing sequences. This type of tight coupling was needed to respond to the types of threats that Aegis would likely face — the time between the appearance of the Iranian aircraft on the radar screen and the decision to fire spanned

70 U.S. General Accounting Office, "Weapons Testing"; Eleanor Chelimsky, "Review of the Office of Operational Test and Evaluation: Hearing before the Acquisition Policy Panel Committee on Armed Services," September 14, 1988.

71 Changes in the rules of engagement are another contributing factor. See Sagan, "Rules of Engagement." Sagan identifies "hair trigger" rules of engagement as a "permissive cause" in the Vincennes tragedy.

72 For one of the strongest arguments against the "scenario fulfillment" explanation, see Dotterway, "Systematic Analysis of Complex Dynamic Systems." Nancy C. Roberts and Kristen Ann Dotterway, "The Vincennes Incident: Another Player on the Stage?" *Defense Analysis* 11, no. 1 (1995): 31–45.

73 Richard Pew, "On the Subject of the USS Vincennes Downing of Iran Air Flight 655," September 14, 1988. Emphasis mine.

74 Hilts, "Aegis System Has Been Controversial from the Start."

four minutes.[75] With this type of system, normal accidents theory tells us that a Vincennes-like disaster was inevitable. If the Vincennes accident was caused by novel and unexpected interactions related to the Aegis system, that would further support the normal accident theory explanation. Based on evidence gathered by Captain Kristen Ann Dotterway, it is possible that the Aegis automatically re-assigned the Iranian plane's track number (TN 4474) to a different track number (TN 4131) entered by the USS Sides, which was operating with the Vincennes at the time.[76] Moments later, when the Vincennes crew asked for an update on TN 4474, thinking it was still attached to the Iranian plane, the Aegis computer had already matched that number to a U.S. Navy aircraft that was landing on a U.S. aircraft carrier in the Gulf of Oman. By some accounts, this was a "freak occurrence" that explains why multiple crew members reported that the Iranian aircraft was descending.[77]

On the other hand, aspects of this case suggest that the breakdowns in information-gathering aboard the Vincennes were predictable. Assuming the TN 4474 re-assignment account holds up, the software development process should have addressed the risks associated with automatically changing track numbers.[78] At the very least, the Aegis should have had an alert mechanism that notified the operator when track numbers had been automatically re-assigned.[79] Dotterway, who originally reconstructed the sequence of events in this account, also blames the "poor interface between the Aegis weapon system and the operator, especially the procedural complexity and a problematic presentation of information illustrated in the auto-correlation and subsequent confusion of track numbers."[80] Furthermore, another explanation for the "descending" call was that the crew had mistaken decreasing range values as altitude values, which would connect back to the system development issues related to rate-of-change indicator for altitude.[81]

By highlighting the operators of the Vincennes, the high reliability organizations approach also offers some insights into this case. According to the theory, preventing Aegis-linked accidents comes down to whether organizations can cultivate environments in which groups can reliably perform in high-pressure situations, such as deference to experienced frontline operators who can independently circumvent rules to prevent accidents when issues arise.[82] In the Vincennes case, the combat information center operators had limited experience managing Aegis systems in high-stress environments. According to a *Newsweek* investigation, the tactical officer tasked with directing the ship's navigation and weapons systems based on data from the combat information center "was uncomfortable with computers" and, according to one fellow officer, "used his screen as a surface for 'self-stick' notes" instead of displaying potential incoming threats.[83]

Yet, it is important to not overstate the implications of high reliability organizations theory in this case. Even the most well-trained and experienced crew could have struggled to process Aegis data in combat conditions.[84] As the evidence above demonstrates, the Vincennes' problems were less about the level of training or the ability of operators to learn from mistakes and more entrenched in the flaws of the software development model. By the time operators could train with and test the system, any issues they identified could not have been fixed. Thus, the high reliability organizations approach disregards the role of the contractors who develop Aegis software and the robustness of feedback loops between military operators and system designers.

The Department of Defense's official investigation report on the Vincennes incident numbers 153 pages.[85] There are over 70 references to Captain Will Rogers III, who commanded the Vincennes; nearly 20 mentions of the experience level of operators; and nine references to the compression of time or complexity of the technological system.[86] RCA, the system developer, is not mentioned even once. Fully understanding the causes of the Vincennes tragedy requires looking beyond just the organization using the Aegis system to Moorestown, New Jersey, where the Missile and Surface Radar Division of RCA was located.

## Patriot "Friendly Fire" Incidents

In 2003, during the Iraq War, the U.S. Army's Patriot air defense system committed three "friendly fire" incidents. In the first episode, which occurred on March 23, a Patriot missile shot down a Royal Air Force Tornado fighter-bomber, killing both crewmembers. Two days later, a Patriot radar locked on to a U.S. Air Force F-16; in response, the pilot destroyed the Patriot battery with a missile. Fortunately, this confrontation resulted in no casualties. About a week later, a Patriot battery shot down a U.S. Navy F/A-18C, killing the pilot. These three friendly fire incidents comprised 25 percent of the Patriot's 12 engagements in the conflict.[87]

As they made upgrades to the Patriot's software, contractors and concept designers built technical components to meet certain efficiency and performance requirements, leaving operators to deal with the residual impacts (e.g., the Patriot's difficulties with identifying friends or foes).

Problems with how the Patriot system interpreted identification friend or foe signals were central to these friendly fire incidents. In 2005, a Defense Science Board task force reviewed the Patriot's performance in the Iraq War and concluded that the identification friend or foe technology developed by Raytheon, the system's prime contractor, performed poorly — a problem that had surfaced during training exercises.[88] This begs the question of why this issue was not addressed. The task force stated that it "remains puzzled as to why this deficiency never garner[ed] enough resolve and support to result in a robust fix."[89]

Detailed investigations established that the answer to this puzzle lies in the system development process, which extends back to at least 1985, the year that Raytheon was awarded a Patriot software modernization contract.[90] According to John Hawley, who has over 35 years of expertise on human-machine interactions in Patriot units, the Department of Defense's systems acquisition process was the biggest obstacle to safer Patriot performance in the Iraq war. Raytheon followed the waterfall model, in which feedback and evaluation were left until system development was nearly complete.[91] Hawley comments, "[The system developers] still have this idea that they have this acquisition process, and when it's finished, it's finished. Oftentimes, the user is not equipped to use it that way."[92] The General Accounting Office diagnosed similar issues in earlier cycles of Patriot development, in which "the Army believed it necessary to proceed (with production) even though test results identified major problems."[93]

This waterfall approach was especially ill-suited to Patriot upgrades in identification algorithms and automatic operating modes, which were brittle and demanded military users to intervene in extreme situations.[94] As they made upgrades to the Patriot's software, contractors and concept designers built technical components to meet certain efficiency and performance requirements, leaving operators to deal with the residual impacts (e.g., the Patriot's difficulties with identifying friends or foes). Whether operators could meet the associated

75 Ingvild Bode and Thomas Watts, "Meaning-Less Human Control: Lessons from Air Defence Systems on Meaningful Human Control for the Debate on AWS," Centre for War Studies (University of South Denmark) and Drone Wars UK, 2021, https://dronewars.net/wp-content/uploads/2021/02/DW-Control-WEB.pdf, 44.

76 A track number is the unique label that a radar system assigns to each new possible target.

77 Dirk Maclean, *Shoot, Don't Shoot: Minimizing Risk of Catastrophic Error Through High Consequence Decision-Making,* Air Power Development Centre, 2017, 29.

78 In fact, these types of risks were well-established, not freak occurrences. Interview with Dr. Nancy Roberts, June 12, 2023.

79 Dotterway, "Systematic Analysis of Complex Dynamic Systems," 59.

80 Dotterway, 173.

81 Dotterway, 54.

82 Scharre, *Army of None.*

83 Barry and Charles, "Sea of Lies."

84 Barry and Charles.

85 Fogarty, *Investigation Report.*

86 Though the formal investigation of the shootdown exonerated Captain Rogers from blame, other accounts highlight that the captain's aggressive pursuit of Iranian gunboats contributed to the time pressures faced by the Aegis system. David Evans, "Vincennes: A Case Study," U.S. Naval Institute *Proceedings* 119, no. 8 (August 1993), https://www.usni.org/magazines/proceedings/1993/august/vincennes-case-study.

87 Scharre, *Army of None.*

88 Defense Science Board Task Force. "Patriot System Performance" (Washington, DC: Department of Defense, 2005).

89 Defense Science Board Task Force.

90 This contract initiated the first phase of software modifications (PAC-1). The PAC-2 software changes, which were the most relevant to Operation Iraqi Freedom, began in 1986.

91 Hawley, "Patriot Wars."

92 Interview with John K. Hawley, May 9, 2022.

93 Frank Conahan, "DOD's Management of Government Property Furnished to Defense Contractors," June 23, 1983, 9, https://www.gao.gov/products/t-nsiad-88-19.

94 Hawley, "Patriot Wars."

demands was not tested until the end of the waterfall software development process, when it was too late to change the system's fundamental design. Hawley and Anna L. Mares, another researcher at the U.S. Army Research Laboratory, conclude, "The roots of [the Patriot's] apparent human performance shortcomings can be traced back to systemic problems resulting from decisions made years earlier by concept developers, software engineers, procedures developers, testers, trainers, and unit commanders."[95]

Indeed, even efforts oriented toward identifying deficiencies among U.S. Army Patriot operators eventually turned to the system acquisition process. Following the 2003 fratricides, the Army Research Laboratory initiated "the Patriot Vigilance Project," which, as its name suggests, initially aimed to investigate the discipline and alertness of Patriot operators. Ultimately, after expanding its scope to cover 20 years of the Patriot's evolution, the review warned against laying too much blame on operators, instead emphasizing deeper system development problems like "faulty going-in concepts" that proved "difficult and expensive to correct" later in the process.[96]

How well do the normal accidents and high reliability organizations approaches account for the Patriot friendly fire accidents? Certainly, the two system features of normal accidents were present in the Patriot case. To begin, it was difficult for operators to grasp the intricate connections between the Patriot system's moving parts. In the aftermath of the incidents, experts scrutinized the Patriot's "enormous complexity," which had been enhanced by software upgrades to enable automated engagement of a target.[97] Second, the Patriot system was also tightly coupled. There was very little reaction time between the initial detection of an incoming missile and the decision to respond.[98] As the United Kingdom Ministry of Defence's inquiry into the Patriot-Tornado fratricide put it, "The crew had about one minute to decide whether to engage."[99]

While some aspects of this case bear out the expectations of normal accidents theory, it misses other key contributing factors to the Patriot accidents. The observable implications of normal accidents are most compatible with the second fratricide involving the F/A-18C, in which the Patriot system failed to adapt to novel and unexpected interactions.[100] In this instance, when Patriot radars operated in close proximity and followed the same aircraft, their pulses would produce false ballistic missile trajectories, or "ghost tracks."[101] From a normal accidents viewpoint, this situation was relatively unpredictable, and it would have been difficult to uncover in the development process.

However, other elements of the Patriot accidents were more preventable. The "ghost tracks" issue was not relevant to the Tornado incident or the U.S. F-16 engagement on the Patriot. Most of the Patriot's identification of friend or foe shortcomings were well-known to both system operators and pilots, who feared flying in airspace tracked by Patriots due to the frequency with which the radar systems would lock on to their aircraft.[102] For example, as Paul Scharre points out, the risk that Patriots would mistake aircraft for anti-radiation missiles "had been identified during operational testing but had not been corrected and were not included in operator training."[103]

Other factors in this case also illustrate the high reliability organizations approach's utility and drawbacks. The Patriot crews were relatively inexperienced and overly reliant on automated outputs.[104] In its investigation of the Patriot fratricides, the Defense Science Board task force recommended that operators gain more autonomy over firing decisions. "The solution here will be more operator involvement and control in the functioning of a Patriot battery," the task force report states.[105] Failing to satisfy critical features of a high reliability organization, Patriot unit operators tended to lack the know-how needed to bypass established practices in order to avoid disaster in high-pressure situations.[106]

Still, regarding the Patriot accidents, high reliability organizations theory's explanatory power is limited in two ways. First, it is unlikely that even more experienced operators would have been able to substantially reduce accident risks. The Patriot's identification of friend or foe problems meant that operators would have very little time and information to override the system's classifications.[107]

Second, the high reliability organizations approach neglects the need for operator feedback earlier in the system development process. A more iterative development process could have alerted designers and contractors to the need for targeting algorithms that could be adapted to the prevailing missile threats that operators would encounter in a given operating environment.[108] Instead, the U.S. Army and Raytheon "committed to a system concept that demonstrate[d] patterns of performance unreliability," leaving it to the operators to deal with the additional risks.[109] In this sense, the focus on operator adaptability points toward an end-of-pipe solution, whereas the software development lifecycle approach controls risk from the source.

The above evidence suggests that the Patriot's safety issues are deeply embedded in the military's acquisition approach for software-intensive systems. In response to the Patriot fratricides, the military implemented software upgrades and adjusted human-machine interfaces to help operators adapt to new systems. Yet, these reforms leave unresolved the underlying problems with how the military, in partnership with Raytheon and other contractors, acquires and develops systems like the Patriot.

## USS McCain and Alnic MC Collision

On Aug. 21, 2017, the U.S. Navy destroyer John S. McCain made a sudden turn to port and unintentionally struck the Alnic MC, a Liberian-registered oil tanker, off the coast of Singapore and Malaysia, east of the Strait of Malacca. About an hour earlier, the destroyer's commanding officer, Alfredo J. Sanchez, switched the "Integrated Bridge and Navigation System" to backup mode, setting off a series of mistakes that caused the hard turn. Ten U.S. Navy sailors died because of the collision, making it the Navy's deadliest accident in four decades.[110]

A number of investigations into the collision pointed to design flaws in the McCain's navigation system as playing a major role in the accident. First, in its backup mode, the system allowed crew from different parts of the ship to take charge of steering, which led to confusion over which station had thrust control for different propellers. Second, for steering commands, the system only provided touch-screen controls, as opposed to mechanical throttles that give more tactile feedback to operators.[111] As the National Transportation Safety Board's investigation concludes, "The design of the John S. McCain's touch-screen steering and thrust control system increased the likelihood of the operator errors that led to the collision."[112]

**Ultimately, however, these design flaws were rooted in the proclivity of system designers to automate and digitalize navigation functions without sufficient attention to the needs of operators.**

To be sure, issues with the integrated bridge and navigation system development process could have been partially mitigated by improved decision-making on the day of the collision or better training in the months before. Ultimately, however, these design flaws were rooted in the proclivity of system designers to automate and digitalize navigation functions without sufficient attention to the needs of operators. According to operators interviewed in the Navy's investigation, they regularly disabled the system's touch screen to avoid accidental rudder changes and

95 John K. Hawley and Anna L. Mares, "Human Performance Challenges for the Future Force: Lessons from Patriot after the Second Gulf War," in *Designing Soldier Systems: Current Issues in Human Factors*, ed. John Martin et al. (London: CRC Press, 2018), 3–34.

96 Hawley, "Patriot Wars."

97 Charles Piller, "Vaunted Patriot Missile Has a 'Friendly Fire' Failing," *Los Angeles Times*, April 21, 2003, https://www.latimes.com/archives/la-xpm-2003-apr-21-war-patriot21-story.html.

98 If the Patriot was operating in auto-fire mode, even less slack was present.

99 Ministry of Defence, "Aircraft Accident to Royal Air Force Tornado GR MK4A ZG710," March 23, 2003, 3, https://www.gov.uk/government/publications/military-aircraft-accident-summary-aircraft-accident-to-raf-tornado-gr-mk4a-zg710; cited in Bode and Watts, "Meaning-Less Human Control."

100 Scharre, *Army of None*, 144.

101 Theodore A. Postol, "An Informed Guess about Why Patriot Fired upon Friendly Aircraft and Saw Numerous False Missile Targets during Operation Iraqi Freedom," Security Studies Program, Massachusetts Institute of Technology, April 20, 2004; Benjamin S. Lambeth, *The Unseen War: Allied Air Power and the Takedown of Saddam Hussein* (Annapolis, MD: Naval Institute Press, 2013), 245.

102 After the F-16 aircraft shot down the Patriot unit, one pilot remarked, "We had no idea where the Patriots were, and those guys were locking us up on a regular basis. No one was hurt when the Patriot was hit, thank God, but from our perspective they're now down one radar. That's one radar they can't target us with any more." Lambeth, *The Unseen War*, 115.

103 Scharre, *Army of None*, 144.

104 Automation bias refers to the phenomenon when operators have "too much" trust in autonomous systems. Bode and Watts, "Meaning-Less Human Control"; Horowitz, "When Speed Kills."

105 Defense Science Board Task Force, "Patriot System Performance."

106 An organization's learning orientation is another key characteristic of HROs. In this case, there is evidence that the Army did not learn from Patriot deficiencies in the first Gulf War. Postol, "Lessons of the Gulf War Experience with Patriot."

107 Ministry of Defence, "Aircraft Accident to Royal Air Force Tornado GR MK4A ZG710."

108 Bode and Watts, "Meaning-Less Human Control," 55.

109 Hawley, "Patriot Wars."

110 Miller et al., "The Navy Installed Touch-screen Steering Systems to Save Money."

111 S.C. Mallam, K. Nordby, S.O. Johnsen, and F.B. Bjørneseth, "The Digitalization of Navigation: Examining the Accident and Aftermath of U.S. Navy Destroyer John S. McCain," *Proceedings of the Royal Institution of Naval Architects Damaged Ship V*, 2020, 57.

112 National Transportation Safety Board, "Collision between U.S. Navy Destroyer John S. McCain and Tanker Alnic MC Singapore Strait, 5 Miles Northeast of Horsburgh Lighthouse August 21, 2017," 2019, https://www.ntsb.gov/investigations/AccidentReports/Reports/MAR1901.pdf, 33.

ignored fault notifications due to displays that were difficult to interpret.[113] Notably, efforts to modernize the system did not involve consultation with the Navy's own experts on human factors engineering.[114]

It was only after the McCain collision that the Navy recognized the need for stronger feedback loops between operators and software developers. Prompted by an internal review, the Navy surveyed surface ship crews about the navigation system. One striking finding — "the number-one feedback from the fleet" according to the program executive officer for ships, Rear Adm. Bill Galinis — was that operators "overwhelmingly" preferred mechanical controls over touch-screen systems.[115] The initial development of the navigation system lacked this type of participatory input, in which end-user perspectives are incorporated into the design of new technology.[116] As Eric Lofgren, an expert on defense acquisition, states, "Such an advanced bridge/navigation system should probably first been tried on smaller ships with continuous user feedback, tested extensively, iterated, then progressively scaled up to larger and more complex ships."[117] In line with software development lifecycle theory, these problems extended back almost a decade of software acquisition and development to 2008, when the Navy first announced a contract with Northrop Grumman to build the integrated bridge and navigation system.[118]

Insights from normal accidents and high reliability organizations theories also bear on this case. Regarding the former, some of the navigation system's issues stemmed from adding unnecessary complexity, including the ability to transfer thrust control for each of the ship's two propellers.[119] Moreover, the accident happened in one of the world's most congested waterways, an environment ripe for unexpected interactions in which one misstep could easily lead to a catastrophe.[120] Applied to this case, normal accidents theory holds that incidents like the McCain collision are inevitable as long as the Navy relies on navigation systems that are highly complex and tightly coupled.

However, normal accidents theory does not fully capture some of the key aspects of the McCain accident. For starters, not all the design issues with the navigation system can be reduced to unexpected interactions and insufficient slack between various components. For instance, automated bridge systems equipped with improved indicators and alarms should produce looser coupling by giving the crew more time to correct issues that arise. Yet, the McCain crew ignored these alert systems because the display area was densely packed and difficult to interpret — the product of not incorporating operator needs into the design process.[121] Furthermore, many of the navigation system's complications were neither novel nor unexpected, as presumed by normal accidents theory, but rather predictable. The replacement of mechanical throttle with touch-screen controls, for example, conflicted with user-centered design principles held by the Navy's own human factors engineering team and Department of Defense standards.[122]

Some of the lessons from high reliability organizations also pertain to the McCain case. Forward-deployed naval forces in the Western Pacific faced high operational tempo and staffing shortages, which resulted in long shifts and inadequate rest.[123] This affected the crew's attention to detail and reaction time, undermining the ability to take safety measures. Indeed, the National Transportation Safety Board's review found that the bridge watchstanders were "acutely fatigued at the time of the accident."[124] Another critical aspect of high reliability organizations is the high experience level of operators. In this case, post-accident investigations highlighted that the crew

113 U.S. Fleet Forces Command, "Comprehensive Review of Recent Surface Forces Incidents."

114 U.S. Fleet Forces Command.

115 Megan Eckstein, "Navy Reverting DDGs Back to Physical Throttles, After Fleet Rejects Touchscreen Controls," *USNI News* (blog), August 9, 2019, https://news.usni.org/2019/08/09/navy-reverting-ddgs-back-to-physical-throttles-after-fleet-rejects-touchscreen-controls.

116 Mallam et al., "The Digitalization of Navigation."

117 Eric Lofgren, "Is Poor Software Design/Testing the REAL Cause of the USS McCain Crash?" *Acquisition Talk* (blog), December 25, 2019, https://acquisitiontalk.com/2019/12/is-poor-software-design-testing-the-real-cause-of-the-uss-mccain-crash/.

118 Miller et al., "The Navy Installed Touch-screen Steering Systems to Save Money."

119 Mallam et al., "The Digitalization of Navigation."

120 Miller et al., "The Navy Installed Touch-screen Steering Systems to Save Money"; Mallam et al., "The Digitalization of Navigation."

121 U.S. Fleet Forces Command, "Comprehensive Review of Recent Surface Forces Incidents."

122 Mallam et al., "The Digitalization of Navigation," 57; U.S. Fleet Forces Command, "Comprehensive Review of Recent Surface Forces Incidents."

123 Joseph Aucoin, "It's Not Just the Forward Deployed," U.S. Naval Institute *Proceedings*, April 2018, https://www.usni.org/magazines/proceedings/2018/april/its-not-just-forward-deployed; Aaron Rowen, Martha Grabowski, and Dale Russell, "The Impact of Work Demands and Operational Tempo on Safety Culture, Motivation and Perceived Performance in Safety Critical Systems," *Safety Science* 155 (November 1, 2022): 105861, https://doi.org/10.1016/j.ssci.2022.105861.

124 National Transportation Safety Board, "Collision between U.S. Navy Destroyer John S. McCain and Tanker Alnic MC Singapore Strait."

lacked sufficient training on the navigation system.[125]

While high reliability organization theory spotlights how the McCain crew could have better managed deficiencies in the navigation system, this approach does not account for why the system was developed to be accident-prone in the first place. In-depth investigations of the McCain collision called attention to poor technical documentation for the system, which would have even hampered the ability of a highly reliable organization to make safety adaptations.[126] In part because he could not understand some of the system's automated functions in sea trials, the McCain's commander grew accustomed to operating the navigation system in backup mode, which removed built-in safeguards.[127] Contrary to the expectations of high reliability organizations theory, which regard the ability of operators to circumvent certain procedures as a benefit to system safety, in this case, such a move enhanced the risk of accidents.

Following the accident, the Navy dismissed the ship's top officers for failing to properly manage the navigation system in the critical minutes before the collision, while the organizations and officials responsible for software development were not held accountable. In August 2019, the Navy announced that it would continue to work with Northrop Grumman to develop more basic touch-screen controls and add physical throttles to the system. Yet, there has been no evidence that this redesign process will incorporate input from end-users.[128] As the software development lifecycle approach suggests, without this fundamental change in the connections between the organizations that procure and develop the software, accidents like the McCain collision will continue to occur.

## Kessel Run Software and the Afghanistan Withdrawal

Amid the Taliban's swift takeover of Afghanistan in August 2021, the United States and its allies and partners scrambled to evacuate foreign citizens and some Afghan citizens from the country before the Biden administration's planned withdrawal deadline of Aug. 31.[129] To manage the complicated evacuation process from the Hamid Karzai International Airport in Kabul — in all, more than 120,000 people were flown out — the U.S. Air Force relied on a software tool to plan out the exact time slots for arrivals and departures.[130] Developed by Kessel Run, the Air Force's own software factory, the planning software played a major role in the largest noncombatant evacuation in U.S. military history.[131] Lt. Gen. Greg Guillot, who led the mission as the Air Forces Central commander, stated that Kessel Run's software "served as a reliable, adaptable tool as we planned and executed this complex, historic operation."[132]

What accounts for the safe and reliable performance of Kessel Run's software in the Afghanistan evacuation? This outcome cannot be fully anticipated by the high reliability organizations and normal accidents frameworks. The operation, a chaotic 17-day sprint that required an immense surge of Air Force aircraft and crew, could not benefit from the continuous learning existent in high reliability organizations.[133] In addition, the evacuation's complexity had increased the number of missions beyond the point that the software was designed to accommodate. The compressed timeline meant that on-the-fly patches were required to deal with software outages.[134]

In line with the expectations of software development lifecycle theory, Kessel Run's software development model was critical to the airlift planning sys-

125 U.S. Fleet Forces Command, "Comprehensive Review of Recent Surface Forces Incidents"; National Transportation Safety Board, "Collision between U.S. Navy Destroyer John S. McCain and Tanker Alnic MC Singapore Strait."

126 This section benefited from Ralph Soule's discussion of HRO theory's implications for the McCain case in his blog, available here: https://www.ralphsoule.com/blog/tag/HRO_JSM.

127 Advice from two other captains helped him make this decision. Miller et al., "The Navy Installed Touch-screen Steering Systems to Save Money."

128 James A. Malachowski, "Robots and Rogue Thinkers: Leveraging Organizational Learning Theory to Prevent Catastrophic Failure After Rapid Fielding of Disruptive Technology," Naval War College, 2020; Director, Operational Test & Evaluation, "FY 2023 Annual Report," Department of Defense, January 2024, https://www.dote.osd.mil/Portals/97/pub/reports/FY2023/other/2023annual-report.pdf, 148–51.

129 Michael D. Shear, Annie Karni, and Eric Schmitt, "Afghanistan, Biden and the Taliban: Biden Says U.S. Is on Track to Finish Evacuation by Deadline," *The New York Times*, August 24, 2021, https://www.nytimes.com/live/2021/08/24/world/afghanistan-taliban-kabul-news.

130 Richard Blumenstein, "Kessel Run's C2IMERA Used during Afghan Evacuation," Air Combat Command, September 23, 2021, https://www.af.mil/News/Article-Display/Article/2787545/#:~:text=%E2%80%9CKessel%20Run's%20C2IMERA%20application%20served,logistics%20and%20NEO%20support%20information.

131 Damany Coleman, "Kessel Run's SlapShot Saves Lives," *Air Force Life Cycle Management Center* (blog), September 28, 2021, https://www.aflcmc.af.mil/NEWS/Article-Display/Article/2791602/.

132 Blumenstein, "Kessel Run's C2IMERA Used during Afghan Evacuation." This matches evaluations of Kessel Run by the U.S. Air Force leaders to the Senate Armed Service Committee, in which they cited the lab's performance in discovering design flaws earlier in the development process; https://www.armed-services.senate.gov/imo/media/doc/Wilson_04-04-19.pdf.

133 Alex Horton and Dan Lamothe. "Inside the Afghanistan Airlift: Split-Second Decisions, Relentless Chaos Drove Historic Military Mission," *Washington Post*, September 28, 2021, https://www.washingtonpost.com/national-security/2021/09/27/afghanistan-airlift-inside-military-mission/.

134 Brian Beachkofski, "Making the Kessel Run," *Air & Space Forces Magazine* (blog), March 23, 2022, https://www.airandspaceforces.com/article/making-the-kessel-run/.

**While the first three case studies serve as the main test ground for software development lifecycle theory's expectations of how organizational structures for software development contribute to military accidents, evidence from Kessel Run demonstrates how iterative software development practices can reduce the risk of mishaps.**

tem's ability to limit the risk of accidents. Accounting for this outcome necessitates an understanding of Kessel Run's iterative software engineering process, under which planning tools like the one used in the evacuation were developed around cycles of experimentation with significant user feedback and testing.[135] This approach starkly differs from the software modernization effort that Kessel Run replaced, the "AOC 10.2" program initially awarded to Lockheed Martin in 2006, which had limited engagement with end-users even a decade after software requirements were established.[136]

The importance of this adaptable approach was validated by Kessel Run's response to a software outage, prompted by the planning tool's intermittent loading issues as it struggled to accommodate the exponential growth in the number of flights demanded by the evacuation. During the mission, the U.S. military planned and directed 2,627 flights. At one point, half of the Air Force's C-17 transport planes were dedicated to the effort.[137] In the span of 12 hours, the Kessel Run team implemented a number of technical fixes that incorporated feedback from end-users in the air operations center.[138] According to one Kessel Run engineer, this adaptation was successful because the team could "dry-run the changes they were making in a sort of digital staging area, while liaison officers — with literally the boots-on-the-ground, in some cases — could communicate with end users to fully realize their immediate needs."[139] As Brian Beachkofski, who led Kessel Run during the evacuation, notes, if this system was developed with a "big bang" release, in which end-users do not engage with the software until after the big product delivery event, this on-the-fly technical adaptation would have been impossible.[140]

While the first three case studies serve as the main test ground for software development lifecycle theory's expectations of how organizational structures for software development contribute to military accidents, evidence from Kessel Run demonstrates how iterative software development practices can reduce the risk of mishaps. Some qualifying factors should be considered. Kessel Run's safety benefits were limited to one specific mission, not proven across years of operations. Fortunately, this type of long-term analysis may be more feasible in the future, since the U.S. military is transitioning major combat systems like the Aegis to more continuous software development practices.[141] In this operation, the consequences of a software mishap were different than the three other cases examined here because a mishap could have resulted in a plane taking off without having aerial refueling support, which could have resulted in the plane being forced to land in a place that would not accept the passengers. For systems where the stakes are high, such as nuclear command and control, a continuous delivery approach that eschews software requirements specification may not be the most appropriate approach.[142]

135 Phil Budden et al., "Kessel Run: An Innovation Opportunity for the U.S. Air Force." MIT Mission Innovation Working Paper, May 2021.

136 Beachkofski, "Making the Kessel Run."

137 Horton and Lamothe, "Inside the Afghanistan Airlift."

138 Beachkofski, "Making the Kessel Run." The technical fixes involved adding more compute instances and compressing the number of missions into more compact visual displays.

139 Coleman, "Kessel Run's SlapShot Saves Lives."

140 Interview with Brian Beachkofski, April 10, 2023. This "big bang" release approach is a common byproduct of the waterfall model.

141 Paul DeLuca et al., "Assessing Aegis Program Transition to an Open-Architecture Model," RAND, 2013, https://apps.dtic.mil/sti/citations/ADA583571. Additionally, Kessel Run now assists the F-22 and F-35 program offices, which govern some of the military's largest procurement programs. Budden et al., "Kessel Run: An Innovation Opportunity for the U.S. Air Force"; Pomerleau, Mark. "Air Force Software Tool Helped Coordinate Afghanistan Evacuation of Civilians." C4ISRNet, September 23, 2021. https://www.c4isrnet.com/battlefield-tech/c2-comms/2021/09/23/air-force-software-tool-helped-coordinate-afghanistan-evacuation-of-civilians/.

142 Nancy Leveson, "An Engineering Perspective on Avoiding Inadvertent Nuclear War," NAPSNet Special Reports, July 26, 2019, https://nautilus.org/napsnet/napsnet-special-reports/an-engineering-perspective-on-avoiding-inadvertent-nuclear-war/.

## Conclusions and Implications for Policymakers

The international relations discipline has largely spurned the study of accidents, possibly because they tend to be seen as random events that cannot be explained by general causes.[143] In this line of thinking, civilian casualties are tragic but inevitable costs of warfare; friendly fire incidents are unfortunate, but it would be pointless to try to identify their common determinants. By contrast, this article maintains that a science of accidents is possible.

By centering the system acquisition and development process, I have explained and demonstrated how software technologies contribute to military accidents. When military software development follows a "waterfall" approach, in which input from operators is limited to the final stages of testing and deployment, many safety hazards will only be revealed after it is too late to rework system designs. Under this model of software development, the risk of accidents is elevated. Conversely, when the software development process allows for early feedback from military operators during system design and requirements specification, confusing interface designs and other human-machine interaction problems can be mitigated.

This article makes two main interventions in the literature on complex technological systems and military accidents. First, studies of military accidents have primarily tapped into two wells of organizational scholarship, the high reliability organization and normal accident approaches, on managing hazardous technologies. By focusing on breakdowns in complex technologies and military organizations in the critical seconds and minutes of a crisis, applications of these two theories neglect the processes that occur before militaries field technological systems: acquisition and development. In extending the causal timeline of military accidents beyond decisions made on the battlefield to those made decades earlier in software design and development, software development lifecycle theory presents an alternative way to explain how software contributes to military accidents.

My aim is not to devalue the contributions of the normal accident and high reliability organization theories to the study of safety in military systems. These two approaches have introduced important concepts like coupling and complexity, and they have underlined the impact of safety culture and organizational structure on accidents.[144] They have also broadened the notion of software failure, beyond the inability of code to meet performance specifications, to encompass problems linked to interactions between software systems and surrounding structures and organizations. In fact, this article demonstrates that the politics of software acquisition can both feed into the problem of normal accidents and inhibit organizations from achieving high reliability. As the case studies illustrated, the software development process can be the reason why militaries field highly complex, tightly coupled systems in the first place. Likewise, limited feedback channels between software designers and military operators can inhibit the risk mitigation practices characteristic of high-reliability organizations.[145]

Second, for policymakers seeking to reduce the risk of accidents involving military AI applications, this article points toward reforms to defense software acquisitions as one effective pathway to produce safer systems. This directly contradicts the notion that the waterfall model's top-down nature and strict requirements enhance safety, which is a viewpoint held by some Department of Defense project managers.[146] This article's historical lessons should also amplify calls of other researchers on the benefits of agile software development for making military systems more resilient to problems that arise from human-machine teaming. A 2021 *War on the Rocks* piece, for instance, partly attributed design flaws in the U.S. Navy's littoral combat ship to a waterfall project management approach.[147]

Given the bureaucratic and political forces that often resist acquisition reforms, the Defense Department's recent championing of agile software development — often framed in the context of improving efficiency, not safety — should be treated with skepticism. According to a Software Engineering Institute assessment, there is a significant mismatch between the department's rhetorical embrace of agile methods and its actual adoption of such practices.[148] Without interventions that account for bureaucratic inertia and prime contractors' vested interests in maintaining waterfall methods, military AI systems may replicate the safety risks of past software-intensive systems.

Indeed, this article puts forward that lessons from the development of automation software in older military systems can directly apply to managing emerging technology risks. Revisiting near–nuclear

143 Owens, "Accidents Don't Just Happen."

144 Leveson et al., "Moving Beyond Normal Accidents and High Reliability Organizations."

145 I thank Josh Rovner for his helpful advice on this section.

146 Schoeni, "Long on Rhetoric, Short on Results."

147 Jonathan Panter and Jonathan Falcone, "The Unplanned Costs of an Unmanned Fleet," *War on the Rocks* (blog), December 28, 2021, https://warontherocks.com/2021/12/the-unplanned-costs-of-an-unmanned-fleet/.

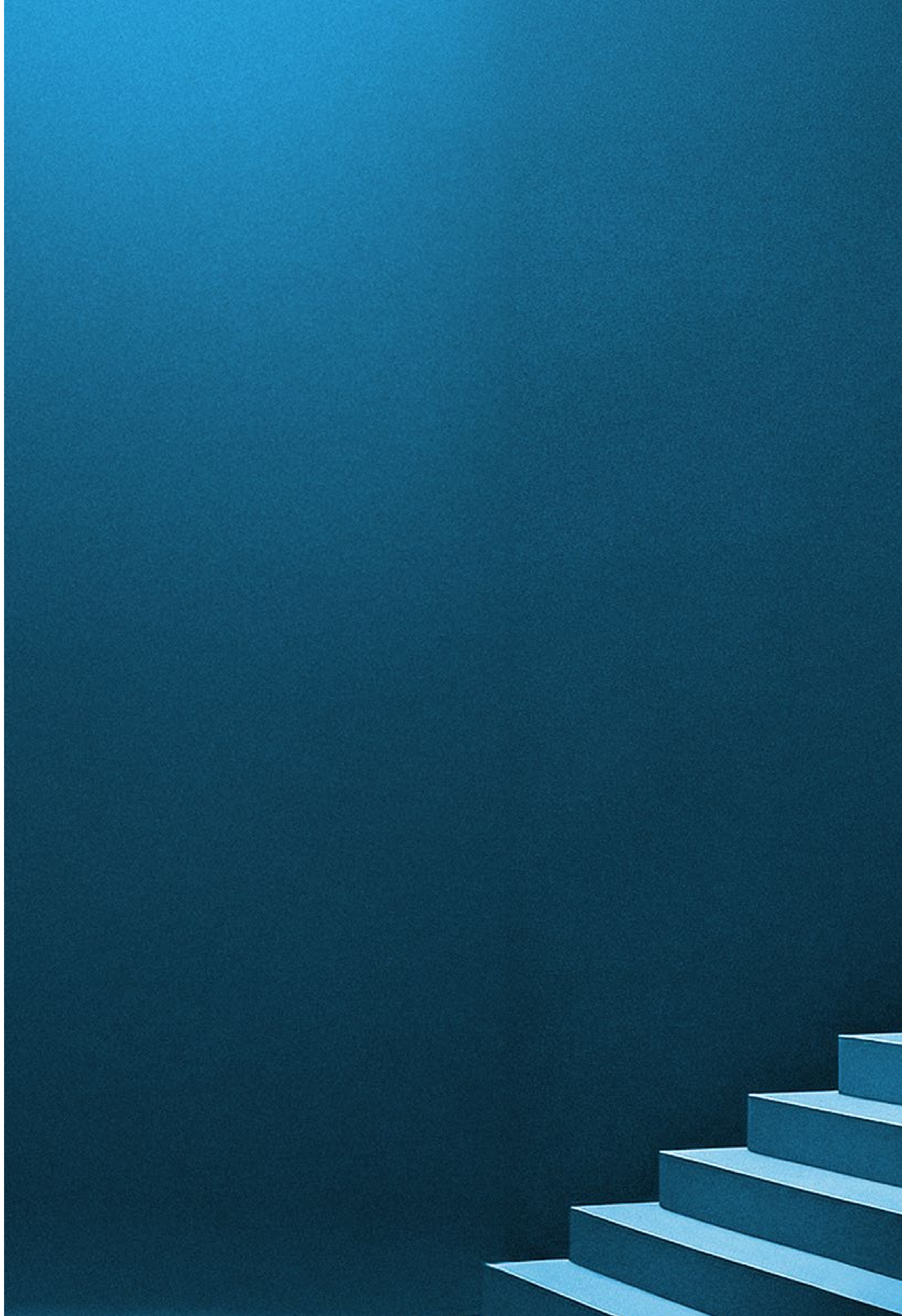
148 Lapham et al., "Considerations for Using Agile in DOD Acquisition."

confrontations in the Cold War, a 2022 *Bulletin of the Atomic Scientists* essay warned, "Today, artificial intelligence, and other new technologies, if thoughtlessly deployed could increase the risks of accidents and miscalculation even further."[149] In these discussions, both scholars and policymakers often gravitate toward "novel" risks, such as those linked to increased speed of decision-making. To be sure, there are many ways that AI systems today differ from the software of old.[150] And, certainly, investigating those unique features will uncover useful insights into understanding how AI will affect military accidents. At the same time, there is much to be learned from historical cases of software-intensive military systems. After all, new technologies cannot so easily escape deep-rooted problems.

***Jeffrey Ding*** *is an Assistant Professor of Political Science at George Washington University. He is the author of* Technology and the Rise of Great Powers *(Princeton University Press, 2024*).

***Acknowledgments:*** *The author wishes to thank Markus Anderljung, Noemi Dreksler, Ben Garfinkel, Julie George, Joshua Rovner, Scott Sagan, Toby Shevlane, Eoghan Stafford, Sanne Verschuren, Baobao Zhang, participants at an International Studies Association panel, the anonymous reviewers, and especially Allan Dafoe, Jacquelyn Schneider, and Rick Landgraf, for their helpful feedback and suggestions. This project also benefited from support of Stanford University's Center for International Security and Cooperation and Stanford's Institute for Human-Centered Artificial Intelligence.*

***Image:*** *U.S. Navy photo by Mass Communication Specialist 2nd Class Joshua Fulton*[151]



149 Christian Ruhl, "Sixty Years after the Cuban Missile Crisis, How to Face a New Era of Global Catastrophic Risks," *Bulletin of the Atomic Scientists* (blog), October 13, 2022, https://thebulletin.org/2022/10/sixty-years-after-the-cuban-missile-crisis-how-to-face-a-new-era-of-global-catastrophic-risks/.

150 For an overview of unique safety issues with AI systems built using machine learning and reinforcement learning techniques, see Dario Amodei et al., "Concrete Problems in AI Safety," *arXiv Preprint arXiv:1606.06565,* 2016.

151 For the image, see https://www.dvidshub.net/image/3692032/uss-john-s-mccain-arrives-changi-naval-base.