

The Old Limitations No Longer Apply — Part 2: The Rentier's Last Stand

By Austen Tucker · 2026

A Singularity Log Special · Part 2 of 4

In this series

1. The Price of Admission
2. The Rentier's Last Stand — this essay
3. What Happens When We Can Make Our Own?
4. What's Left to Build?

Previously

- Part 1: AI lowers the price of turning an intention into working software, letting the idea come before the apprenticeship.

The same desk, computer, bird, game pieces, guitar, workshop, and garden drawn in dark ink, with added crosshatching, tool details, and plant shapes.

For years, we used somebody else's software because building what we wanted cost too much time and effort. For an individual, that was a reasonable compromise. Across an economy, it handed companies control over the tools we depended on.

The Rentier's Last Stand

Rentier capitalism means collecting income through ownership and control of access. Software helped it flourish. Building a useful application required expertise, labor, and infrastructure. Capital assembled those resources, produced tools that simplified our lives, and sold us access.

Plenty of those tools were worth paying for. Many still are. Buying a reliable solution can spare you weeks of work.

RetroMachine's Games catalog filtered for Super Mario, showing readable game cards, system labels, search, and TV controls.

The Games catalog brings search, system filters, cover art, and TV controls into one local interface.

But once a company owned the software through which people accomplished something important, it could charge for keeping the gate open. Customers had built their workflows, data, and habits around the product. Switching could cost more than another month of rent.

Hosting, maintenance, security, support, and development cost money. A subscription can pay for those things. It can also exploit a dependency. You didn't have to keep inventing new Photoshop versions if you could keep owning the gate through which people accessed it.

RetroMachine's Videos catalog shows Xena episodes with cover art, episode titles, year labels, and a TV-episode filter.

The Videos tab puts episodic media in the same home system as the game catalog.

The bargain rested on an economic fact: most people couldn't afford to produce their own alternatives. The apprenticeship and labor cost more than paying the owner. That fact is beginning to change.

Then I saw ArtCraft, and the scale of the argument changed. Its creator announced he'd used AI to build open-source alternatives to seven Adobe applications: Photoshop, Illustrator, Premiere, Lightroom, After Effects, InDesign, and Acrobat Pro. People have built professions and businesses around those tools. Here was a developer attempting alternatives across the whole family, with code other people could inspect, change, and improve.

I care whether the software works. A broken application can wreck a deadline. The creator acknowledges unfinished functionality and file-format support; the applications haven't demonstrated complete parity. But bugs don't dispose of the economic argument. A project I would once have associated with an enormous institutional investment had become something a developer could initiate with AI assistance. The cost of trying had changed before every feature worked.

He brings years of technical experience and describes funded collaborators. Expertise hasn't evaporated. But if recreating familiar capabilities becomes a tractable engineering project, the incumbent has to defend its position with more than the expense of producing them the first time.

That's why CTOs around the country are trying to outrun what they're deeming the SaaSacre. Software used to be a moat: the labor and capital needed to build an alternative protected the owner from competition. An industry that spent decades automating everyone else's work is confronting the automation of its own production.

A prototype isn't a reliable commercial service. Not every application can be rebuilt in an afternoon. But a business whose chief advantage was the cost of reproducing its product faces a customer who can choose between renting a convenience and making one. My own entertainment system began crossing that gap in an afternoon, even with the TV-control integration left to fix.

The Kill Zone

In the technology industry, we called it the kill zone. You didn't compete with the FAANGs—Facebook, Amazon, Apple, Netflix, and Google—without a good reason. Once you threatened them, they could buy you, copy you, or use their distribution and capital to crush you.

Building a better product wasn't enough. You had to survive long enough for anybody to use it. Economists Sai Krishna Kamepalli, Raghuram Rajan, and Luigi Zingales developed a model of the kill zone in which the prospect of acquisition by an incumbent discourages the adoption and investment a competing platform needs to grow. The expectation of being swallowed can damage a market before anyone gets swallowed.

Producing software required capital. Distributing it could depend on infrastructure incumbents controlled. Challenging them demanded more capital. The cost of implementation was the first barrier; surviving as a business was another.

AI doesn't erase distribution, customer relationships, data, infrastructure, or network effects. Owning the platform on which other people's software runs remains a powerful position. Companies with the money to operate AI infrastructure can use these tools too. Cheaper implementation can strengthen their advantages.

But it also lets a small team attempt work that once required a large organization. And it lets an individual build useful software without becoming a company. That possibility interests me because the kill zone operated around commercial competition. A startup needed investment, customers, growth, and revenue to survive.

What happens when the software isn't a startup? When its builder doesn't need venture capital or millions of customers and has no intention of selling the company? An incumbent can offer a better product. But commercial pressure has less purchase on a hobbyist who isn't trying to win the market. You can't kill a startup that was never trying to be a startup.

Platform restrictions and dependence on commercial tools can still hurt that hobbyist. The difference is what counts as success. If the project works for the person who wanted it, it has accomplished its purpose. Publish the code, and somebody else can build on it without persuading an investor the market is worth entering.

Someone who wants an entertainment system, a game, or a tool for their own life has a different definition of success. And now they have the means to pursue it, capital be damned.