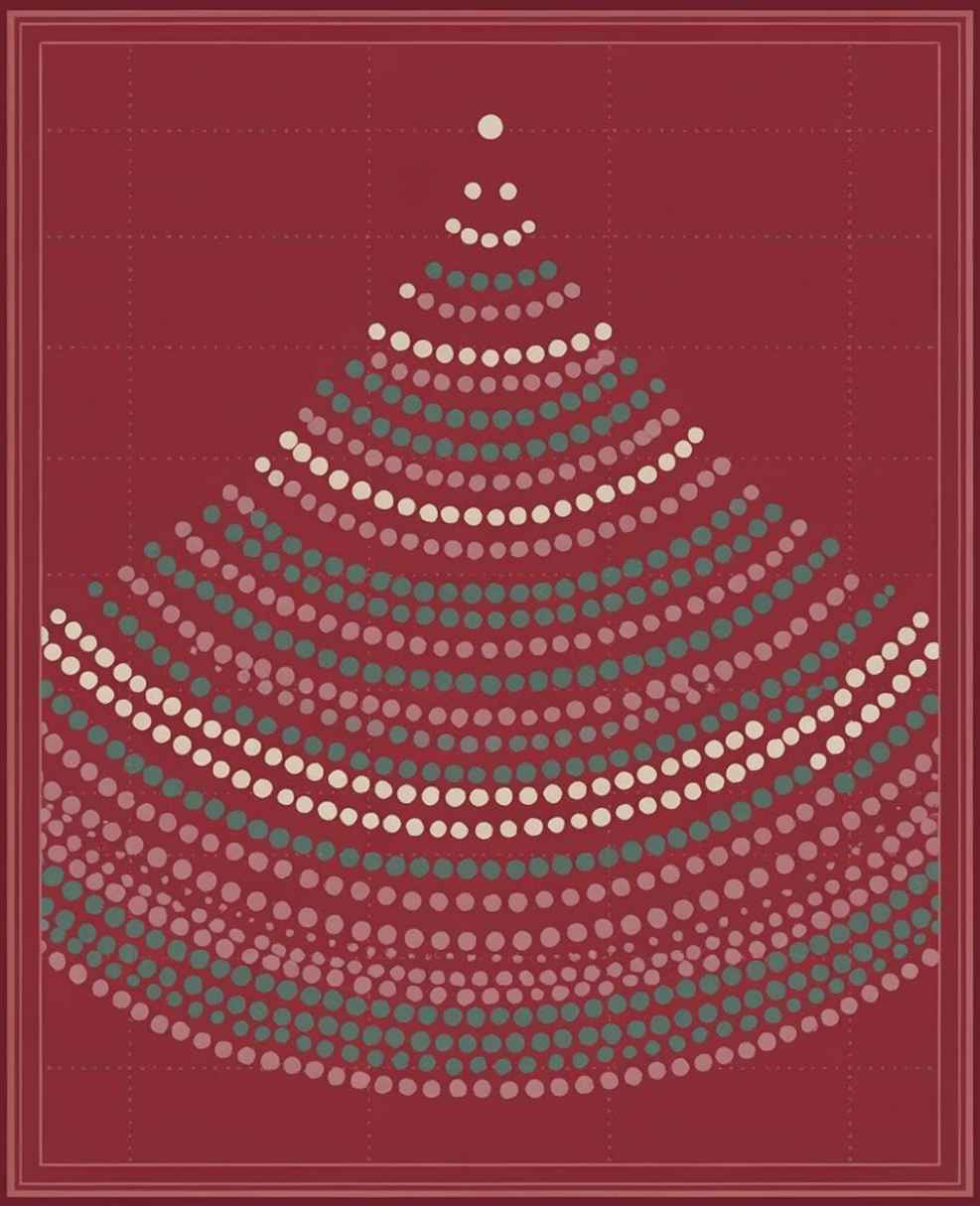
COURAGE GARIKAI



ZERO MARGINAL COST

Software, Replication, and the Economics of Abundance

CC BY 4.0 · COLDCODE SI LABS · HARARE

For every developer who has ever contributed to open source on nights and weekends without knowing who would use their work.

And for the generations who will inherit the abundance we choose to build instead of the walls we choose to erect.

Zero Marginal Cost

Software, Replication, and the Economics of Abundance

Courage Garikai

ColdCode Si Labs · Harare

About this manifesto

Software is the first major industry in human history where the marginal cost of production is approximately zero. Once a piece of software is built, copying it costs nothing. Distributing it over the internet costs almost nothing. A customer downloading an app does not consume inventory, does not require factory production, does not deplete raw materials. The thousandth copy costs the maker essentially the same as the first copy: pennies for bandwidth and server time.

This economic fact — near-zero marginal cost — is the most important thing about software as an industry and as a creative medium. It shapes business models, pricing, distribution, competition, labour, culture, and power. It makes open source possible. It makes SaaS possible. It makes piracy trivially easy. It makes platform monopolies enormously profitable. It makes freemium viable. It makes digital abundance achievable and digital enclosure profitable.

This short book (exactly 33 pages, deliberately brief) is a manifesto and a working paper about the economics of zero marginal cost software. It is deliberately published under CC BY 4.0, in the tradition of the

Distributed Cognitive Autonomous Infrastructure working papers and Facts.json as Infrastructure. It argues that:

- Zero marginal cost is the defining economic property of software.
- The choice between abundance and enclosure is political, not technical.
- Open source, free software, and creative commons are rational responses to zero marginal cost.
- Software monopolies are also rational responses, enabled by network effects and legal enclosure.
- For the global periphery — Africa, the global south, marginalized communities — the economics of zero marginal cost creates unprecedented opportunities for access to knowledge, tools, and infrastructure.
- Building for zero marginal cost is a design choice and an ethical position.
- The future of software will be determined by whether we treat zero marginal cost as an opportunity to be shared or a moat to be enclosed.

License

This work is published under the Creative Commons Attribution 4.0 International License (CC BY 4.0). You are free to share, adapt, copy, distribute, transmit, and build upon this work for any purpose, even commercially, as long as you give appropriate credit to the original author.

I · The economics

Marginal cost defined

In classical economics, marginal cost is the cost of producing one additional unit of a good. For a car manufacturer, the marginal cost of the 10,000th car includes steel, rubber, labour, factory time, components, and shipping — perhaps $25,000. For a farmer, the marginal cost of an additional bag of maize includes seed, fertilizer, land, labour, and transport. For a restaurant, the marginal cost of an additional plate of food includes ingredients and a small amount of kitchen labour.

For software, the marginal cost of the 10,000th copy is approximately: - Bandwidth: $0.001–$0.01 (a few megabytes transferred over CDN). - Payment processing: 2.9% + $0.30 (if sold, but this is transaction cost, not production cost). - Server costs: fractions of a penny per additional user for static distribution. - Total: less than one cent.

This is historically unprecedented. No other industry has ever produced goods where the cost of replicating the product is effectively zero. Books cost money to print. Music cost money to press onto records or CDs before digital distribution. Films cost money to distribute to theatres. Software — once written — can be replicated infinitely at negligible cost. (Digital music and video have since joined software at near-zero marginal cost, but software arrived first and most thoroughly.)

Zero marginal cost does not mean zero total cost. Software has enormous fixed costs — the cost of writing it, testing it, designing it, maintaining it, supporting it, updating it. The first copy of Microsoft Windows cost billions in R&D. The millionth copy cost pennies to distribute. This cost structure — high fixed costs, near-zero marginal

costs — is the economic structure of digital goods, and it creates unusual market dynamics.

Why software monopolies form

In markets with high fixed costs and near-zero marginal costs, natural monopolies form easily: - The largest producer can spread their fixed costs across more customers, allowing lower prices while remaining profitable. - Network effects (the product is more valuable when more people use it) reinforce dominance. - Switching costs lock users into the dominant platform. - Once a company captures the market, competitors face the prohibitive cost of building a competing product while unable to spread their fixed costs across enough customers to compete on price.

This is why operating systems, office suites, social networks, search engines, app stores, and cloud platforms tend toward monopoly or oligopoly. It is not that the dominant product is always the best; it is that the economics of zero marginal cost and network effects make competition structurally difficult once a standard emerges.

This economic reality does not justify monopolistic behaviour. It explains it. The same economics that make software monopolies likely also make the case for intentional public alternatives — open standards, open source, public platforms — to prevent monopoly rents from being extracted indefinitely on goods that cost nothing to replicate.

The enclosure of digital abundance

If software costs nothing to copy, everyone on earth could, in principle, have access to all the world's software, knowledge, and culture for nearly nothing. That is the radical promise of digital abundance.

What prevents this is not technology but law and business models. Copyright law, patents, DRM, paywalls, software as a service subscriptions, app store gatekeeping, licensing agreements, and

proprietary formats all enclose the digital commons, creating artificial scarcity where the underlying economics permit abundance.

Some enclosure is economically necessary: software developers must be paid, servers cost money, maintenance requires ongoing investment. If no one pays for software, no one can afford to build it (unless it is built by volunteers, grant funding, or governments — all of which exist but do not cover all software). The question is not whether any enclosure is justified, but where the line between fair compensation and rent extraction lies.

When a pharmaceutical company sells a pill that costs $0.50 to manufacture for $500 because of patent protection, that is a form of rent extraction enabled by legal monopoly. When a software company sells a product that costs $0.01 to distribute for $500/month per seat with no per-customer service costs, the dynamic is similar — though software markets have more competition and faster innovation than pharmaceutical markets.

The honest conversation about software economics is: how do we compensate creators fairly for the enormous fixed cost of building software, while taking advantage of the fact that copies cost nothing and digital abundance is possible? The answers are not all free software or all paid software; they require nuance, experimentation, and an awareness of the unusual economics at play.

II · Responses to zero marginal cost

Open source and free software

The open-source movement and the free software movement are the most mature responses to zero marginal cost. They take the economics of costless replication seriously: since copying costs nothing, copying should be permitted. The GNU GPL (copyleft) ensures that this freedom is preserved even when software is modified. Permissive licenses (MIT, BSD, Apache) permit anyone to use, modify, and redistribute the software, including in proprietary products.

Open source is not anti-commercial. Red Hat (sold to IBM for $34 billion) built a multi-billion dollar business selling services, support, and guarantees around free Linux software. Countless companies build commercial products on open-source foundations. The economics work: you pay for services, support, reliability, expertise, and integration — not for the bits themselves.

The free software movement (Stallman, FSF) makes an ethical argument alongside the economic one: that users have a right to inspect, modify, and share the software they use. Proprietary software that denies users these rights is, in Stallman's framing, unjust — it treats users as helpless consumers dependent on a vendor for control over their own computing. This argument is strengthened by zero marginal cost: denying users the ability to copy and modify software is not an economic necessity; it is a choice enforced by law and DRM.

SaaS: pay forever

Software as a Service (SaaS) is the dominant commercial response to zero marginal cost. Instead of selling a perpetual license (where the customer buys a copy once and uses it indefinitely), SaaS sells access —

software hosted by the vendor, paid for on a recurring basis (monthly or annually). The customer never owns a copy; they rent access.

SaaS has genuine advantages: automatic updates, cloud hosting, no local installation, cross-device sync, managed security. But it also fundamentally changes the economic relationship: - The customer pays forever, rather than once. - The vendor controls the software completely; they can change features, raise prices, or discontinue service. - The customer's data lives on the vendor's servers. - There is no option to keep using the software after you stop paying — the software stops working.

SaaS extracts revenue over time from a product with zero marginal cost, producing recurring revenue streams that are valued extremely highly by investors. For users, SaaS is often convenient but creates lock-in and perpetual expense. For vendors, it solves the "piracy" problem (no copy is ever distributed; access is controlled) and ensures ongoing revenue.

SaaS is a rational business response to zero marginal cost. Whether it is good for users depends on the product, the pricing, the switching costs, and whether the ongoing service justifies the ongoing cost.

Freemium, ads, and surveillance

A third response is freemium: give away a basic version for free (possible because marginal cost is zero), charge for premium features or higher usage. This works because the cost of serving a free user is near zero, and free users can be converted to paid users or monetized indirectly.

The indirect monetization route is advertising and data collection. If the marginal cost of serving a user is near zero, you can afford to give the product away for free and monetize attention (advertising) or data (selling user information to advertisers, brokers, or data aggregators). This is the economic model that funds Google Search, Facebook, free mobile apps, and most consumer internet services. It creates perverse

incentives — if users are not paying, they are the product being sold — but it is economically rational in a zero-marginal-cost world.

One-time purchase

The traditional model — buy the software once and use it forever — still exists (some indie apps, some desktop software, games). It has declined because it produces revenue only once while costs (maintenance, updates, support) are ongoing, and because piracy is trivially easy for distributed software. It works best for games (where ongoing updates are optional and many customers buy at launch) and niche professional tools with small customer bases.

III · Zero marginal cost on the periphery

Africa as a beneficiary of abundance

Zero marginal cost is the single most powerful technological force for developing countries. For the first time in history, people in low-income countries can access tools, knowledge, education, and software that previously required expensive physical goods, licenses, or infrastructure:

- A student in rural Zimbabwe with a $25 smartphone can access more knowledge on Wikipedia than a university student in the 1990s could access in a well-stocked library — for the cost of a data bundle.
- A startup in Nairobi can run open-source software (Linux, PostgreSQL, Python, Django, React) without paying a penny in licensing fees — software that would have cost millions of dollars in proprietary license fees 25 years ago.
- A clinic in Malawi can use open-source medical records software for free instead of paying $50/user/month to a proprietary vendor.
- A developer in Kampala can learn programming from free resources online and build software without buying textbooks, development environments, or server licenses.
- A school in rural Tanzania can run open educational resources on a Raspberry Pi server without internet access, distributing knowledge at marginal costs of hardware alone.

This abundance is not evenly distributed — hardware costs money, internet costs money, data costs money, and devices cost money — but the software and knowledge themselves, once built, can reach anyone at near-zero cost. This was not true of any prior technology. Books, medicines, machines, and buildings cannot be copied infinitely; software and digital knowledge can.

Piracy as market signal

Software piracy is widely condemned as theft, but in zero-marginal-cost economics it is also a market signal: it means the vendor's price is far above the marginal cost and above what buyers are willing or able to pay. In high-income countries this is seen as criminal; in low-income countries where vendor pricing is set at US/EU price levels unaffordable to most people, piracy is often the only way people access software.

The rational vendor response to piracy in low-income markets is not lawsuits (which are unenforceable and unpopular) but price discrimination: lower prices for markets where purchasing power is lower, educational discounts, free tiers for non-commercial use, or open-source versions. Adobe and Microsoft have both experimented with discounted pricing in developing markets after years of piracy; other companies never adjust and lose those markets to open-source alternatives (which are free, legal, and often better).

Open-source software wins decisively in price-sensitive markets because it is not just cheap — it is zero cost, legally free, with no piracy required. This is why Linux, Android (based on Linux), open-source databases, open-source web servers, and free developer tools dominate developing-world infrastructure. Zero marginal cost means free software is not just morally preferable; it is economically inevitable in markets that cannot pay first-world prices.

Building for the periphery at zero marginal cost

The opportunity for African software builders is enormous precisely because of zero marginal cost: you can build software once and sell it globally without incurring proportional production costs. A developer in Harare can sell a SaaS tool to customers in 30 countries as easily as a developer in San Francisco — the marginal cost of serving a customer in New York is the same as serving one in Nairobi. Software businesses do not need factories, shipping, customs, or physical supply chains. A laptop and an internet connection are enough to build and distribute globally.

The challenge — explored throughout this series of books — is payments, connectivity, hardware, and access. But the software itself is free to replicate. That is a structural advantage no previous generation of African entrepreneurs ever had.

V · History of abundance and enclosure

The printing press as precedent

Johannes Gutenberg did not invent movable type (the Chinese and Koreans had it centuries earlier), but his mechanical printing press, developed around 1440 in Mainz, Germany, brought the marginal cost of book reproduction down by orders of magnitude. Before Gutenberg, a single hand-copied book required months of labour from a skilled scribe and cost as much as a house. A Gutenberg Bible cost the equivalent of three years' wages for a clerk — expensive, but perhaps one-hundredth the cost of a manuscript Bible. Within 50 years, an estimated 20 million books had been printed in Europe.

The consequences were profound and unpredictable. The Catholic Church lost its near-monopoly on scriptural interpretation when printed Bibles in vernacular languages allowed individuals to read the text themselves. The Protestant Reformation spread because Luther's 95 Theses were printed and reprinted across Europe in weeks. Scientific knowledge accelerated because researchers could publish findings to a wide audience rather than copying letters by hand. Literacy rates rose. National languages standardized around printed dialects. Political authority was challenged. The modern world, in many ways, is a product of the printing press.

Every new copying technology produces similar upheavals. Print reduced the marginal cost of books. Sound recording reduced the marginal cost of music. Film reduced the marginal cost of performances. Photocopying reduced the marginal cost of printed pages. Each was met with panic by incumbent interests (the Church attempted to ban vernacular Bibles; music publishers fought player pianos, radio, cassette tapes, and MP3s; film studios fought VCRs), and each produced abundance that could not be put back in the bottle.

Software is the most powerful copying technology ever invented, because it copies itself. The economic and social upheavals will be correspondingly larger. We are still in the early decades.

The free software movement

In 1983 Richard Stallman announced the GNU Project: a complete free operating system, written from scratch, with the explicit goal of giving users the four freedoms: to run the program for any purpose, to study how it works and modify it, to redistribute copies, and to distribute modified versions. In 1985 he published the GNU Manifesto and founded the Free Software Foundation.

Stallman's argument was explicitly moral rather than economic: proprietary software that denies users these freedoms is unjust. It subjugates users to the vendor, who controls their computing, restricts their behaviour, and can remotely disable or alter software. Free software is a matter of liberty, not price (free as in freedom, not free as in beer — though the economics work out to near-zero price as a consequence).

The GNU GPL (General Public License), published in 1989, is Stallman's crucial legal innovation. It uses copyright law — the very legal mechanism that encloses software — to enforce freedom: anyone who distributes GPL-licensed software (modified or unmodified) must pass along the same freedoms. They cannot enclose the software and add additional restrictions. This copyleft provision ensures that software that begins free remains free through all future generations of modification, preventing the enclosure of public code in proprietary products.

Open source as development methodology

In 1998, a group of developers including Eric Raymond, Bruce Perens, and Linus Torvalds coined the term "open source" as a deliberate alternative to "free software." The shift was partly strategic: "free" was ambiguous in English (free as in freedom vs free as in price), and the

free software movement's moral framing was alienating to corporate America. Open source reframed freely shared source code as a superior development methodology rather than an ethical cause: code that anyone can read and modify will be better, more reliable, and more secure than proprietary code developed behind closed doors.

The argument was proven in practice. Linux, Apache, Sendmail, Perl, Python, PostgreSQL, MySQL, and other open-source projects came to dominate internet infrastructure in the 1990s and 2000s. By 2026 almost the entire internet runs on open-source software; every smartphone (whether Android or iOS) is built on open-source foundations; every major technology company contributes to and depends on open source.

The difference between free software and open source is not technical (the code is largely the same) but philosophical. The free software movement asks "is this just?" The open-source movement asks "does this produce better software?" Both answers matter.

The enclosure of the commons, round by round

Despite the success of open source, digital enclosure proceeds continuously: - **Software patents** have been used to attack open-source implementations and extract rents from companies that use widely adopted algorithms. - **DRM (Digital Rights Management)** — legal in the US under the DMCA and in the EU under similar laws — makes it a crime to circumvent technological restrictions on copying, even for legal purposes. - **App store gatekeeping** — Apple and Google control 99% of mobile app distribution, taxing all software sales at 30% and imposing arbitrary rules on developers. - **SaaS and walled gardens** — when software runs only on the vendor's servers and is accessed via subscription, users never hold a copy and never have the rights the GPL protects. - **Cloud platforms** — AWS, Azure, and Google Cloud have built enormous businesses hosting open-source software as a service without always contributing back, capturing the value of open-source projects while contributing little. - **Training data enclosure** — AI models trained

on public data (websites, books, images, code) create proprietary services that are then enclosed behind paywalls and rate limits, with no compensation to the creators of the training data. - **Copyright term extension** — copyright terms have been repeatedly extended (in the US, from 28 years to life of the author plus 70 years), locking cultural works out of the public domain for generations.

These enclosures are not inevitable facts of nature; they are legal choices, made by legislatures under lobbying pressure from incumbent industries. They can be changed by legislatures as well.

VI · Technical architecture for zero marginal cost

Design for distribution

Software designed for abundance is architected for easy distribution: - **No mandatory server.** Desktop applications, static sites, offline-first apps, and P2P systems can be copied and run without depending on a single vendor-controlled server. Web apps that require a central server are inherently enclosable. - **Open formats.** Data should be stored in standard, documented, non-proprietary formats so users can migrate away. Formats that require proprietary software to read are locks. - **No DRM.** Digital rights management is technically a mechanism for preventing copying and legally a set of restrictions that override user rights. Software built for abundance does not include DRM. - **Small, portable, buildable.** Software that is small, has few dependencies, and can be built from source by anyone is easier to distribute, preserve, and fork than software with hundreds of dependencies and complex build requirements. - **Decentralized protocols over centralized platforms.** Email (SMTP), the web (HTTP), and BitTorrent are protocols anyone can implement; Twitter/X, Facebook, and Slack are centralized platforms controlled by a single company. Protocols outlast platforms.

Federated and P2P systems

Federated and peer-to-peer architectures distribute software in ways that no single entity can enclose: - **Federated systems** (email, Matrix, Mastodon, XMPP) use shared protocols that allow independent servers to interoperate. Anyone can run a server; users can choose a provider; no single company controls the network. - **Peer-to-peer systems** (BitTorrent, IPFS, Bitcoin, local mesh networks) distribute data directly

between users without central servers, making censorship and shutdown technically difficult.

These architectures trade some convenience for resilience and user control. They are harder to monetize through walled-garden business models (which is why venture capital prefers centralized platforms), but they are aligned with the economics of zero marginal cost: when copying is free, distribution is cheap, and no gatekeeper is required.

Preservation as a design requirement

If software costs nothing to copy, it can in principle be preserved indefinitely — but in practice, software disappears when servers shut down, companies fail, or DRM servers are turned off. Building for long-term preservation means: - **Self-contained artifacts.** A program that can be compiled from source, or a static site that can be downloaded and served locally, will outlive web services that depend on a running server. - **No build-time secrets.** Software that requires secret API keys or proprietary services to run cannot be preserved. - **Emulation-friendly.** Even if the original hardware disappears, software that runs on emulated platforms can be preserved. This is why retrocomputing communities still run software from the 1970s. - **Documented build process.** Software that builds with a `make` command in 2026 should, with reasonable effort, build in 2056. Pin dependencies, document the build environment, and avoid dependencies on services that may not exist.

The copy button

The simplest piece of architecture for zero marginal cost is the copy button. Software that allows users to copy it, export their data, fork the code, and run it independently respects the economics of abundance. Software that prevents copying — through technical, legal, or platform restrictions — is fighting the fundamental economic fact that bits copy for free.

Build in the copy button. Make it easy for users to take their data and leave. Not only is this honest; it forces you to keep improving the product to earn users' continued loyalty rather than relying on lock-in.

VII · Paying for software in a zero-marginal-cost world

How creators can get paid

Zero marginal cost does not mean software creators should work for free. Several models work honestly without enclosing abundance:

- **Selling services** (support, implementation, consulting, customization, hosting, training) around open-source software. The software itself is free; expertise, reliability, and time are not. Red Hat, Docker, HashiCorp (in parts), and many open-source businesses use this model.
- **Selling convenience.** SaaS can be justified when the vendor provides genuine ongoing value (hosting, updates, monitoring, uptime guarantees, backup) that users are willing to pay for rather than running the software themselves. The key is that the convenience must be real, not artificial lock-in.
- **Voluntary support.** Patreon, GitHub Sponsors, donations, and tip jars allow users who value the software to support it voluntarily. This funds an enormous amount of open-source work, though it rarely scales to pay large teams.
- **Grants and institutional funding.** Governments, foundations, and corporations fund public digital infrastructure the same way they fund roads, research, and public health. This is how some of the most important software infrastructure (OpenSSL, Python, cryptographic libraries) is funded, though it remains underfunded relative to its importance.
- **Dual licensing.** Offer the software under a strong copyleft license (GPL/AGPL) for free, and sell a commercial proprietary license to companies that want to build closed products on top of

it without reciprocating. This funds open development while preserving freedom for the open-source version.

- **Bounties and crowdfunding** for specific features. Users collectively pay for features they need; the software remains free.

Each model has strengths and weaknesses. None is perfect, but together they demonstrate that creators can be paid in a zero-marginal-cost world without artificial scarcity.

What rent extraction looks like

Rent extraction occurs when the price charged far exceeds the cost of providing the service and depends on artificial monopoly rather than real value: - A software company charging $500/user/month for a product with no per-user costs, defended by patents and lock-in, extracts rent. - An app store charging 30% on all transactions while banning competing payment methods extracts rent. - A DRM-locked subscription service that prevents users from accessing content they paid for when they cancel extracts rent. - A patent holder who does not produce a product but sues companies that independently implement the patented technology extracts rent.

The test is simple: if the price would collapse in a competitive market where users could legally copy the software and run it themselves, the excess above the cost of service provision is rent, enabled by legal or technical enclosure.

VIII · The next 50 years

What abundance looks like

If we choose abundance over enclosure, the next 50 years could produce a world where: - Every person on earth with an internet connection has free access to the world's knowledge: educational materials, scientific literature, reference works, software tools, and cultural works that are out of copyright or freely licensed. - Open-source software runs the majority of the world's critical infrastructure, maintained by a combination of public funding, corporate contributions, and volunteer effort. - Software tools for education, healthcare, agriculture, governance, and commerce are available to governments, NGOs, and small businesses in every country without licensing fees. - New creators are not locked out by walled gardens or patent thickets; the digital commons provides a foundation anyone can build on. - Software innovation is driven by what users need rather than what maximizes rent extraction.

This is not a utopian vision. It is a realistic description of what is technically possible given that copies cost nothing. The obstacles are not technical; they are political, legal, and economic.

What enclosure looks like

If we continue down the path of enclosure, the next 50 years could produce a world where: - A small number of platform companies control access to all digital communication, commerce, media, and software, extracting rent on every digital interaction. - Software, media, and culture are accessible only by ongoing subscription; nothing is owned, everything is rented, and access can be revoked at any time for any reason. - Patents, DRM, and platform terms of service prevent independent competitors from offering alternatives. - AI models trained

on public data are enclosed behind proprietary APIs, concentrating yet more power in a small number of companies. - The digital divide widens: users in wealthy countries pay high rents for access while users in low-income countries are priced out or subject to surveillance-heavy state-controlled alternatives.

Which future we get

The choice is ours, collectively, through the software we build, the licenses we choose, the companies we support, the laws we enact, and the cultures we create around technology. Zero marginal cost is an extraordinary technical fact — one that could democratize access to knowledge and tools on a scale the printing press only began. Whether it does depends on what we build and how we choose to license and distribute it.

Build open. Publish freely. Support public digital infrastructure. Remember that copies cost nothing, and abundance is a choice.

IX · AI and zero marginal cost

AI models as supercharged replication

Artificial intelligence models (large language models, image generators, speech models) represent a new phase of zero-marginal-cost economics. Training a model like GPT-4 or Llama costs tens to hundreds of millions of dollars in computing resources — enormous fixed cost, comparable to the R&D cost of a major pharmaceutical or a new semiconductor. But once trained, each inference (each generated token, each generated image) costs fractions of a cent — near-zero marginal cost, converging to zero as hardware improves.

This same cost structure (enormous fixed cost, near-zero marginal cost) makes AI models natural monopolies or oligopolies, concentrated in the hands of companies that can afford the training runs. OpenAI, Google, Meta, Anthropic, and a handful of Chinese companies dominate the frontier models. The same zero-marginal-cost economics that made Microsoft and Google into monopolies will concentrate AI power unless countervailing forces — open models, public investment, regulatory intervention — are applied.

Open models as counterweight

The open-source AI movement (Meta's Llama series, Mistral, Falcon, open-weight models from research institutions, and community-built models) is the direct descendant of the free software and open-source movements applied to AI. Open-weight models can be downloaded, run locally, fine-tuned, and distributed without paying an API provider. They make AI capability available to anyone with sufficient hardware — universities, startups, developing-world developers, individuals — without requiring permission or payment to a central provider.

This is a fork in the road. If frontier AI capability remains concentrated in three or four companies, the future will be one of centralized control, rent extraction, and limited access. If open models remain competitive with closed models, the same abundance that open source brought to software infrastructure will come to AI. Which future emerges depends on technical progress (can open models keep up with closed ones?), regulatory choices (will governments restrict open models for safety reasons while allowing closed ones?), and investment (will governments, foundations, and companies fund open AI research?).

AI training data and the commons

The sharpest unresolved tension in AI economics is training data. Today's frontier models are trained on vast quantities of text, images, code, audio, and video scraped from the public internet — books, blogs, news articles, Wikipedia, GitHub repositories, social media posts, photographs, videos — most of which was created by authors, photographers, programmers, and artists who were not compensated and did not consent to their work being used this way.

This is enclosure of a different sort: the commons of publicly available creative work is being mined to build proprietary AI systems whose outputs are then enclosed behind paywalls. Whether this is legally fair use, whether creators should be compensated, whether open models trained on public data are more or less justifiable than closed models — these questions are unresolved and will define the political economy of AI for decades.

My position is consistent with the rest of this essay: models built on public data should be released back to the public. Closed models trained on public data are privatizing the commons without reciprocation. Open models are far from a perfect solution (compensating creators remains a hard problem), but they at least return the fruits of that training to the commons.

X · Practical recommendations

For developers

- Choose open licenses deliberately. Use permissive licenses (MIT, BSD, Apache) for maximum adoption; use copyleft licenses (GPL, AGPL) when you want to prevent enclosure of your work; consider public domain (CC0, Unlicense) for reference data and tools that should be maximally free.
- Build with open formats and protocols. Avoid locking users into proprietary data formats.
- Contribute to open source. Even small contributions (documentation fixes, bug reports, typo corrections) help maintain the commons.
- Charge fairly for your work. You do not need to give everything away. But price based on the service and value you provide, not on artificial scarcity enabled by lock-in.
- Build export features into every product. Users should be able to leave with their data.
- Avoid DRM and artificial restrictions on copying. They punish legitimate users more than they stop pirates.
- Support the digital commons. Sponsor open-source projects you depend on. Contribute documentation and bug reports. Pay for tools that provide real value.

For companies

- Pay your engineers to contribute to open-source projects you depend on. It is infrastructure maintenance — as necessary as paying for servers.
- Use open standards over proprietary protocols where possible.

- Do not use patents to attack competitors implementing widely used techniques.
- If you build on open source, contribute back — code, funding, documentation, or infrastructure. The commons requires maintenance.
- Do not enclose public data behind paywalls or API gates without providing public-access options.

For governments

- Fund public digital infrastructure the same way you fund physical infrastructure. Open-source software, open datasets, security research, reference implementations, and protocol standards are public goods.
- Limit software patents. They do more to enable rent extraction than to encourage innovation in software.
- Protect fair use and users' rights to modify, reverse-engineer, and repair the software and devices they own.
- Mandate open standards and interoperability for government systems.
- Do not allow DRM to override legitimate user rights (backup, archiving, accessibility, interoperability, fair use).
- Support open AI models and public AI computing infrastructure so AI capability is not entirely controlled by three corporations.
- Require that publicly funded research, publicly funded software, and publicly collected data be released as open data and open source whenever possible. Publicly paid work should be public.

For users

- Use open-source alternatives where they meet your needs.
- Pay for software that provides real value, even if free alternatives exist — especially from independent developers.

- Demand data portability. Refuse tools that lock your data in proprietary formats.
- Share freely what you have created, if you can — writing, code, images, knowledge. The commons grows because people contribute to it.
- Do not accept "you don't own it" as a permanent condition of digital life. You should be able to own your tools, your data, and your creative work.

XI · Coda: 30 lines of code that changed the world

In 1991, a 21-year-old Finnish student named Linus Torvalds posted a message to a Usenet newsgroup:

> I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones...

He began writing the Linux kernel, released it under the GPL, and invited others to help. Thirty-five years later, Linux runs on every Android phone (3 billion devices), every supercomputer in the Top 500, most internet servers, all cloud infrastructure, the International Space Station, the Mars rovers, and the majority of embedded systems. It is arguably the most successful piece of technology ever built by a distributed volunteer community.

Linux did not become dominant because Linus was a genius businessman. He did not raise venture capital. He did not patent anything. He did not lock customers in. He wrote a kernel, released it freely under a license that ensured it stayed free, and invited contributions from anyone who was interested. Zero marginal cost did the rest: anyone could copy, modify, use, and redistribute the code for free, and they did — building servers, phones, embedded devices, companies, research projects, and entire industries on top of it.

Thirty lines of code, a free license, and a worldwide community of contributors produced a technology that powers most of modern civilization. That is what zero marginal cost makes possible: abundance built by volunteers, shared freely, available to everyone, impossible to enclose because the license prevents it, and too valuable to replace.

We have done it before. We can do it again. The cost is zero.

IV · Building for abundance

Design choices

Zero marginal cost is a fact. What we do with it is a choice. Software builders, policymakers, and users make design decisions that either extend digital abundance or enclose it:

- **Publishing code openly** extends abundance.
- **Choosing open formats** extends abundance.
- **Building open APIs** extends abundance.
- **Supporting open standards** extends abundance.
- **Paywalling basic facts** encloses.
- **Patenting software** encloses.
- **Locking in users with proprietary formats** encloses.
- **Using DRM to prevent legal copying** encloses.

Both approaches can produce successful businesses. Red Hat built a multi-billion dollar company on open source. Microsoft built a trillion-dollar company on proprietary software. But builders should be conscious of which path they are choosing and why.

The argument for public digital infrastructure

Some software is infrastructure. It should be built and maintained as public goods, the same way roads, bridges, standards, and protocols are public goods: - Operating systems, language runtimes, and core protocols (TCP/IP, HTTP, DNS). - Reference data (Facts.json — country codes, currency codes, public facility lists). - Encryption and security tools (OpenSSL, cryptographic libraries). - Open educational resources. - Government services and civic data. - Accessibility tools. - Reference implementations of open standards.

These are the digital equivalents of roads and bridges — infrastructure that everyone uses, that benefits from network effects, and that is inefficient to duplicate privately. They should be funded publicly (through governments, foundations, or consortia), built openly, and freely available to everyone at zero cost — exactly as their marginal cost suggests they should be.

The long view

In the long view of history, the invention of digital copying — software, digital media, the internet — is as significant as the invention of the printing press. The printing press reduced the marginal cost of a book from months of monk labour to minutes of mechanical reproduction. That changed civilization: it made universal literacy possible, enabled the scientific revolution, spread knowledge across continents, and undermined monopolies on information held by churches and states.

Digital distribution reduces marginal cost from pennies per page to effectively zero. It is early in this revolution — 50 years in, comparable to the first century of the printing press. We do not know what institutions, economies, and cultures will emerge from it. We do know that abundance is technically possible, and whether it is achieved is a political and social choice, not a technical one.

Zero marginal cost software can create a world where every person with an internet connection has access to the world's knowledge, tools, and culture — or a world where a handful of platform companies extract rent from every digital interaction, enclosing abundance behind paywalls, patents, DRM, and walled gardens. Both futures are possible with the same technology.

Building software with an awareness of zero marginal cost means asking, for everything we build: does this extend abundance or enclose it? Does this give users more capability and control, or less? Is this price justified by the actual cost of providing the service, or is it rent extraction enabled by monopoly and legal enforcement?

About the author

Courage Garikai builds and runs autonomous business systems from Harare, Zimbabwe, and writes about what actually works. The Coldcode Si Labs Library is the written record of that practice — forty-four titles and counting, each one built to be finished, kept, and passed on.

Also by Courage Garikai

The Leapfrog Series — The Leapfrog Thesis · The Leapfrog Playbook · The Chisel and the Light

The 2026 Business Series — The Tender Desk · The Governable Enterprise · The Human-Approval Loop · Mobile Money Rails · From Spreadsheet Interface to Governable Autonomy · Invisible Proof · The Peripheral Internet · The Contact-Enquiry Price · The One-Person Studio · Facts.json as Infrastructure · The Honest Bot · Slow Software

Books — Intelligence, Data and Judgement · Replaced · Pinterest Goldmine · The Ledger and the Shopkeeper · Operatorship · The Tight Bucket · Emotional Engineering · Nobody Was Watching · Proof of Mechanism · The Mechanism Auditor · Traffic Is the New Oil · Unchained Blueprint · The Point of Failure · The Musk Mirror · The Rocket Man Reckoning · The Bliss Attractor · Clickbait · Why Is Africa Still Poor? · The Complete Science and Art of Publishing a Great Book

Working papers — Constraint-Native Business Model Design · Simulated Completeness · Unwitnessed Autonomy · Grounded Cadence · Order Without Custody · The Mechanism Audit · The Sovereign Micro-Enterprise · Distributed Cognitive Autonomous Infrastructure · Zero Marginal Cost

Free samples of every title — PDF and EPUB — are at library.coldcode-si-labs.workers.dev.

If this book earned its hours, pass it on. Recommendation is how books like this are found.

Colophon

This edition was assembled with Python, typeset with WeasyPrint using open fonts, and built into PDF (A4) and EPUB formats by Coldcode Si Labs, Harare, Zimbabwe — November 2026.

Call to build

Build open tools. Publish open data. Contribute to open source. Use open formats. Avoid lock-in. Support public digital infrastructure. Price your work in a way that sustains you without exploiting the fact that copies cost nothing. Share what you can. Sell what you must. Build as if the future matters.

The economics are clear. The marginal cost is zero. The choice is ours.

Harare, November 2026