Software Degradation and the Loss of Taste

Flyxion
Independent Researcher

November 2026

Abstract

This essay argues that the dominant course of generative AI over the last several years, in which models are used to produce as much code, text, imagery, and interface as possible at the lowest marginal cost, has been a wrong turn for software and for the culture that surrounds it. The argument has two parts. The first concerns software. Code is a liability that must be read, secured, and maintained, so a technology that makes code nearly free to write while leaving it as costly as before to review and keep alive transfers cost from the author to everyone downstream. Measurements of duplication, churn, refactoring, error handling, delivery stability, and security, together with a controlled trial in which experienced developers misjudged the effect of the tools on their own speed, are consistent with this transfer, though each has limits that are stated. The second part concerns taste, understood as the trained capacity to discriminate and to refuse. Taste is formed and exercised by the friction of choosing, and a medium in which the median output is supplied for free and shipped by default weakens both the supply of taste and the demand for it, a weakening visible in the template-made event poster, the illustrated mathematics group, and the uniform filter. The cost transfer is formalized in a model of generation, verification, and admission, and taste is coupled to the evaluative component of verification capacity, so that software degradation and the loss of taste may be a single process seen from two sides. The essay considers the strongest replies, namely that throughput has risen, that bad software and bad media long predate the technology, that the models are improving, that production has been democratized, and that the backlash against slop is itself often undiscriminating. It concludes with an account of an alternative course in which models are used to reduce, verify, and delete, and with the observations that would count against the thesis.

Keywords: generative AI, software quality, technical debt, taste, AI slop, event posters, software maintenance

1 The Thesis and Its Scope

The claim defended here is narrow in one respect and broad in another. It is narrow because it concerns a course, a pattern of deployment and incentive, and not the capability of language models or the legitimacy of any single use of them. A model that finds a defect in a parser, drafts a unit test that a competent person then reads, or helps a beginner understand a compiler error is doing something useful and is not the object of complaint. It is broad because the course in question is the dominant one. It consists of treating the production of artifacts as the goal, measuring success by volume and by the share of work done by the machine, embedding generation into every product by default, and leaving review, maintenance, and judgment to whoever happens to come after. The argument is that this course has degraded software at very large scale and has weakened the cultural capacity to tell good work from acceptable work, that the two effects have a common cause, and that a different course was and remains available.

The common cause is the separation of generation from judgment. For nearly all of the history of making things, the effort of producing an artifact and the effort of judging it were bound together in the same person, who could not produce without repeatedly deciding, and whose decisions accumulated into a sensibility. Generative systems cut that tie. They produce candidate artifacts at a cost that is negligible beside the cost of deciding whether any of them is good, and they do so in a form that is fluent enough to pass a quick inspection. When the cost of candidates collapses and the cost of judgment does not, the system as a whole receives more candidates than it can judge, and the unjudged remainder is shipped. Everything that follows in this essay is an elaboration of that sentence: first for software, where the consequences can be measured, and then for taste, where they can be argued but not measured with the same confidence.

One dependency should be stated at the outset, because the second half of the argument turns on it. The claim about taste rests on a hypothesis that has not been measured: that discrimination trained by selecting among supplied options is thinner than discrimination trained by making. If that hypothesis is false, and selection trains discrimination as well as construction does, then the claim about the loss of taste reduces to the economics of the median, which is real but weaker. The software half of the argument does not depend on the hypothesis, and the reader may accept it while reserving judgment on the rest.

A note on method belongs here. The empirical claims below rest on a mixture of controlled trials, large observational analyses of code repositories, industry surveys, and vendor reports. The evidence differs in quality and in independence, and each item is described with its limits. Where a number comes from a vendor with an interest in the answer, or from an analysis of commits that cannot distinguish cause from correlation, the text says so. The argument does not depend on any single figure, and it would not be rescued or refuted by the revision of one. What it depends on is a structural claim about costs, which the evidence is consistent with and does not by itself establish. The four kinds of evidence support different kinds of inference and should not be pooled. A randomized trial supports a narrow causal conclusion about the population and tasks studied. Observational analyses of repositories support claims about association and about persistence over time, and cannot separate the effect of the tools from other changes in how code is written. Annual surveys such as DORA support claims about correlation in self-reported data. Vendor benchmarks, such as Veracode's, estimate failure rates on designed tasks, which are not estimates of the

prevalence of vulnerabilities in deployed software. Several of the sources are also secondary reports of primary studies, and the bibliography marks which these are. The sources are also not fully independent. The GitClear findings reported in different years come from one research group, the several references to DORA describe one underlying survey series, and the Veracode reports are successive measurements from one vendor's benchmark program, so repeated reporting of a study is not independent confirmation of it. Where the evidence is mixed, as with delivery throughput, the text says so.

2 What Software Quality Is, and Why Abundance Is Not Its Measure

A program is not an asset in the way a finished physical object is. It runs in an environment that changes, depends on components maintained by others, and must be altered whenever its users, its platform, or its threats change. Lehman's laws of software evolution, formulated from observation of large systems in the 1970s, state the consequence in two sentences that remain accurate: a program in use must be continually adapted or it becomes progressively less satisfactory, and as it is adapted its complexity grows unless work is done specifically to reduce it [14]. The corollary is that the quantity of code is a cost and not a product. Each line carries a continuing obligation to be understood by someone, tested against changing conditions, and defended against attack. A system that achieves a function in a small number of lines that are well understood is in every respect that matters better than one that achieves it in a large number that nobody holds in mind, even if the two behave identically on the day of delivery.

Brooks drew the relevant distinction between the essential and accidental difficulties of software [5]. Accidental difficulties are those that arise from the tools and notations used to express a design, and a great many advances in programming have reduced them. The essential difficulty is the design itself: deciding what the thing should be, which distinctions it must honor, how its parts relate, and what it should refuse to do. Brooks argued that no single development would deliver an order of magnitude of improvement because the essential difficulty is conceptual and cannot be tooled away. He returned to the argument in 1995 and defended the distinction while reconsidering particular claims about how far certain techniques could reduce the difficulty [6], and the distinction is used here in that qualified form. Generative models attack the accidental difficulty of typing out a design with extraordinary force, and for that reason the temptation is to conclude that they attack the problem of software. They attack it only in part. Models can contribute to conceptual work, for instance in requirements analysis, in exploring the space of possible designs, and in pointing out cases a design has missed, and these are real aids. What they cannot do is discharge responsibility for the correctness of a design, where responsibility means accountability, the standing to answer for a design and to be corrected for its failure, and not causal contribution to it, which models plainly make, and they make the expression of an under-specified intention cheap, so that an under-specified intention expressed cheaply is a design problem that has been postponed and multiplied. The conceptual work of deciding what the program should be, and of answering for it, remains where it was, and when the typing is no longer a bottleneck the absence of that work becomes the dominant source of failure. The danger is accordingly not that effort is eliminated, which is often a gain,

but that the occasions for making decisions can be eliminated while the artifact keeps the appearance of having been decided.

It follows that volume is the wrong measure of software progress, and that any technology which raises volume faster than it raises understanding will lower quality unless something else intervenes. The thing to ask of a development in software tooling is whether it reduces the amount of code that must be held in mind to achieve a given function, and whether it makes the remaining code easier to verify. A tool that does the first and second is an advance. A tool that increases the amount of code while leaving verification unchanged is a loan, and the question is who repays it.

3 The Evidence

3.1 Duplication, churn, and the disappearance of reuse

The most direct measurements of what happens to codebases when generation becomes cheap come from large analyses of code changes. A 2025 analysis by GitClear of 211 million changed lines across private repositories and twenty-five large open source projects found, for 2024, an eight-fold increase in the frequency of duplicated blocks of five or more lines relative to two years earlier, and described a collapse in the movement of code, which is the signature of reuse and refactoring [15]. A later joint analysis by GitClear and GitKraken of 623 million code changes from 2023 to 2026 reported that AI-assisted commits now account for about a quarter of all commits, that duplication had risen by 81 percent while code reuse had fallen by 70 percent, and that legacy refactoring had fallen by 74 percent since 2023 [16]. The same report found that code which catches errors without evaluating their cause, a pattern it calls obfuscation, was up 47 percent, and that functional connectivity, a measure of how often new code calls existing functions, had fallen by 35 percent [16].

The interpretation of these figures needs care. They are derived from commit data and from the analytic definitions of a single research group that sells tools in this area, they are reported here through a trade publication and not from the primary report, and they cannot by themselves separate the effect of AI assistance from other changes in how code is written over the same period. A claim that AI assistance caused each of these movements would outrun the evidence. What the figures establish is the shape of the change: code is being added more than it is being reorganized, copied more than it is shared, and written to avoid failing visibly more than to be correct. That shape is the one predicted by the cost asymmetry of the previous section. Adding a duplicate block costs nothing to the one who requests it and the model that supplies it, while the work of finding the existing function, understanding its contract, and calling it, which is what reuse requires, costs attention that the generation process does not spend.

An independent academic study points in the same direction from a different method. Liu and colleagues traced technical-debt issues introduced by AI coding assistants in real repositories from the introducing commit to the latest revision, to ask whether such issues are quickly fixed or persist and accumulate, and concluded that AI-generated code can introduce long-term maintenance costs into real software projects [18]. Developers describing their own experience reach similar conclusions. A 2026 qualitative study of how developers discuss “AI slop” in software found that generation lowers the effort of making a contribution

while shifting the burden of quality assurance onto maintainers, and recorded a developer's observation that one can go very fast with these tools while accruing technical debt at a much higher speed [25]. The same study quotes a self-described supporter of the technology who nonetheless feared that deployment at scale would degrade software quality across the board [25].

3.2 Stability and security

The annual survey research of the DORA group at Google measures software delivery in two dimensions, throughput and stability. Its 2024 report, drawing on about three thousand respondents, found a 1.5 percent decrease in delivery throughput and a 7.2 percent decrease in delivery stability for every 25 percent increase in an organization's adoption of AI [31]. The 2025 report, based on nearly five thousand professionals, reported that adoption had become nearly universal and that the throughput penalty had disappeared, so that AI adoption now improves delivery throughput, but that it still increases delivery instability [9, 12]. Its authors wrote that adoption not only fails to fix instability but is currently associated with more of it [24]. This is the finding most favorable to the defenders of the technology and the least favorable to the course criticized here at once: the tools help teams ship more, and teams that ship more with them also break more.

Security shows the pattern with unusual clarity because it can be tested directly. Veracode's 2025 report gave more than one hundred language models eighty coding tasks designed around known weakness classes, offered each a choice of a secure or insecure way to complete the code, and found that the models introduced vulnerabilities from the OWASP top ten in 45 percent of cases [32]. Newer and larger models were no better [29], and security performance had remained unchanged over time even as the models grew better at producing syntactically correct code [32]. Veracode's 2026 report put the rate at 44 percent, noting that it had barely moved since the research began and that what had changed was the scale at which such code now enters production [33]. These are failure rates on designed benchmark tasks and not measurements of how many vulnerabilities reside in deployed software, since deployed code is also reviewed, scanned, and constrained in ways the benchmark is not. Because the vendor sells security products, the numbers should be read as indicative; the stability of the result across successive years and across model sizes is nonetheless the more informative feature, since it suggests that the failure is not a matter of capability that scale alone repairs.

3.3 The perception gap

The most unsettling result for any argument conducted by appeal to experience is the controlled trial by METR. Sixteen experienced open source developers completed 246 real tasks in mature repositories that they had worked on for years, with each task randomly assigned to allow or forbid AI tools. Before the work they forecast that the tools would reduce completion time by 24 percent. The tasks in which AI was allowed took 19 percent longer [4]. After the study the developers still estimated that the tools had made them 20 percent faster [20]. The study concerned early-2025 tools, a small number of participants, and a specific population of expert maintainers working on code they knew intimately, and it should not be generalized to other settings or to later tools without further evidence. These

limits are real and should not be waved away. The lesson that survives them is epistemic and not a matter of speed. The people closest to the work, and with every incentive to judge accurately, could not tell from the inside whether the tool was helping. A technology whose users cannot perceive its effect on their own output cannot be steered by the ordinary feedback of satisfaction, and a market built on that satisfaction will reward what feels productive over what is.

3.4 What the evidence does not show

A fair summary has to include what is absent. The evidence does not show that every use of generation degrades software. It does not show that careful teams who review everything and delete aggressively are worse off, and the DORA reports themselves indicate that outcomes depend heavily on the organization that adopts the tools and on its existing practices. It does not show that the trends are permanent, since the models and the habits around them are changing quickly. It does not show the counterfactual, namely what the same teams would have produced without the tools, apart from the METR trial. What it shows is that, in aggregate and on the properties that determine the cost of owning software, the measured movement has gone in the direction the structural argument predicts, in several datasets and research programs that use different methods and have partially overlapping sources, and that the one trial built to test the felt benefit found that the felt benefit was not reliable.

4 Why It Happens

Five mechanisms account for the pattern, and they reinforce one another.

The first is the cost asymmetry already described. The marginal cost of generating a further candidate is close to zero and is borne by the generator, while the cost of reviewing it is borne by a reader whose time was not consulted. The shift is visible in the way the practice of open source maintenance has been described: the effort needed to make a contribution has fallen to almost nothing while the effort of judging whether it belongs has not, and maintainers absorb the difference [25]. This is an old pattern in the economics of spam and of unsolicited demands generally, and the novelty is that it now operates inside the process of building the thing, among colleagues.

The second is the objective that the systems are trained and prompted to meet, which is to satisfy the request in front of them. A request such as a function that does X is satisfied by a function that appears to do X on the inputs the requester will try. Satisfying it properly requires inferring which other inputs may arrive and deciding what to do with them, which takes effort and tokens, whereas writing code that never raises an error under any input is a cheaper way to appear to succeed [16]. The rise in code that swallows errors is what one would expect from systems rewarded for the appearance of completion.

The third is locality. A model sees a window of context and not the system, and it cannot easily know that a function with the needed behavior exists three directories away or that the convention it is departing from was adopted after an outage. Reuse is the practice of knowing what already exists and honoring its contract, and it depends on a kind of global familiarity that the process of prompting and accepting suggestions does not build in the

person either. When both the author and the tool lack the global view, duplication is the default outcome, and it is the outcome the measurements show.

The fourth is the corruption of feedback. If users cannot reliably perceive the effect of the tools on their own productivity [20], then the signal on which organizations and individuals would otherwise adjust is unreliable, and enthusiasm will persist where outcomes are poor. When combined with metrics that count the share of code written by the machine as an achievement in itself, as when a trade report on a security study states, without a primary source consulted here, that as much as a third of new code at some large companies may now be machine-generated [29], the result is an incentive structure in which the amount of generated code becomes a target. It is an old observation, associated with Goodhart and generalized by Strathern, that a measure which becomes a target ceases to be a good measure [27]. The proportion of code that is machine-written is a measure of a process and says nothing about whether the process is good, and its adoption as a target has all the properties that the observation predicts.

The fifth is deployment by default. The tools are not only available but integrated, switched on, and promoted, and the people who would decline them have a diminishing ability to do so inside their own editors, operating systems, and applications. The backlash described in the cultural press is directed in large part at the forced nature of the implementation and the inescapability of its output, and not simply at the quality of any single output [23]. A technology that must be accepted because it cannot be avoided is not subject to the discipline of voluntary adoption, which is the only discipline that would reward quality.

A Minimal Model of Generation, Verification, and Admission

The economic core of the argument can be stated in a small model, offered as an idealization and not as an estimate. Let G be the rate at which candidate artifacts are generated, V the verification capacity, that is, the rate at which artifacts can be effectively verified when work is available, and B the backlog of unverified artifacts. Let $A(B)$ be the rate at which unverified artifacts are admitted into production, with $A(0) = 0$ and A nondecreasing in B, so that the pressure to ship grows as the backlog grows. Then

$$\frac{dB}{dt} = G - V - A(B).$$

The equation describes the overloaded regime $G > V$, in which the backlog is positive and verification is saturated. When $G \leq V$ actual verification is limited by the available work, the backlog stays at zero, and no unverified artifact is admitted, so the regime of interest is the first. Generation in excess of verification is not by itself harmful. If admission is strictly refused, $A \equiv 0$, and the backlog grows without bound but nothing unverified enters production; the cost is wasted generation and a growing queue, and it is borne by the producer. The harmful regime requires both an excess of generation over verification and a positive rate of admission of unverified work. The following statement makes the second condition precise.

Proposition 1. *Let $G > V$ be constant, and let $A : [0, \infty) \to [0, \infty)$ be continuous and nondecreasing with $A(0) = 0$ and $\bar{a} = \sup_B A(B)$. Then the following hold. (i) The equilibria of the backlog are exactly the points B with $A(B) = G - V$, so at every equilibrium the rate at which unverified artifacts enter production equals $G - V$; the equilibrium set is a*

closed interval, which is a single point whenever the level set $\{B : A(B) = G - V\}$ *contains exactly one point. (ii) If* $\bar{a} > G - V$*, the equilibrium set is a nonempty compact interval* $[B_-, B_+]$*, and every trajectory converges to the nearest point of it. (iii) If* $\bar{a} < G - V$*, then* $\dot{B} \geq G - V - \bar{a} > 0$ *and the backlog diverges at least linearly; in particular, if* $A \equiv 0$ *the backlog grows at rate* $G - V$ *and no unverified artifact is admitted. (iv) If* $\bar{a} = G - V$*, an equilibrium exists if and only if* A *attains the value* $\bar{a}$ *at a finite* B*; otherwise the backlog diverges sublinearly, in the sense that* $B(t)/t \to 0$*.*

Proof. Write $f(B) = G - V - A(B)$, which is continuous and nonincreasing with $f(0) = G - V > 0$. Equilibria are zeros of f, that is, points with $A(B) = G - V$, and since A is continuous and monotone the preimage of a point is a closed interval, possibly empty or a single point. This gives (i). If $\bar{a} > G - V$, the intermediate value theorem gives a point with $A(B) = G - V$, the set of such points is bounded because A exceeds $G - V$ beyond some finite B, and for B below the interval $f > 0$ while for B above it $f < 0$, so the one-dimensional flow moves toward the interval, which gives (ii). If $\bar{a} < G - V$ then $f \geq G - V - \bar{a} > 0$ everywhere, which gives (iii). If $\bar{a} = G - V$ and A attains $\bar{a}$ at a finite point, that point is an equilibrium and the argument of (ii) applies; if A never attains it then $f > 0$ everywhere, so B is increasing, and a bounded increasing solution would converge to a finite limit L with $f(L) > 0$, a contradiction, so B diverges, and $B(t)/t \to 0$ because $\dot{B} = f(B) \to 0$, which gives (iv). □

The result separates two things that discussions of AI-assisted work tend to run together. The form of the admission rule determines how large the backlog is, and it does not determine how much unverified work reaches production in steady state, which at any equilibrium equals the excess $G - V$. An organization that wishes to avoid leakage has only three levers: raise V, lower G, or hold A at zero and tolerate the queue. If generation is throttled by the backlog, so that G falls as B rises and reaches V at some B_c, the system settles at B_c with $A \equiv 0$ and no leakage. In practice the third lever is rarely held, because the backlog is visible, the pressure to ship is constant, and the artifact is already in hand. If each unverified artifact carries a defect probability q, the rate at which defects enter production is qA, and the stock of defects in production grows whenever this exceeds the rate at which defects are found and repaired, which is the accumulation of debt that the repository studies describe. The defect probability q is not constant, and the composition of the backlog matters: artifacts that are easy to verify leave the backlog first, so the artifacts that remain, and that are eventually admitted under pressure, are enriched for those that are hard to check and plausibly more likely to be defective. The effective q of admitted work may therefore rise with the size of the backlog, which makes the debt-accumulation condition easier to meet than a constant q would suggest.

The model is intentionally minimal. The assumption that A is nondecreasing describes one institutional response, in which the pressure to ship rises with the backlog, and is not an empirical law; in some organizations a growing backlog triggers a release freeze, so that admission falls as the backlog grows. It treats G and V as given and does not estimate them, and none of the data cited above measure its parameters; the repository and survey studies are consistent with a regime of excess generation and positive admission, and do not demonstrate one. What the model contributes is the statement of the conditions under which the thesis would hold: generative abundance becomes destructive when an institution

treats the availability of an artifact as sufficient reason for its admission.

5 Taste and Its Erosion

5.1 What taste is

Taste is easily mistaken for preference or for ornament, and neither is the sense in which it matters here. Hume's essay on the standard of taste argued that although sentiment is the basis of judgment, judgments are not equally good, and that the best are those of people who have delicacy of perception, practice, comparison, and freedom from prejudice, and who have been exposed to a great many of the objects in question [11]. On this account taste is a trained capacity and a kind of compressed record of consequences. A person with taste in typography, in interfaces, in the design of an API, or in prose has seen many failures and successes, has formed discriminations that are hard to put into words, and can refuse, which is the essential act. Taste is exercised most clearly in what is left out and in what is rejected, and it is built by the repeated cost of choosing and of living with the choice.

In software, taste shows as the preference for the smaller design, the clearer name, the interface that does one thing and exposes few concepts, the dependency not taken, the feature declined. It is the faculty that makes the essential difficulty of Brooks tractable, since the conceptual integrity he thought the most important property of a system is a matter of consistent judgment applied across a large number of decisions [5]. Where taste is absent, systems accrete: each addition is individually defensible, and the whole has no shape.

5.2 How generation weakens it

Generative systems weaken taste through both of the channels by which it is sustained, which are its formation and its demand.

The formation of taste requires friction of a particular kind, and the distinction between kinds matters to the argument. Some friction is accidental: it consumes effort without improving discrimination. A typographer does not lose judgment when software removes the labor of manual typesetting, and a programmer does not lose architectural understanding when a compiler automates repetitive transformations. Other friction is productive: it exposes a consequential choice and its outcome to the person who made it. Whoever writes the function must decide what to call it and notices when the decision is hard, which is the beginning of recognizing a confused concept. Whoever draws the layout feels the awkwardness of a misaligned element. Whoever composes a sentence experiences the rejection of the first formulation as a loss and learns from it. The objection to delegating the first draft is therefore not that it removes effort, for much effort deserves removal, but that it can remove the occasions on which such decisions are made while preserving the appearance that they have been made. A plausible draft arrives complete, and the choices that a person would have faced in producing it are resolved in advance by a system whose resolutions the person did not weigh. The relation between recognition and production needs care as well. Recognition is not a weaker faculty than production, and expert reviewers, editors, critics, and auditors can discriminate far better than many who produce. The more defensible claim is that the two train partly different competencies, and that substituting selection for

construction may deprive learners of forms of feedback that only construction provides, such as discovering through failed attempts why a design does not hold. Whether a population that mostly selects from supplied options develops thinner discrimination than one that mostly makes is the central unmeasured claim of this essay. A skilled person reviewing a draft still faces each decision, whether this is the right name or the right emphasis, and faces it as a recognition task and not a generation task, and nobody currently knows whether the discrimination so trained is thinner. The essay advances the thinner-discrimination view as a hypothesis motivated by the METR result, in which people closest to the work misjudged its effect on themselves [20], and it states plainly that the argument about taste is weaker to the extent that the hypothesis fails.

The demand for taste weakens through the economics of the median. Generative systems are capable of unusual output, since sampling, prompting, fine-tuning, and selection can all produce it, so the mechanism is not that they can only produce what is typical. It is that shared model priors, common templates, and platform incentives bias the outputs that are actually selected and distributed toward a common center, and a mass of such outputs sets a new baseline against which the acceptable is measured. There is some direct evidence of the convergence involved. A January 2026 study reported that when an image generator was set to train repeatedly on its own outputs it drifted toward very generic images, which the researchers called visual elevator music [23]. That is a laboratory-scale model of what happens at cultural scale when generated material becomes the training material and the reference material of the next round. As the term slop entered the dictionary, with the Cambridge Dictionary adding a definition for low-quality internet content produced especially by artificial intelligence in June 2025 [26], and as usage of the word rose about ninefold during 2025 [19], people's recognition of the genre grew. Recognition is not the same as a restored supply of discriminating makers, and the existence of widespread disgust at the median is compatible with the median continuing to be what most products ship.

The interface of software is the place where this meets the argument of the previous section. Interfaces built from generated components and generated copy converge on the same shapes, and features are added by default because adding costs nothing. The user is offered an assistant in every surface, a summary of every page, and a suggestion at every cursor, regardless of whether anyone with taste decided that the surface needed one. The decision not to include a feature is the one that the process makes unlikely, since no one is rewarded for the absence and the absence cannot be shown in a demonstration.

5.3 Taste and the standards of judgment

A reasonable worry is that talk of taste is an elitist rationalization for the preferences of those who already hold the standards of the previous era. The answer lies in what taste does functionally. In software it is not an aesthetic luxury, since the properties it selects for, such as small surface area, reuse, and clarity, are the ones that make systems cheaper to secure and to change. In media, the discrimination of the competent maker from the fluent imitator protects the audience from being served the average of everything at once. Taste has a public function in both, and a society that lets it atrophy has to find the same discriminations elsewhere, usually in metrics of engagement, which select for something different.

5.4 Taste as Evaluative Verification Capacity

The model of the previous section and the argument of this one can be joined. Verification is not a single mechanical operation. Some of it is formal: checking that code satisfies a specification, that a test passes, that a type is respected, that a dependency has no known vulnerability. Tests, static analysis, and automated tools can raise this formal capacity, and models can help build such tools. Some of it is evaluative: deciding whether an interface is comprehensible, whether an abstraction belongs in a codebase, whether an explanation misleads, whether an illustration contributes anything to a post. This kind of verification depends on context and on trained discrimination, which is what has been called taste. On this view taste is a component of verification capacity, and the formal and evaluative parts are better understood as complementary stages than as substitutes. A program may pass every automated test and still require architectural approval, and a sound design may still fail security testing. If every artifact must pass both stages, a serial pipeline with continuously available work, no rework, and stable capacities has steady-state throughput $V = \min(V_f, V_e)$. If the two were instead substitutable and measured in common units, then $V = V_f + V_e$, and this additive case is the best case for the defenders of the present course, since it allows formal improvements to compensate fully for evaluative decline, and it is kept as a limiting comparison.

The coupling has a consequence that the model makes exact, and the form of the consequence depends on which formulation holds. Since the leakage of unverified work at equilibrium is $G-V$, a fall in evaluative capacity raises leakage one for one under the additive formulation, where it could in principle be offset by more automated checking. Under the complementary formulation leakage is $G - \min(V_f, V_e)$, and once evaluative judgment is the bottleneck further improvements in formal verification leave leakage unchanged. The second formulation seems the more realistic and gives a stronger version of the thesis: automation can greatly increase the capacity to check what has been specified while leaving the capacity to judge what ought to have been specified as the limiting factor, so that a system can become better at producing and mechanically checking artifacts while becoming less able to decide which of them deserve to exist. The composition effect described in the model section is the artifact-level version of this bottleneck: the artifacts that formal checks cannot settle are those that remain in the backlog, and the capacity that settles them is evaluative. The cultural hypothesis then takes a precise form: widespread delegation of consequential decisions may reduce the practice through which evaluative capacity is developed, even while automated tools raise formal capacity. A feedback mechanism follows. Abundant generation reduces occasions for practiced judgment, which lowers evaluative capacity, which lowers V, which raises the admission pressure that sustains the backlog, which permits still more generation to be treated as acceptable. This is the cultural analogue of the corruption of feedback described among the mechanisms above, in which poor perception sustains poor practice, and it has the same potentially self-reinforcing structure at the level of a population and not an organization. Whether it is in fact self-reinforcing depends on the gain of the loop relative to stabilizing effects. Increased generation also supplies more occasions for reviewing, rejecting, and correcting artifacts, which could improve evaluative skill, and institutions may respond to admission failures by tightening review. The proposed coupling therefore permits a self-reinforcing degradation regime under certain institutional and learning conditions, and it is not claimed that such a regime necessarily emerges. This

sequence is not an established causal chain, and its second step is the unmeasured claim named at the outset. It is a testable hypothesis, and its value is to connect the two halves of the essay: software degradation and the loss of taste are not only parallel consequences of one development but potentially coupled processes, in which declining judgment makes further degradation harder to detect and resist.

6 The Visual Vernacular

The material of this section belongs to a different evidentiary register from the software sections. It rests on reporting, on the practices of venues, and on observation, and it is offered as the phenomenology of the thesis and not as its proof. The erosion of taste is not only an abstract matter of discrimination. It has a visible form, and the form is the gaudy, over-saturated, memeified look that now covers a large share of what ordinary people encounter every day: the filtered photograph, the event poster, the illustrated post. Three places show it with particular clarity.

6.1 Event posters

The event poster is a small and old genre with a clear job. It must tell a passer-by what is happening, when, and where, and it must signal by its look what kind of event it is. A poster for a folk night, a punk show, a farmers' market, and a lecture should not resemble one another, and the differences are the point. They are produced by someone deciding what this event is like and drawing, setting type, or cutting paper accordingly. The hand-made poster is also a potentially informative signal of local participation, in that someone near the event cared enough to make it, although the effort of production is not always observable and a signal of this kind is reliable only under additional incentive conditions.

Generated posters dissolve this function. A writer for TechRadar observed that such posters are easy to spot because they all look similar and generic, and drew the consequence that essentially all events end up with the same branding, so that a very good event and a poor one become visually indistinguishable [30]. The reaction has been strong enough to show up in the practices of small venues. Axios reported in 2026 that breweries, markets, and landscapers in the Tampa Bay area were being called out in comment sections for posters made with generative AI [2], and that Utah businesses had banned such flyers from their platforms and pledged to advertise without them, with one saloon telling bands to find a local artist or cut out magazines and adding that its stance was to do it yourself or die [3]. A Halifax music venue announced that it would no longer accept AI event posters, in a pub whose walls are covered with posters by local designers [10]. An Oakland venue banned AI flyers and reported that its announcement drew nearly eight thousand likes, with commenters saying that generated concert posters made them less likely to attend [7]. The 404 Media report that gave the phenomenon the name of a flyer pandemic summarized the complaint as the same one made against all other slop, that the work looks generic and lazy and as though the business does not care [1].

Fairness requires recording the best defense. When an Oakland arts festival was criticized for an AI flyer, its organizers answered that they were a nonprofit running a free event with no budget to hire artists, and that they had edited the output considerably [22]. That is a

real constraint, and it explains why a technology that offers a finished image for nothing is attractive to those who have nothing. It does not change the aesthetic fact, and it points to the alternatives that the venues themselves proposed, which are cheaper than they sound: a stick figure drawn by hand with a line saying that the drawer will attend the event was shared approvingly in preference to a polished generated image [8]. A crude human flyer carries information that the polished one does not, namely that a person made it, and the audience has begun to read the polish itself as a warning.

6.2 The mathematical groups

A second place where the vernacular is visible is the online group devoted to sharing mathematics. Groups that circulate equations, identities, and proofs once had a recognizable look, which was plain: a typeset formula, a photograph of a page of working, a diagram. Increasingly the posts are wrapped in generated illustrations in which the equation floats over a glowing cosmos, a brain made of circuitry, golden gears, or a figure in a laboratory coat, in colors saturated past the point of legibility, and with a caption in the confident cadence of engagement bait. This is offered as an observation of how such groups look and not as a counted result, and no systematic survey of mathematical groups is cited here. It is nonetheless continuous with documented behavior on the same platforms. Journalists have recorded generated images appearing in Facebook groups dedicated to science and claiming to show new discoveries [21], and researchers at Stanford and Georgetown tracked 120 Facebook pages that frequently posted generated images and found that they collectively received hundreds of millions of engagements and exposures, with the feed ranking algorithm apparently promoting the content to people who did not follow the pages [13].

Mathematics is the limiting case because it is the subject in which ornament is least excusable. The content of a mathematical post is a statement that is true or false, and the entire value of the post lies in the correctness and clarity of that statement. An illustration that adds a glowing nebula to the Euler identity adds nothing to the identity and signals that the post was made for the reaction and not for the result. Worse, generated images of equations are routinely wrong in their details, so the ornament can contradict the content it decorates, and the audience is trained to look past the formula to the spectacle. A group built to share mathematics and filled with such images has kept the form of a community of understanding and lost its function.

The economics is not mysterious. Reporting on the Facebook case found that the platform’s operator pays creators for viral images and that generators are the easiest way to collect, with courses on how to do so circulating on other platforms [28]. When engagement is rewarded and the cost of the image is zero, the gaudier image wins, because gaudiness is what an engagement metric has learned to select. The equation is incidental, and the group is a distribution channel.

6.3 The filter and the template

The third place is the filter, the effect, and the template: the one-tap transformation that makes a photograph look as though it has been painted, glossed, or lit by a film set, and the one-click template that supplies a layout and an image for a poster. These are the same technology at the smallest scale, and they share its character. They apply a median look to

whatever is put into them, and they do so uniformly, so that the faces, rooms, and meals of very different people come out in the same palette and the same grain. It was no accident that the most widely noticed early backlash concerned a filter-like effect, the March 2025 reaction to images imitating a well-known animation studio's style [19]. Where templates for event posters are built into a general image assistant, as one writer suspects a recent release has done, the sameness is delivered as a feature [30].

6.4 What the vernacular shows

The convergence is easy to mock and easy to dismiss as a matter of fashion. It is more serious than that for two reasons. The first is that visual taste is the form of taste most widely shared, so the ordinary person's visual environment is where a society's discrimination is formed, and an environment saturated in the same gaudy median teaches that the median is what images are. The second is that the forms in which the look appears, posters, group posts, and filters, are exactly those that people once made themselves, badly or well, and from which they learned. The poster made by a friend with a marker taught its maker something about hierarchy, emphasis, and restraint. The generated poster gives its requester little occasion to learn the same things, since the choices of hierarchy and emphasis were made elsewhere, and it replaces the friend. The likely result, though this is a hypothesis and not a measured one, is a population with less practice at making and a visual world with fewer examples of people trying, which is a loss that no improvement in the average quality of the generated image can restore.

7 The Replies

7.1 Throughput has risen

The strongest reply is that the tools do help people ship, and that the DORA finding of improved throughput in 2025 is evidence against the thesis [9]. This is accepted. The thesis is not that nothing is gained but that what is gained is the quantity that is easiest to measure and the least related to long-run value, while what is lost is the set of properties that determine the cost of ownership. A development that raises delivery throughput and delivery instability together has increased the rate at which change is made without increasing the rate at which correct change is made, and whether this is a net benefit depends on the cost of failures and of the work that follows them, which short-horizon metrics do not capture. The reply also assumes that more software is better. For many purposes the relevant quantity is the size of the problem solved, and an organization that ships ten times as many features may have simply made its product ten times harder to use.

7.2 Bad software and bad media are older than this technology

It is true, and often said, that software was bad long before the current generation of models, that most of what is made in any medium is mediocre, and that complaints about slop revive older complaints about the commercialization of culture. One analysis of the backlash puts it clearly: slop is what happens when production outpaces imagination, and generative systems only removed the friction [17]. The concession is correct and supports the argument.

What the technology changed is the rate at which production can outpace judgment, and the removal of friction is exactly what the argument identifies as the cause. A claim that nothing is new because mediocrity is old would be like observing that fire has always existed in answer to a complaint about the speed at which a particular accelerant spreads it.

7.3 The models are improving

The strongest form of this reply points out that the studies measure earlier systems, and that improvements in reasoning, context length, and agentic review will reduce duplication and error masking. This may happen, and the essay should not claim to know otherwise. Two things limit its force. The first is that the security evidence shows no improvement with larger or newer models over successive years [29, 33], which suggests that some failure modes are not a matter of capability that scale cures. The second is structural. A better model lowers the cost of review as well as the cost of generation, but it lowers the cost of generation first and furthest, because generation is what the system is optimized to do, and an equilibrium in which each of the two costs falls but the ratio of generation to review cost keeps widening would reproduce the problem at a higher level. A change in the direction of the argument would require that the improvements be applied on purpose to verification, deletion, and the reduction of code, which is a matter of how they are used and not of how good they are.

7.4 Production has been democratized

It is also said that the tools allow many people who could not write programs or draw to make things, and that complaints about quality are the complaints of a priesthood defending its guild. The access is real and valuable, and the argument has no quarrel with someone building a tool for their own use. The difficulty lies elsewhere. Software that anyone can create must still be maintained by someone, and a technology that democratizes the creation of programs while leaving maintenance as costly as before produces a growing stock of software that has an author and no one who can keep it alive. Democratization of production without democratization of understanding creates dependence on the tool that produced the program and not independence from specialists. Moreover, the public impact of software does not depend on who wrote it. A flawed authentication layer harms its users regardless of the creator's good intentions.

7.5 The backlash against slop is itself undiscriminating

A complication deserves to be admitted openly. In May 2026 an artist posted what appeared to be an AI-generated image in the style of Monet, and asked viewers to explain why it was inferior to a real Monet [17]. The observation drawn from episodes of this kind is that hostility to generated work is often applied without discrimination, so that the reflex of rejection can become a kind of slop of its own. This should be granted. It is not an objection to the thesis and is rather a confirmation of one of its premises, which is that taste is a trained capacity and not an attitude. A culture that responds to generated work with reflexive disgust has not recovered discrimination any more than one that responds with reflexive enthusiasm, and what is wanted in both cases is the ability to say, of a particular thing, what is wrong with it and what would make it right. The episode is offered

as illustrative of a kind of response and not as evidence of its frequency. How common undiscriminating rejection is, given that a single episode shows only that it occurs, is an empirical question, and the argument of this essay does not depend on its answer.

8 An Alternative Course

The argument would be incomplete if it said only that a turn was wrong. A turn is wrong relative to another available, and several were available, and are still. The alternatives follow from the model of the previous section and are not a matter of aesthetic preference: the levers on leakage are the rate of verification, the rate of generation, and the rule of admission, and an alternative course is one that works on them.

The first alternative is to point the models at the properties that the evidence says are being lost. A system that is good at reading a large codebase could be used to find the existing function that a new change should call, to propose the deletion of code that duplicates other code, to trace the consequences of a change through callers, to write the test that would have caught a regression, and to explain to a maintainer what a contribution does and what it breaks. Each of these reduces the amount of code that must be held in mind, and each increases the speed of verification, and none requires the production of a large volume of new text. Measured against the criterion stated earlier, whether a tool reduces what must be held in mind and makes the remainder easier to check, these are the uses that count as advances, and they are the uses for which the incentives of the present course provide least reward.

The second is a change of metric. Organizations that count the share of code written by machines should instead count the lines removed, the dependencies eliminated, the time to understand a module, the rate of regressions, and the age of the oldest code that nobody dares touch. Because it is hard to move such numbers by generation, they would reorient the use of the tools toward the work that matters. The third is a change of default. Features based on generation should be opt-in, their outputs labeled, and their absence a supported configuration, so that adoption is a choice made by people who have a reason, which is the only condition under which the market could say anything about their worth. The fourth is a preference, in how systems are built, for small programs composed of tested parts over large generated ones, with human authorship concentrated at the points where a decision shapes everything downstream: the interface, the data model, the boundaries, and the things the system refuses to do. A person who keeps those decisions keeps the essential difficulty in the right hands while letting the machine reduce the accidental one.

The four alternatives share a limitation that the fifth mechanism of the earlier section makes plain. If the present course is sustained by deployment by default, then voluntary alternatives are what that course forecloses, since an individual developer or a small venue cannot decline a feature that has been built into every editor, operating system, and platform, and cannot change the reward that a platform pays for engagement. The levers of the minimal model, raising verification, lowering generation, and holding admission to verified work, are largely institutional and not personal. They require collective action that no individual can supply: opt-in defaults enforced by vendors or regulators, procurement standards that require provenance and verification for software acquired by public bodies, labeling of generated material so that audiences and maintainers can discount it, and

platform reward systems that stop paying for volume. The small refusals described earlier, such as venues declining generated flyers and maintainers declining unreviewed contributions, are examples of how such norms begin, and they are worth supporting for that reason, but they are the start of a corrective and not the whole of one.

None of this is a novelty. It is the old discipline of software engineering and of craft in general, restated for a technology that makes it easy to forget. The argument for it does not rest on nostalgia. It rests on the claim that the cost of keeping software alive is the dominant cost and that practices which lower it are the ones that deserve to be called progress.

9 What Would Count Against the Thesis

An argument of this kind owes its readers the conditions under which it would be abandoned. It would be weakened by evidence from independent, preregistered studies that long-lived codebases developed with heavy generative assistance show no greater duplication, churn, or defect density than comparable codebases developed without, after controlling for team and domain. It would be weakened by a replication of the controlled trial with current tools, larger samples, and varied populations, finding that experienced developers are in fact faster and that their self-assessments are accurate. It would be weakened if the stability penalty found in the DORA series reversed and remained reversed, so that adoption were associated with fewer change failures and less rework. It would be weakened if the security failure rates fell sharply with model improvement and not merely with prompting scaffolds. And the claim about taste would be weakened if populations raised on generated material proved better, by an independent test of discrimination, at telling competent work from fluent imitation than populations raised without it, which is an empirical matter not yet studied well, and which is difficult to test by comparing populations because exposure to generated material is entangled with education, age, and technology use. A more tractable test would randomly assign participants to sustained periods of constructing artifacts, of selecting among generated candidates, and of revising generated drafts, and would then assess them on unfamiliar artifacts with independently scored tasks requiring error detection, design discrimination, and justified rejection, using a rubric for design critique applied by raters who are blind to condition, with inter-rater reliability reported for the scoring of rejection justifications. The comparison should measure both immediate performance and transfer to new tasks, since that would distinguish improved proficiency at selecting from the acquisition of a more general discriminating competence.

Those who would defend the present course have a coherent case: that the gains are real, that the problems are the problems of any new technology and are being fixed, that the alternatives described above can be built on top of what exists, and that the sensible policy is to proceed and to measure. The position of this essay can be put in its most general and defensible form: generative abundance becomes destructive when institutions treat the availability of an artifact as sufficient reason for its admission, and it does not follow that generation necessarily weakens taste or that every use of the tools degrades what it touches. Within that frame, the counter-position of this essay is that the present course already includes its own measurements, and that those measurements, with the caveats given, point in a direction that the incentives of the course make hard to correct. If the

correction comes, it will come through institutions that decide that the unit of progress is the program that is understood and the object that is judged, and that spend the cheapness of generation on the work of refusing what ought not to exist, with individual discipline contributing to it but unable to supply it alone.

References

[1] 404 Media. We Are Living in a ChatGPT Flyer Pandemic. *404 Media*, n.d. `https://www.404media.co/we-are-living-in-a-chatgpt-flyer-pandemic/`.

[2] Axios Tampa Bay. Tampa Bay Businesses Are Getting Called Out for AI Slop. *Axios*, 17 September 2026.

[3] Axios Salt Lake City. As AI Flyers Proliferate, More Utah Businesses Promise Human-Only Advertising. *Axios*, 3 August 2026.

[4] Becker, J., Rush, N., Barnes, B., and Rein, D. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. arXiv:2507.09089, 2025. Primary source for the randomized trial.

[5] Brooks, F. P. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, 20(4):10–19, 1987.

[6] Brooks, F. P. *The Mythical Man-Month: Essays on Software Engineering*. Anniversary edition. Addison-Wesley, Reading, 1995. Includes the chapter "No Silver Bullet Refired."

[7] CO/AI. Oakland Music Venue Bans AI-Generated Flyers to Support Local Artists. *CO/AI*, 29 October 2026. Secondary report of an Instagram announcement.

[8] The Cool Down. Small Businesses Reject Wave of Near-Identical AI Flyers in Favor of Hand-Drawn Signs. *The Cool Down*, n.d. Secondary report, citing 404 Media.

[9] Google Cloud DORA. *State of AI-Assisted Software Development 2025*. Google, 2025. Primary source for the 2025 survey findings.

[10] Global News. Keep Human Culture Alive: Halifax Music Venue Takes a Stand Against AI Posters. *Global News*, n.d. `https://globalnews.ca/news/12045069/halifax-pub-ai-posters/`.

[11] Hume, D. Of the Standard of Taste. In *Four Dissertations*. London, 1757.

[12] InfoQ. DORA Report Finds AI Is an Amplifier in Software Development, but Trust Remains Low. *InfoQ*, September 2025. Secondary report of the 2025 DORA survey.

[13] KEYT. AI-Generated or Recycled Meme Content Appears on Facebook's Most Viewed Content List. *KEYT*, 2024. Secondary report of Stanford and Georgetown research on 120 Facebook pages, a preprint at the time of reporting.

[14] Lehman, M. M. Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.

[15] LeadDev. How AI-Generated Code Compounds Technical Debt. *LeadDev*, 2025. Secondary report of GitClear's AI Copilot Code Quality research.

[16] LeadDev. Code Maintainability Plummets in the AI Coding Era. *LeadDev*, 7 July 2026. Secondary report of GitClear and GitKraken research on 623 million code changes; the primary report was not consulted.

[17] ListenFirst. AI Slop: When the Internet Drowns in Synthetic Junk. ListenFirst Media, 2026.

[18] Liu, Y., Widyasari, R., Zhao, Y., Irsan, I. C., and Lo, D. Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild. arXiv:2603.28592, 2026. Primary source; observational.

[19] Meltwater. What the Rise of AI Slop Means for Marketers. Meltwater, n.d. Marketing analysis of social listening data.

[20] METR. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. METR blog, 10 July 2025. `https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/`.

[21] NPR. AI-Generated Spam Is Starting to Fill Social Media. Here's Why. *NPR*, 14 May 2024.

[22] Simon, S. Opinion: Art Fair Flyers Should Showcase Human Creativity. *NPR*, 5 April 2025.

[23] Pooler, F. Generative AI and the Slopification of Coffee Culture. *The Pour Over*, 23 October 2026. Secondary report of a January 2026 study of image generators trained on their own outputs.

[24] Stephens, R. DORA 2025: Measuring Software Delivery After AI. RedMonk, 18 December 2025. Quotes the 2025 DORA report.

[25] An Endless Stream of AI Slop: How Developers Discuss the Burden of AI-Assisted Software Development. arXiv:2603.27249, 2026. Primary source; qualitative.

[26] Defective, Amplified, Contested: Folk Conceptualizations of AI Slop. arXiv:2610.14231, 2026. Primary source.

[27] Strathern, M. Improving Ratings: Audit in the British University System. *European Review*, 5(3):305–321, 1997.

[28] Tech.co. What Is AI Slop and Why Is It All Over Your Facebook Page? *Tech.co*, 2024. Secondary report, citing 404 Media.

[29] Hale, C. Nearly Half of All Code Generated by AI Found to Contain Security Flaws, Even Big LLMs Affected. *TechRadar*, 2025. Secondary report of Veracode's 2025 study.

[30] TechRadar. AI Slop Has Escaped the Internet: From Event Posters to Cafe Menus, It's Suddenly Everywhere in the Real World. *TechRadar*, n.d. Opinion piece.

[31] TechTarget. Google DORA: Software Delivery Caught Up to AI Coding Tools. *TechTarget*, 2025. Secondary report, source here for the 2024 DORA figures.

[32] Veracode. AI-Generated Code Poses Major Security Risks in Nearly Half of All Development Tasks, Veracode Research Reveals. Press release, 30 July 2025. Vendor benchmark of 80 designed tasks.

[33] Veracode. Is AI-Generated Code Secure? Key Findings from the 2026 GenAI Code Security Report. Infographic, 2026. Vendor benchmark.