The Square Law of Code: Allocating the AI Productivity Windfall Across Quantity, Quality, and Security

Sara Malik and Naveed A. Aun

PakCrypt.Org, Pakistan
smk@pakcrypt.org

Abstract. AI code generation multiplies the rate at which software can be written. Practitioners read that multiplier as permission to write proportionally more software. This paper argues that the reading is arithmetically wrong. Verification effort in a system of N interacting features grows with the number of interactions, which is quadratic in N. Security effort grows with the number of exploitable compositions of those interactions, which is quadratic again. We formalize this as the *Square Law of Code*: a windfall W in raw production capacity supports a feature-volume increase of only q where $q + q^2 + q^4 \leq 3W$. The consequence is blunt. A hundredfold windfall buys roughly a fourfold product. The remainder of the windfall belongs to testing and to security, in the ratio q^2 to q^4. We ground the exponents in four decades of empirical software engineering, from interaction-fault studies to attack-surface metrics, and we test the law against the 2023–2026 measurements of AI-assisted development: a 55% speed-up in isolated tasks, a 19% slowdown in mature codebases, a 7.2% stability loss per 25% increase in adoption, a doubling of duplicated code, and a security pass rate for generated code frozen at 55% for two years. We show that the same models that write insecure code now find vulnerabilities at industrial scale, which makes the law affordable: AI must fund its own audit. We close with a workflow, a metric set, and a macro-level argument that the law is a survival constraint rather than a best practice, because the adversary has received the same windfall. The constants in the law are stated as hypotheses with explicit derivations. The reader is invited to falsify them.

Keywords: AI-assisted programming · software complexity · attack surface · secure software engineering · productivity metrics · combinatorial testing.

1 Introduction

Every generation of tooling has promised to make programmers faster. Compilers did. Garbage collectors did. Package managers did. Each time, the industry converted the speed into volume, and each time the volume arrived with its own bill. Brooks named the bill in 1975: complexity is the essential difficulty of

software, and tools remove only the accidental part [6,7]. Large language models are the first tool that removes accidental difficulty at a rate measured in orders of magnitude. The essential difficulty is untouched. That asymmetry is the subject of this paper.

The intuition we attack is linear. If a developer can produce code ten times faster, ship ten times as much. The intuition is wrong for a reason that has nothing to do with AI. Features interact. The cost of establishing that a system works is a function of its interactions, and the cost of establishing that it resists an adversary is a function of the compositions of those interactions. Both functions are superlinear. A tool that accelerates only the writing of features leaves the superlinear terms unfunded. Unfunded verification becomes defect debt. Unfunded security becomes attack surface. Both compound.

We call the resulting allocation rule the *Square Law of Code*. In its simplest statement: for every doubling of shipped functionality, verification effort must quadruple and security effort must grow sixteenfold. The numbers 2, 4, and 16 are consequences of two exponents, and the exponents are consequences of two counting arguments. We give the arguments in Section 3 and the evidence for them in Section 4.

This is a position paper with a formal core. We make the following contributions.

1. A cost model with three activities (production, verification, security) and three elasticities (1, 2, 4) with respect to feature volume, each derived from a stated combinatorial assumption (Section 3).
2. An allocation theorem: the sustainable feature-volume multiplier under a capacity windfall W grows as $\Theta(W^{1/4})$. A hundredfold windfall supports a $4.1\times$ product (Theorem 1).
3. A synthesis of the empirical literature, 1976–2026, that supports the exponents and identifies where they are weakest (Sections 2 and 4).
4. An execution model in which AI funds the audit it makes necessary, with a three-role adversarial workflow and a metric set that survives the shift from output to outcome (Sections 5 and 6).
5. A macro-level argument, grounded in monoculture theory and in the first documented AI-orchestrated intrusion campaign, that the law is a defensive necessity (Section 7).

We state our epistemic position once, here, and hold to it. The exponents 2 and 4 are modeling choices supported by evidence, and we say precisely which evidence. The exact constants in any real organization will differ. The direction and the order of the growth will not. A reader who rejects the constants and keeps the exponents has accepted the paper.

2 What the Windfall Actually Is

The phrase "AI productivity windfall" is doing a great deal of work in current discourse. We pin it down before allocating it.

Task-level speed. The controlled experiment that anchors most optimism is Peng et al. [32]: developers with GitHub Copilot completed an isolated HTTP-server task 55.8% faster than controls. The task was self-contained, greenfield, and unreviewed. Those three adjectives define the domain in which the result holds.

Codebase-level speed. METR's randomized controlled trial with 16 experienced open-source maintainers on 246 real issues in their own repositories found the opposite sign [4]. With AI tools allowed, task completion took 19% longer. The developers had forecast a 24% speed-up and, after the study, believed they had received a 20% speed-up. The perception gap is itself a finding. METR's February 2026 follow-up with 57 developers and over 800 tasks estimated an 18% speed-up for returning participants and 4% for new ones, with confidence intervals crossing zero and with self-reported selection effects severe enough that the authors called the results "only very weak evidence" [24]. Read together, the two METR studies say that the windfall in mature code is somewhere between a mild slowdown and a mild speed-up, and that developers cannot feel which.

Delivery-level throughput. DORA's 2024 survey of roughly 3,000 practitioners found that a 25% increase in AI adoption was associated with an estimated 1.5% decrease in delivery throughput and a 7.2% decrease in delivery stability [10]. Individual satisfaction rose. System-level outcomes fell. This is the signature of a local optimization degrading a global one.

Code shape. GitClear's analysis of 623 million changed lines from 2023 to early 2026 reports that duplicated code blocks rose 81%, within-commit copy/paste rose 41% to 15.7% of changed lines, refactoring fell from 21% of changed lines in 2022 to 3.8%, and error-masking constructs rose 47% [14]. Heavy AI users out-produced non-users by a factor of four to ten. The material they produced was more duplicated, less moved, and more prone to swallow exceptions. This is exactly the code shape that Lehman's second law predicts when growth outruns the work required to hold complexity constant [21].

Security of the output. Veracode's benchmark of over 150 models on 80 tasks in four languages reports a security pass rate of approximately 55%, "virtually identical to where [it] stood two years ago," while syntactic correctness exceeds 95% [38]. Pass rates for cross-site scripting (15%) and log injection (13%) are close to coin-flip territory with the coin weighted against the defender. The finding is consistent with Pearce et al. in 2022 (roughly 40% of Copilot completions in security-relevant scenarios vulnerable) [31], with Perry et al. in 2023 (assisted users wrote less secure code and believed it more secure) [33], and with Sandoval et al. (a smaller effect in C, with the caveat that the tasks were narrow) [34]. The models got fluent. They did not get safe.

Developer belief. Stack Overflow's 2025 survey of 33,662 respondents found 84% using or planning to use AI tools, 45.7% distrusting their accuracy against 32.7% trusting it, 66% naming "almost right, but not quite" as their chief frustration,

and 45.2% reporting that debugging AI-generated code costs more time than it saves [37].

Synthesis. The windfall is real in the act of producing text that compiles. It is unproven at the level of delivered systems. It is negative at the level of verified security unless something is done about it. The rest of this paper is about what to do about it, and how much of the windfall doing it will cost.

3 The Square Law

3.1 Definitions

Let a system consist of N *features*. A feature is any unit of behavior that can be specified, shipped, and blamed. Endpoints, flags, commands, and integrations all qualify. Let the baseline system have N_0 features and let $q = N/N_0$ be the *volume multiplier*.

We partition engineering effort into three activities. *Production* (P) is writing the features. *Verification* (V) is establishing that the features do what they should, individually and together. *Security* (S) is establishing that the features cannot be made to do what they should not, individually and together, by an adversary who chooses the inputs.

Let $c_P(q)$, $c_V(q)$, $c_S(q)$ denote the effort each activity requires to keep its assurance level constant as volume grows, normalized so that $c_P(1) = c_V(1) = c_S(1) = 1$.

Let $W \geq 1$ be the *windfall*: the multiplier on total engineering capacity delivered by AI tooling, expressed in units of baseline effort. A team whose baseline capacity was 3 units (one per activity) has $3W$ units after the windfall.

3.2 Three propositions

Proposition 1 (Linear production). $c_P(q) = q$.

Writing qN_0 features costs q times writing N_0 features. This is the one activity where the linear intuition is correct, and it is the only one that AI tooling was designed to accelerate.

Proposition 2 (Quadratic verification). $c_V(q) = q^2$.

Derivation. A system of N features has $\binom{N}{2} \approx N^2/2$ feature pairs. The interaction-fault literature, summarized in Section 4, establishes that a large majority of field failures are triggered by the interaction of one or two parameters, and essentially all by six or fewer [19]. Holding pairwise assurance constant therefore requires effort proportional to the number of pairs. Doubling N quadruples the pairs. If one insists on t-way assurance, $c_V(q) = q^t$, and Proposition 2 is the conservative floor at $t = 2$.

Proposition 3 (Quartic security). $c_S(q) = q^4$.

Table 1. Sustainable volume multiplier q^* under windfall W (equal baseline split, exponents 1/2/4). "Debt" is the unfunded verification and security effort, in baseline units, if the whole windfall is spent on volume ($q = W$).

W	q^*	$c_V = q^{*2}$	$c_S = q^{*4}$	Debt_V	Debt_S
2	1.31	1.7	3.0	2	14
5	1.78	3.2	10.0	20	620
7.3	2.00	4.0	16.0	46	2,833
10	2.19	4.8	23.0	90	9,990
20	2.66	7.1	50.2	380	159,980
100	4.09	16.7	279.2	9,900	$\approx 10^8$
1000	7.36	54.2	2,938	$\approx 10^6$	$\approx 10^{12}$

Derivation. A vulnerability is a fault an adversary can reach. Modern exploitation rarely uses one fault. It composes them: an injection to reach a deserializer, a deserializer to reach a loader, a loader to reach the network. Each link in the chain is an interaction between two features, and the chain is an interaction between interactions. Let the interaction graph have $I \approx N^2/2$ edges. A two-link chain selects an ordered pair of edges: $\Theta(I^2) = \Theta(N^4)$ candidates. Holding chain-level assurance constant requires effort proportional to the candidate count. A k-link chain gives q^{2k}. Proposition 3 is the floor at $k = 2$.

We note the structural symmetry, which gives the law its name. Verification squares production. Security squares verification. $2 \to 4 \to 16$ is a square of a square.

3.3 The allocation theorem

The windfall W applies to all three activities. This is an assumption in favor of the optimist, and we defend it in Section 6. Capacity after the windfall is $3W$. Sustainable operation requires

$$c_P(q) + c_V(q) + c_S(q) = q + q^2 + q^4 \;\le\; 3W. \tag{1}$$

Theorem 1 (Fourth-root growth). *The largest sustainable volume multiplier $q^*(W)$ satisfying (1) is monotone in W and satisfies $q^*(W) = (3W)^{1/4}(1 - o(1))$ as $W \to \infty$.*

Proof. The left side of (1) is strictly increasing in q for $q > 0$, so q^* is well defined and monotone in W. For large q, q^4 dominates: $q^4 \le q + q^2 + q^4 \le q^4(1 + 2q^{-2})$, and the bounds squeeze q^* to $(3W)^{1/4}$ up to a factor $(1 + 2q^{-2})^{-1/4} \to 1$. □

Table 1 and Figure 1 give q^* for windfalls that are being claimed in practice. Two rows deserve attention. The row $W \approx 7.3$ is where $q^* = 2$ exactly, which yields the 2/4/16 allocation of the law's popular form. The row $W = 100$ is the marketing claim; it yields $q^* \approx 4.1$.

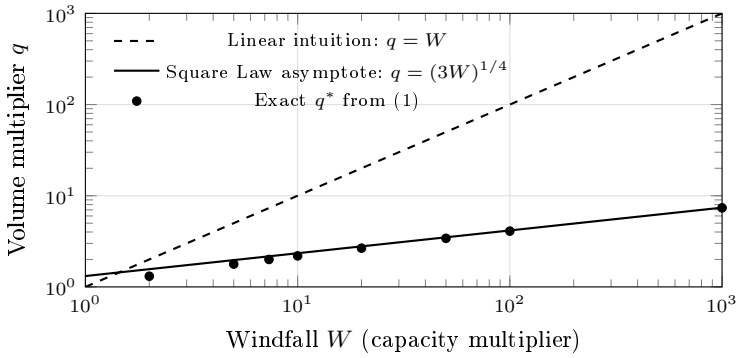

Fig. 1. The gap between the linear intuition and the Square Law is the unfunded assurance debt. On log axes the law is a line of slope 1/4.

3.4 Corollaries

Corollary 1 (Negative productivity). *If a team spends the windfall on volume alone ($q = W$), it accrues unfunded assurance debt of $(W^2 - W) + (W^4 - W)$ baseline units. At $W = 10$ the debt is 10,080 units against 27 units of added capacity.*

The debt is measured in the same currency as the windfall, and the debt is larger. A team that writes 100 features in a day and spends a month patching what they broke has negative productivity in every unit that matters. Boehm's escalation of defect-repair cost with phase [5] says the month is an underestimate.

Corollary 2 (The auditor's share). *At the sustainable point, the fraction of capacity spent on security is $q^{*4}/(3W)$, which tends to 1 as $W \to \infty$.*

In the limit, almost all of the windfall is security work. This sounds absurd until one recalls that the alternative is almost all of the windfall becoming attack surface.

Corollary 3 (Sensitivity to the baseline split). *Replacing the equal baseline split with $(0.6, 0.3, 0.1)$ for (P, V, S) raises $q^*(100)$ from 4.09 to 5.44. Replacing chain depth $k = 2$ with $k = 3$ (exponent 8) lowers $q^*(100)$ to 2.03.*

The constants move. The order of magnitude and the fourth-root shape survive every parameterization we tried. That robustness is the law's actual content.

3.5 What the law does not claim

The law does not say that quality assurance must literally consume four times the head-count of production. It says that assurance effort, in whatever currency, must scale with the square of volume, and that security effort must scale with the fourth power. Section 6 shows that the currency can be machine time.

The law does not say that all features interact. It says that the number of interactions one must *rule out* grows quadratically. Modularity, in Parnas's sense [30], reduces the constant. It does not reduce the exponent, because the adversary is under no obligation to respect the module boundary.

The law does not depend on AI. It has been true since the first two subroutines shared a variable. AI matters because it is the first tool that makes the linear term cheap enough for the superlinear terms to dominate within a single quarter.

4 Evidence for the Exponents

A law with unsupported exponents is a slogan. This section gives the support and, with equal care, the gaps.

4.1 Verification scales with interactions

Kuhn, Wallace, and Gallo examined field failures across medical devices, a browser, a web server, and a NASA distributed database [19]. In every dataset, failures were triggered by the interaction of at most six parameters. In the medical-device data, 97% were triggered by one or two. This finding founded combinatorial testing as a discipline [18] and it is the empirical core of Proposition 2: pairwise coverage catches most faults, and pairwise coverage costs $\Theta(N^2)$.

Lehman's second law states that as a program evolves, its complexity increases unless work is done to maintain or reduce it [21]. The GitClear data in Section 2 show refactoring collapsing from 21% to 3.8% of changed lines precisely while volume rose. The maintenance work Lehman's law requires is the work being skipped.

Basili and Perricone found, in a large Fortran system, that error density per line was highest in the *smallest* modules, and attributed the effect to interface faults [3]. Faults live at the seams. More features means more seams, and the seams multiply faster than the features. McCabe's cyclomatic measure [23] and Nagappan and Ball's churn measures [27] are both proxies for the same underlying quantity: the number of paths and edits that must be reconciled.

4.2 Security scales with compositions

Howard, Pincus, and Wing introduced the attack surface as a relative measure over entry points, channels, and untrusted data items [17]. Manadhata and Wing formalized it as a sum over resources weighted by damage potential and effort [22]. Both metrics are additive in resources, which gives the *linear* term of attack surface. The superlinear terms come from composition, and the evidence for composition is the modern exploit itself.

Log4Shell (CVE-2021-44228) [25] was a logging library, a string-substitution feature, a JNDI lookup, and an LDAP client, each individually defensible, composed into unauthenticated remote code execution across a very large fraction

of the world's Java deployments. The xz backdoor (CVE-2024-3094) [26] composed a build-system quirk, a test-fixture loader, a linker feature, and an SSH daemon that happened to load the compression library. In neither case would per-feature review have found the flaw. The flaw was the edge between edges. Browser exploit chains demonstrated at public competitions routinely compose several bugs; each additional link is another factor of q^2 in the candidate space an auditor must consider.

4.3 Where the evidence is weak

Honesty requires the counter-evidence. Shin and Williams found only weak correlation between per-file complexity metrics and vulnerability location in Mozilla JavaScript engine code [35]. Zimmermann, Nagappan, and Williams found that classical metrics predicted Windows Vista vulnerabilities with high precision and low recall [39]. Ozment and Schechter found that OpenBSD's foundational code became *safer* with age, with vulnerability discovery rates declining as the codebase matured [29].

We read these results as follows. Per-file metrics measure the linear term. Vulnerabilities live in the superlinear terms, which are cross-file by construction, so per-file metrics miss them. This is consistent with the law. The OpenBSD result says that a codebase whose volume is held roughly constant can pay down its security debt. It says nothing about a codebase whose volume is multiplied by four in a quarter, and it says nothing at all about the code shape GitClear documents.

What we lack, and what we ask the community to produce, is a direct measurement of vulnerability count as a function of feature count in AI-accelerated codebases with controlled review effort. Section 8 says how.

5 Productivity, Redefined

If management measures features shipped, the law will lose, because the law says to ship fewer. The metric must move from output to outcome.

5.1 Metrics that survive the law

Forsgren et al.'s SPACE framework already warns against single-dimension productivity measures [12]. We propose a minimal set derived from the three activities.

Verified-feature throughput. Features that have passed a defined interaction-coverage threshold per unit time. A feature with no pairwise tests against its neighbors counts as zero. This one change converts velocity from an output metric to an outcome metric.

Interaction coverage ratio. Tested feature pairs divided by total feature pairs. This is the direct observable for Proposition 2 and it should decline visibly when a team spends the windfall on volume.

Security debt ratio. Known-but-unremediated findings, weighted by severity, divided by verified features. The law predicts that this ratio grows as q^3 if security effort stays linear in volume.

Mean time to remediate (MTTR) for security findings, measured from discovery to deployed fix. Glasswing's partners report a median of two weeks for high-severity findings [2]; a team below that is ahead of the field.

Change failure rate and rollback rate, already standard in DORA, are the lagging indicators. When they move, the debt has already come due.

5.2 The courage to ship less

Hoare, in his Turing lecture, offered two ways to construct a design: make it so simple that there are obviously no deficiencies, or make it so complicated that there are no obvious deficiencies [16]. Every AI code generator in production today is optimized for the second. The generator is fluent, and fluency is the enemy of obviousness.

The psychological data are unkind. METR's developers believed they were 20% faster while being 19% slower [4]. Perry et al.'s participants believed their assisted code was more secure while it was less [33]. Stack Overflow's respondents distrust AI accuracy by a plurality and use it daily by a majority [37]. The pattern is that generated code *feels* like progress in a way that verification never will. Dijkstra observed in 1972 that the competent programmer is fully aware of the strictly limited size of his own skull [9]. The AI era has added an autocomplete to the skull without enlarging it.

Shipping less requires a manager who can say, in a planning meeting, that ten verified features are worth more than a hundred unverified ones, and a compensation scheme that agrees. Neither exists by default. The law is a tool for building both, because it converts the argument from taste to arithmetic.

6 Execution: AI Must Fund Its Own Audit

The obvious objection: if security effort must grow sixteenfold, who pays? Sixteen times the security head-count is unavailable at any price. The answer is that the windfall applies to all three activities, and that the evidence for AI as auditor is now stronger than the evidence for AI as author.

6.1 The auditor is better than the author

Google's Big Sleep agent found an exploitable stack buffer underflow in SQLite in October 2024, before the code reached a release [15]. Fang et al. showed a GPT-4 agent exploiting 87% of a set of one-day vulnerabilities given the CVE description [11]. DARPA's AI Cyber Challenge final in August 2025 had autonomous systems find 54 of 63 injected vulnerabilities and patch 43 across 54 million lines of code, discover 18 real zero-days along the way, at roughly $152 per task and 45 minutes per patch [8]. Anthropic's Project Glasswing reported,

in May 2026, more than ten thousand high- or critical-severity findings across partner systems within a month, with 90.6% of 1,752 assessed findings confirmed as true positives and 62.4% confirmed at high or critical severity [2].

Set these beside Veracode's 55% security pass rate for generated code, flat for two years [38]. The same class of model that cannot reliably write a safe log statement can find the unsafe one. The asymmetry has a simple explanation. Writing safe code requires satisfying every constraint at once. Finding unsafe code requires violating one. Search is easier than synthesis, and the auditor's job is search.

This asymmetry is what makes the law affordable. The q^4 term is paid in machine time, and machine time is the one input whose price is falling.

6.2 The A³ loop

We propose a workflow with three machine roles and one human role, organized so that the law's allocation is structural rather than aspirational.

Author. A model generates the feature against a specification. Budget: proportional to q.

Adversary. A *different* model, ideally from a different vendor and training lineage, receives the feature and the interaction graph of its neighbors and is instructed to break it: generate interaction tests, fuzz the boundaries, and construct exploit chains through adjacent features. Budget: proportional to q^2 for tests and q^4 for chains, allocated by (1).

Auditor. A third role, which may be a model or a static pipeline, verifies the adversary's findings against ground truth (does the exploit run, does the test fail on the unpatched code), rejects hallucinated findings, and produces the remediation. Glasswing's 9.4% false-positive rate [2] and the fabricated credentials in the GTG-1002 campaign [1] both say this role is mandatory.

Arbiter. The human. The arbiter does not read the code. The arbiter reads the adversary's report and the auditor's confirmations, decides what ships, and owns the residual risk. This is the only role that scales with the number of decisions rather than the number of features.

The gating rule is that a feature ships only when its interaction coverage ratio meets threshold and its confirmed-finding count is zero. Under this rule, spending the windfall on volume alone is impossible by construction, because the adversary's queue grows as q^4 and the arbiter's throughput is finite. The workflow makes the law self-enforcing.

The insistence on a different model for the adversary is a monoculture argument, developed in Section 7. A model auditing its own output shares its own blind spots.

6.3 Cost

At AIxCC's $152 per task, the security term for a system that grew from $N_0 = 100$ features to $N = 400$ ($q = 4$) requires assurance over $\Theta(N^4/4) \approx 6 \times 10^9$

chain candidates. That number is not a task count; it is a search space, and the AIxCC systems searched 54 million lines for $152 per task by pruning it. The correct reading of the cost figure is that the q^4 term is a budget for search effort, and that search effort in 2026 costs roughly what a junior engineer's hour cost in 2016. The law is expensive in the currency that is getting cheaper and cheap in the currency that is getting dearer.

7 The Macro Picture

7.1 Monoculture

Geer et al. argued in 2003 that a software monoculture converts independent risks into correlated ones, so that a single flaw becomes a systemic event [13]. The argument was about operating systems. It now applies to authors. If most new code is written by a handful of model families, an architectural habit in a model becomes an architectural habit in the world. Veracode's data show the pass rate for log injection at 13% across the model population [38]: the models share the blind spot. Spracklen et al. found that 5.2% of commercial-model and 21.7% of open-source-model code samples import packages that do not exist, producing 205,474 unique hallucinated names across 576,000 samples, a substantial fraction of them recurring across prompts [36]. A recurring hallucinated name is a supply-chain attack waiting for someone to register it. The attack is already named: slopsquatting.

The consequence for the law is that the q^4 term is correlated across organizations. A composition flaw in one generated codebase is likely present in ten thousand others. The adversary's search amortizes. The defender's does not, unless the defender shares findings. The Glasswing disclosure figures (530 high- or critical-severity open-source bugs identified, 75 patched with 65 public advisories at time of writing [2]) show the shape of that sharing and its lag.

7.2 Asymmetry

In November 2025 Anthropic disclosed a campaign, attributed to a state-sponsored group designated GTG-1002, in which an agentic model executed an estimated 80–90% of tactical intrusion operations against roughly thirty targets, with humans supplying 10–20% of the effort [1]. The model hallucinated credentials and overstated findings, which slowed the operator. It did not stop them. The report is the first documented case of the adversary receiving the windfall.

Lanchester's square law of 1916 states that the fighting strength of a force under aimed fire scales with the square of its numbers [20]. We borrow the name with intent. The adversary's search is aimed fire: every generated feature is a target, every composition a firing solution, and the search parallelizes perfectly. A defender who grows volume by W and security by a constant presents W^4 compositions to an adversary whose capacity also grew by W. The exchange ratio is W^3 against the defender. This is the arithmetic of the "cyber pandemic" scenario, and it needs no metaphor.

7.3 The flood

NIST reported in April 2026 that CVE submissions grew 263% between 2020 and 2025, that nearly 42,000 CVEs were enriched in 2025 (45% more than any prior year), that first-quarter 2026 submissions ran a third above the prior year, and that NVD would henceforth enrich only CVEs meeting risk-based criteria [28]. The vulnerability pipeline of record has begun triaging by necessity. Glasswing alone produced, in one month, findings equal in number to roughly a quarter of a year's enriched CVEs. Both of these facts are what the law predicts when volume outruns assurance on a global scale: the finders scale, the fixers do not, and the registry fills.

The Square Law is stated as an allocation rule. At the macro level it is a survival constraint. An industry that spends the windfall on volume will produce, in a few years, a global codebase whose composition space no defender has searched and every adversary can.

8 Threats to Validity

Construct. "Feature" is defined operationally and will be counted differently across organizations. The law is invariant to the counting unit as long as the unit is applied consistently, because q is a ratio.

The exponents. Proposition 2 rests on the interaction-fault literature, which is strong. Proposition 3 rests on a counting argument over exploit chains and on case evidence, which is suggestive. We have chosen $k = 2$ as a floor; real chains are longer, which makes the law conservative, and real chains are constrained by reachability, which makes it aggressive. The two effects are of unknown relative size. A direct measurement would fix, for a set of AI-accelerated repositories, the feature count and the review effort, and regress confirmed vulnerability count on feature count. We predict an exponent between 2 and 4 and we will consider the law falsified by a robust estimate below 1.5.

The equal-windfall assumption. We assume AI accelerates all three activities by the same W. The evidence in Section 6 suggests it accelerates search more than synthesis, which favors the law. If future models close the gap the other way, q^* falls further.

Recency. Much of the 2025–2026 evidence is industry-published (Veracode, GitClear, Anthropic, DARPA) and has not passed peer review. We have used only figures that the publishers state with method and sample size, and we have flagged each as industry data. The peer-reviewed studies [31,33,34,36,4] point the same direction.

Authorship. This paper argues that generated text should be audited by an adversary of different lineage. It was drafted with AI assistance and reviewed by the authors under that rule.

9 Conclusion

The windfall is real. It is also misread. The tool accelerates the one activity whose cost is linear in volume and leaves untouched the two whose costs are quadratic and quartic. A hundredfold windfall buys a fourfold product. The rest is assurance, and the assurance must be bought with the same tool, pointed the other way.

We have given the law a derivation, a theorem, a table, an evidence base with its weak points marked, a metric set, a workflow, and a falsification criterion. What remains is the measurement. We would like to be wrong about the exponent. We would like it more if someone checked.

Acknowledgments. The authors thank the maintainers whose codebases furnished the evidence, and the adversaries, human and otherwise, who furnished the motivation.

References

1. Anthropic: Disrupting the first reported AI-orchestrated cyber espionage campaign. https://www.anthropic.com/news/disrupting-AI-espionage (2025), november 13, 2025
2. Anthropic: Project Glasswing: An initial update. https://www.anthropic.com/research/glasswing-initial-update (2026), may 22, 2026
3. Basili, V.R., Perricone, B.T.: Software errors and complexity: An empirical investigation. Communications of the ACM **27**(1), 42–52 (1984)
4. Becker, J., Rush, N., Barnes, E., Rein, D.: Measuring the impact of early-2025 AI on experienced open-source developer productivity (2025), arXiv:2507.09089
5. Boehm, B.W.: Software Engineering Economics. Prentice-Hall (1981)
6. Brooks, F.P.: The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley (1975)
7. Brooks, F.P.: No silver bullet: Essence and accidents of software engineering. IEEE Computer **20**(4), 10–19 (1987)
8. DARPA: AI cyber challenge marks pivotal inflection point for cyber defense. https://www.darpa.mil/news/2025/aixcc-results (2025), august 8, 2025
9. Dijkstra, E.W.: The humble programmer. Communications of the ACM **15**(10), 859–866 (1972)
10. DORA: Accelerate state of DevOps report 2024. Tech. rep., Google Cloud (2024), https://dora.dev/research/2024/dora-report/
11. Fang, R., Bindu, R., Gupta, A., Kang, D.: LLM agents can autonomously exploit one-day vulnerabilities (2024), arXiv:2404.08144
12. Forsgren, N., Storey, M.A., Maddila, C., Zimmermann, T., Butler, B., Houck, J.: The SPACE of developer productivity: There's more to it than you think. Communications of the ACM **64**(6), 46–53 (2021)
13. Geer, D., Bace, R., Gutmann, P., Metzger, P., Pfleeger, C.P., Quarterman, J.S., Schneier, B.: CyberInsecurity: The cost of monopoly. how the dominance of Microsoft's products poses a risk to security. Tech. rep., Computer & Communications Industry Association (2003)

14. GitClear: The maintainability gap: 2026 AI code quality research. https://www.gitclear.com/the_ai_code_quality_maintainability_gap (2026), january 2026; 623M changed lines, 2023–2026
15. Google Project Zero: From Naptime to Big Sleep: Using large language models to catch vulnerabilities in real-world code. https://projectzero.google/2024/10/from-naptime-to-big-sleep.html (2024), november 2024
16. Hoare, C.A.R.: The emperor's old clothes. Communications of the ACM **24**(2), 75–83 (1981)
17. Howard, M., Pincus, J., Wing, J.M.: Measuring relative attack surfaces. In: Proc. Workshop on Advanced Developments in Software and Systems Security (2003), also CMU-CS-03-169
18. Kuhn, D.R., Kacker, R.N., Lei, Y.: Practical combinatorial testing. Tech. Rep. SP 800-142, National Institute of Standards and Technology (2010)
19. Kuhn, D.R., Wallace, D.R., Gallo, A.M.: Software fault interactions and implications for software testing. IEEE Transactions on Software Engineering **30**(6), 418–421 (2004)
20. Lanchester, F.W.: Aircraft in Warfare: The Dawn of the Fourth Arm. Constable, London (1916)
21. Lehman, M.M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE **68**(9), 1060–1076 (1980)
22. Manadhata, P.K., Wing, J.M.: An attack surface metric. IEEE Transactions on Software Engineering **37**(3), 371–386 (2011)
23. McCabe, T.J.: A complexity measure. IEEE Transactions on Software Engineering **SE-2**(4), 308–320 (1976)
24. METR: We are changing our developer productivity experiment design. https://metr.org/blog/2026-02-24-uplift-update/ (2026), february 24, 2026
25. MITRE: CVE-2021-44228: Apache Log4j2 JNDI features do not protect against attacker controlled LDAP and other JNDI related endpoints. https://www.cve.org/CVERecord?id=CVE-2021-44228 (2021)
26. MITRE: CVE-2024-3094: Malicious code in xz 5.6.0 and 5.6.1. https://www.cve.org/CVERecord?id=CVE-2024-3094 (2024)
27. Nagappan, N., Ball, T.: Use of relative code churn measures to predict system defect density. In: Proc. 27th International Conference on Software Engineering (ICSE). pp. 284–292 (2005)
28. NIST: NIST Updates NVD Operations to Address Record CVE Growth. https://www.nist.gov/news-events/news/2026/04/nist-updates-nvd-operations-address-record-cve-growth (2026), april 15, 2026
29. Ozment, A., Schechter, S.E.: Milk or wine: Does software security improve with age? In: Proc. 15th USENIX Security Symposium (2006)
30. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM **15**(12), 1053–1058 (1972)
31. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R.: Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. In: Proc. IEEE Symposium on Security and Privacy (S&P). pp. 754–768 (2022)
32. Peng, S., Kalliamvakou, E., Cihon, P., Demirer, M.: The impact of AI on developer productivity: Evidence from GitHub Copilot (2023), arXiv:2302.06590
33. Perry, N., Srivastava, M., Kumar, D., Boneh, D.: Do users write more insecure code with AI assistants? In: Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS) (2023)

34. Sandoval, G., Pearce, H., Nys, T., Karri, R., Garg, S., Dolan-Gavitt, B.: Lost at C: A user study on the security implications of large language model code assistants. In: Proc. 32nd USENIX Security Symposium (2023)
35. Shin, Y., Williams, L.: Is complexity really the enemy of software security? In: Proc. 4th ACM Workshop on Quality of Protection (QoP). pp. 47–50 (2008)
36. Spracklen, J., Wijewickrama, R., Sakib, A.H.M.N., Maiti, A., Viswanath, B., Jadliwala, M.: We have a package for you! A comprehensive analysis of package hallucinations by code generating LLMs. In: Proc. 34th USENIX Security Symposium (2025)
37. Stack Overflow: 2025 developer survey: AI. https://survey.stackoverflow.co/2025/ai (2025)
38. Veracode: Spring 2026 GenAI code security update: Despite claims, AI models are still failing security. https://www.veracode.com/blog/spring-2026-genai-code-security/ (2026), march 24, 2026
39. Zimmermann, T., Nagappan, N., Williams, L.: Searching for a needle in a haystack: Predicting security vulnerabilities for Windows Vista. In: Proc. 3rd International Conference on Software Testing, Verification and Validation (ICST). pp. 421–428 (2010)