Time still matters as a software moat

Kabui, Charles

2026-07-15

Table of Contents

Introduction

A solo founder spends a year building software with AI while a few customers use it every week. They expose edge cases, request integrations, and reveal which problems are painful enough to pay for. The founder fixes the product and learns what not to build.

Then a well-funded team arrives with strong engineers and a fleet of coding agents. Can it become equal within another year?

It can copy visible features faster and produce more code. But equal software is not a matching screenshot. It must handle the same messy situations, fit the same workflows, earn similar trust, and retain customers.

This leaves a narrower claim: **time matters when it produces assets that a rival cannot buy or generate on demand.** A year of customer learning can be hard to compress. A year spent polishing unwanted features can be overtaken in weeks.

---

1. AI has crushed the value of an implementation lead

In a controlled experiment, Peng et al. asked developers to build a JavaScript HTTP server. Those using GitHub Copilot finished **55.8% faster** than the control group (arXiv:2302.06590). The narrow, unfamiliar task suited an assistant that could generate standard code.

A much larger field study found a smaller but still substantial gain. Cui et al. combined randomized trials at Microsoft, Accenture, and a Fortune 100 company. Across **4,867 developers**, access to a coding assistant increased completed tasks by **26.08%**, with larger gains among less experienced developers (Microsoft Research).

METR's two rounds show how quickly the baseline is moving. Its early-2025 trial found that 16 experienced maintainers took **19% longer** with AI on familiar repositories, but METR now labels that result outdated. Its 2026 update says newer tools likely provide more speed, although selection problems prevent a reliable estimate. The later study covered **57 developers, 143 repositories, and more than 800 tasks** (METR, 2026).

An ordinary feature backlog is a weak defense. If a competitor can describe the behavior, test it, and reach the same users, AI makes reproduction cheaper.

---

2. More code is not the same as an equal product

Google's 2025 DORA report drew on nearly **5,000 technology professionals** and more than 100 hours of interviews. **90%** of respondents used AI at work, and more than 80% believed it increased their productivity. AI use was positively related to delivery throughput and product performance, but it still had a negative relationship with delivery stability (Harvey and DeBellis, 2025).

Code can move faster while the full system becomes less reliable. DORA found that teams with automated tests, mature version control, loosely connected components, and fast feedback gained more. Teams with slow, tightly connected systems gained little or nothing.

The previous DORA report put numbers on the mismatch. A **25% increase in AI adoption** was associated with 7.5% better documentation, 3.4% better code quality, and 3.1% faster reviews, but also 1.5% lower delivery throughput and 7.2% lower stability (DORA, 2024). These survey associations do not prove causation, but coding speed cannot stand in for business performance.

Customers pay for outcomes that keep working inside their constraints. A lead survives when the first year produces knowledge and systems that support those outcomes.

---

3. Some assets resist calendar compression

Strategy researchers Ingemar Dierickx and Karel Cool gave this mechanism a name in 1989: **time-compression diseconomies**. Their argument was that some strategic assets are accumulated through a path of activity over time. Spending twice as much for half as long does not always create the same stock of know-how (Dierickx and Cool, 1989).

AI makes code easier to fit into a shorter schedule. A founder can still accumulate recurring support patterns, failed feature ideas, migration tools for unusual data, incident-driven reliability fixes, and judgment about which requests signal a broad need.

An incident can change monitoring, expose a hidden usage pattern, and lead to a different onboarding process. A renewal then reveals whether the change mattered. The value comes from the sequence, not only the final code.

A larger team can parallelize technical work, but not every sequence. Twenty engineers cannot produce twenty months of decisions from a customer who renews annually. They cannot instantly observe seasonal traffic or a rare failure.

An old product does not earn this advantage automatically. If the founder avoids customers, records nothing, or treats every request as strategy, elapsed time produces little useful knowledge.

---

4. Real customers produce knowledge that a clone cannot see

Source code reveals what a product does now. It rarely reveals the rejected alternatives or the evidence that selected the current behavior.

Ron Kohavi and Stefan Thomke recount what happened at Bing. A small change to ad headlines sat at low priority for more than six months. When an engineer finally ran an online experiment, the change increased revenue by **12%**, worth more than **$100 million a year** in the United States, without damaging user experience (Harvard Business Review, 2017). Experts had the idea in front of them and still misjudged its value. Real user behavior supplied the missing answer.

A year with customers can answer which setup step causes abandonment, which alert is ignored, which integration gets a deal approved, and which apparent bug is a workflow mismatch. A competitor can copy the resulting interface, but not inspect why one choice survived and another did not.

A challenger with distribution can recruit users and run experiments concurrently, so the incumbent has to keep learning. Unread feedback has no defensive value. It becomes useful when the team links it to behavior, tests a response, and uses the result in later decisions.

5. Customer data can compound, but AI can also copy experience

Hagiu and Wright model two forms of learning: pooling data across customers and learning from repeated use by the same customer. They show that competitive advantage depends on the shape of the learning process, differences between firms, the amount of accumulated data, and what customers believe about future quality (RAND Journal of Economics, 2023). More data is not automatically decisive.

AI can also package and transfer experience. Brynjolfsson, Li, and Raymond studied **5,179 customer-support agents**. An AI assistant raised issues resolved per hour by **14% on average** and **34% for novice and lower-skilled workers**, with little effect on the most experienced (NBER Working Paper 31161). It appears to have spread top agents' routines and accelerated learning.

A challenger can use the same mechanism. Public documentation, ordinary logs, hired employees, and standard practices allow a rival to absorb hard-won knowledge without reliving every mistake.

Customer-specific knowledge is harder to transfer because it keeps changing: permissions, data mappings, exception handling, trusted relationships, and the details of how one organization works. Static advice written once for everyone is much easier to copy.

---

6. An exceptional team can still catch up

A strong team can catch a solo founder when the product is a thin interface over a common model, the workflow is public, switching is easy, historical data adds little, and replacement is low-risk. It can build integrations in parallel, buy distribution, hire experts, and run many experiments.

Catch-up is harder when equality requires the same operating state. A customer may depend on cleaned records, custom permissions, audit evidence, support history, and integrations. Rebuilding the application does not migrate that state or earn permission to touch it.

A Danish labor study found the same gap between changed tasks and final outcomes. Humlum and Vestergaard linked adoption surveys with administrative records. Employers reorganized work and workers reported productivity benefits, yet the researchers found no detectable effect on earnings or recorded hours and ruled out effects larger than **2%** two years after ChatGPT's launch (NBER Working Paper 33777, revised 2026). AI changed tasks before it changed measured results.

The definition of equality changes the estimate. A comparable demo may take weeks. Matching retention, reliability, workflow fit, support, and trust can take much longer.

---

7. How to turn twelve months into a moat

AI should increase the rate of learning, not just the rate of code production.

- Link each request to evidence. Record the customer's job, the observed behavior, the requested change, and the outcome after release. A request without context is an opinion. A measured result before and after release becomes product knowledge.
- Keep a decision history of failed experiments, rejected features, incident causes, and the assumptions behind important designs. Competitors can see what shipped, but not every expensive wrong turn that preceded it.
- Build into the customer's workflow with data migration, permissions, audit trails, integrations, and reliable support where the market requires them. These investments matter when they remove real switching pain, not when they merely obstruct departure.
- Measure time from signal to tested change, repeated support problems, activation, renewal, incident rate, and customer expansion. Feature count rewards output even when the product is not improving.
- Keep the code understandable. AI-generated volume can create review and maintenance costs that consume the speed gain. The related analysis, *The AI coding slop loop will eat its own gains*, explains why small changes, strong tests, and clear ownership matter.

After a year, this record should show which problems customers encountered, what the team tried, and what changed retention or reliability. It lives partly in the product and partly in the founder's judgment.

---

8. Conclusion

A competitor can now reproduce visible software quickly. Reproducing what a team learned from a year of customer use is slower.

A one-year lead matters most when customer use creates knowledge, operating state, trust, and deep workflow fit. It matters much less when the product's value is mostly a copyable interface or feature set. An exceptional team can compress coding, hire expertise, and run work in parallel, but it still needs access to the right customers and enough real consequences to learn what the incumbent already knows.

- Time alone creates no defense. It matters when the product accumulates validated knowledge and customer-specific state.
- AI erodes implementation leads first. Controlled studies show large gains on scoped coding work and meaningful gains across thousands of professional developers.
- Customer feedback only becomes an advantage when it changes decisions. Unread feedback and unused data create no defense.
- AI can transfer documented experience. Its large gains for novice support workers show why public routines are easier to copy than changing customer context.
- A founder has to keep learning. Once that stops, the one-year lead begins to shrink.

One person with AI and a few engaged customers can build a serious lead in a year, provided each month of use improves the decisions made in the next.

References

1. Becker, J., et al. (2026). *We Are Changing Our Developer Productivity Experiment Design.* METR.
2. Humlum, A. & Vestergaard, E. (2026). *Still Waters, Rapid Currents: Early Labor Market Transformation under Generative AI.* NBER Working Paper 33777, revised March 2026.
3. Cui, Z. K., et al. (2025). *The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers.* Management Science.
4. Harvey, N. & DeBellis, D. (2025). *State of AI-Assisted Software Development 2025.* Google Cloud DORA.
5. Brynjolfsson, E., Li, D. & Raymond, L. R. (2025). *Generative AI at Work.* The Quarterly Journal of Economics, 140(2), 889-942.
6. DORA. (2024). *Accelerate State of DevOps Report 2024.* Google Cloud.
7. Hagiu, A. & Wright, J. (2023). *Data-Enabled Learning, Network Effects, and Competitive Advantage.* The RAND Journal of Economics, 54(4), 638-667.
8. Peng, S., et al. (2023). *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.* arXiv:2302.06590.
9. Kohavi, R. & Thomke, S. (2017). *The Surprising Power of Online Experiments.* Harvard Business Review.
10. Dierickx, I. & Cool, K. (1989). *Asset Stock Accumulation and Sustainability of Competitive Advantage.* Management Science, 35(12), 1504-1511.
