2019

Lessons from Literal Crashes for Code

Margot Kaminski
University of Colorado Law School

Lessons from Literal Crashes for Code

Author : Margot Kaminski

Date : October 3, 2019

Bryan H. Choi, *Crashworthy Code*, 94 **Wash. L. Rev.** 39 (2019).

Software crashes all the time, and the law does little about it. But as Bryan H. Choi notes in *Crashworthy Code*, "anticipation has been building that the rules for cyber-physical liability will be different." (P. 43.) It is one thing for your laptop to eat the latest version of your article, and another for your self-driving lawn mower to run over your foot. The former might not trigger losses of the kind tort law cares about, but the latter seems pretty indistinguishable from physical accidents of yore. Whatever one may think of CDA 230 now, the bargain struck in this country to protect innovation and expression on the internet is by no means the right one for addressing physical harms. Robots may be special, but so are people's limbs.

In this article, Choi joins the fray of scholars debating what comes next for tort law in the age of embodied software: robots, the internet of things, and self-driving cars. Meticulously researched, legally sharp, and truly interdisciplinary, *Crashworthy Code* offers a thoughtful way out of the impasse tort law currently faces. While arguing that software is exceptional not in the harms that it causes but in the way that it crashes, Choi refuses to revert to the tropes of libertarianism or protectionism. We can have risk mitigation without killing off innovation, he argues. Tort, it turns out, has done this sort of thing before.

Choi dedicates Part I of the article to the Goldilocksean voices in the current debate. One camp, which Choi labels consumer protectionism, argues that with human drivers out of the loop, companies should pay the cost of accidents caused by autonomous software. Companies are the "least cost avoiders" and the "best risk spreaders." This argument tends to result in calls for strict liability or no-fault insurance, neither of which Choi believes to be practicable.

Swinging from too hot to too cold, what Choi calls technology protectionism "starts from the opposite premise that it is cyber-physical manufacturers who need safeguarding." (P. 58.) This camp argues that burdensome liability will prevent valuable innovation. This article is worth reading for the literature review here alone. Choi briskly summarizes numerous calls for immunity from liability, often paired with some version of administrative oversight.

Where Goldilocks's bears found happiness in the third option, Choi's third path forward is found wanting. What he calls doctrinal conventionalism takes the view that tort law as-is can handle things. Between negligence and strict products liability, this group argues, tort will figure robots out.

This third way, too, initially seems unsatisfactory. The law may be able to handle technological developments, Choi acknowledges, but the interesting question isn't whether; it's how. And crashing code, he argues in Part II, is at least somewhat exceptional. Its uniqueness isn't in the usual lack of physical injuries, or the need for a safe harbor for innovation. It's the problem of software complexity that makes ordinary tort frameworks ill-suited for governing code. Choi explains that software complexity makes it impossible for programmers to guarantee a crash-free program. This "very basic property of software…[thus] defies conventional judicial methods of assessing reasonableness." (P. 79.) No matter how many resources one applied to quality assurance, one could "still emerge with errors so basic a jury would be appalled." (P. 80.) (Another bonus for the curious: Choi's discussion of top-down attempts to bypass these problems through mandating particular formal languages in high-stakes fields such as national security and aviation.)

The puzzle, then, isn't that software now produces physical injuries, thus threatening the existing policy balance between protecting innovation and remediating harm. It's that these newly physical injuries make visible a

characteristic of software that makes it particularly hard to regulate ex post, through lawsuits. In other words, "[s]oftware liability is stuck on crash prevention," when it should be focused instead on making programmers mitigate risk. (P. 87.)

In Part III, Choi turns to a line of cases in which courts found a way to get industry to increase its efforts at prevention and risk mitigation, without crushing innovation or otherwise shutting companies down. In a series of crashworthiness cases from the 1960s, courts found that car manufacturers were responsible for mitigating injuries in a car crash, even if (a) such crashes were statistically inevitable, and (b) the chain of causation was extremely hard to determine. While an automaker might not be responsible for the crash itself, it could be held liable for failing to make crashing safer.

Crashworthiness doctrine, Choi argues, should be extended from its limited use in the context of vehicular accidents to code. In the software context, he argues that "there are analogous opportunities for cyber-physical manufacturers to use safer designs that can mitigate the effects of a software error between the onset and the end of a code crash event." (P. 101.) Programmers should be required not to prevent crashes entirely, but to use known methods of fault tolerance, which Choi discusses in detail. Courts applying crashworthiness doctrine to failed software thus would inspect the code to determine whether it used reasonable fault tolerance techniques.

Crashworthy Code could be three articles: one categorizing policy tropes; one identifying what makes software legally exceptional or disruptive; and one discussing tort law's ability to handle risk mitigation. But what is most delightful about it is Choi's thoroughness, his refusal to simplify or overreach. There is something truly delicious about finding a solution to a "new" problem in a strain of case law from the 1960s that most people probably don't know existed. This is what common law is good at: analogies. And this is what the best technology lawyers among us are best at: finding those analogies, and explaining in detail why they fit technology facts.

Cite as: Margot Kaminski, *Lessons from Literal Crashes for Code*, JOTWELL (October 3, 2019) (reviewing Bryan H. Choi, *Crashworthy Code*, 94 **Wash. L. Rev.** 39 (2019)), https://cyber.jotwell.com/when-computer-code-crashes-get-corporeal/.