International Journal of Research in Computing

ORIGINAL RESEARCH
Volume 5, Issue 1 (2026)

Computational Knightian Uncertainty: Undecidability and the Limits of Cyber Risk Quantification in Software-Intensive Firms

Michél Nguyen

University of the People, 595 E Colorado Blvd Suite 623, Pasadena, USA

Corresponding Author: michel_ng@icloud.com ORCID: 000-0001-6834-4422

Abstract

Frank Knight's distinction between measurable risk and unmeasurable uncertainty is central in economics and finance. Contemporary practice often collapses uncertainty into risk by assuming that all material hazards can, in principle, be quantified given sufficient data and computation. This assumption breaks down when the asset in question is large-scale software.This paper argues that undecidability in computability theory, as exemplified by Turing's halting problem and Rice's theorem, creates a structural form of uncertainty for software-intensive firms that cannot be reduced to standard probabilistic risk. Many security, safety, and compliance properties of interest to insurers, acquirers, and regulators are non-trivial semantic properties of programs and therefore undecidable in general, even under idealized conditions of perfect code visibility and unlimited classical computation. We call the resulting residual uncertainty computational Knightian uncertainty (CKU): a component of uncertainty that persists even if all observable information is known and arbitrarily robust classical computation is available.We introduce structural opacity the extent to which a codebase resists compression into a small set of regular patterns under a chosen description language and explore approximate Kolmogorov complexity (KC) of codebases as one proxy for this opacity. We develop a conceptual model that links undecidability, structural opacity, and observable outcomes, including cyber incident severity, cyber insurance loss experience, and merger and acquisition (M&A) valuation discounts. In this model, KC and related metrics act as structural covariates that may correlate with CKU, rather than as direct risk measures.Two small synthetic simulations illustrate the empirical logic: first, a crude gzip-based compressibility index sharply separates highly regular from highly irregular synthetic code; second, a KC-like covariate is recoverable in regression when it truly affects incident severity and does not appear systematically when it does not. Our theoretical commitment is modest: computability results guarantee that CKU is non-zero for sufficiently expressive systems. The further claims that CKU is economically material in large, structurally opaque codebases and that structural metrics provide usable proxies are empirical hypotheses to be argued and tested, not consequences of undecidability alone.

KEYWORDS

Computational Knightian uncertainty, Knightian uncertainty, Undecidability, Cyber risk, Rice's theorem, Halting problem, Cyber insurance, Kolmogorov complexity

ARTICLE HISTORY

Published: 14 Jan 2026

DATA/CODE AVAILABILITY

All data and code are included in this published article.

SDG ALIGNMENT

SDG 8
SDG 9
SDG 16

1 Introduction

Modern firms increasingly derive their value from software. Trading platforms, payment systems, cloud services, and AI-enabled products are built on large, evolving codebases that are tightly coupled to configuration artefacts, infrastructure-as-code, and third-party components. Failures or breaches in these systems can lead to material losses and systemic events, as documented in cyber-loss and insurance studies [8], [9] and large-scale software security reports [11].

Risk management practice, cyber risk models, and cyber insurance markets often treat software-related loss as if it were like other insurable perils: random but ultimately quantifiable given enough data, controls, and computation. This view is broadly consistent with economic analyses of information security that model incentives and expected-loss trade-offs under probabilistic assumptions [17].

Computability theory challenges this assumption. Turing's halting problem and Rice's theorem show that there is no general algorithm that can decide, for every program, whether it has a given non-trivial semantic property [1], [2]. Many properties that matter for security, safety, and regulatory compliance are of exactly this form. This implies that there are aspects of software behavior that cannot be exhaustively ruled out or certified by any algorithm, even under idealized conditions of perfect code visibility and unlimited classical computation.

This paper develops the concept of computational Knightian uncertainty (CKU) to capture this structurally irreducible component of uncertainty arising from software behavior. It explores how the structural opacity of codebases, approximated via Kolmogorov complexity, might shape firm-level digital risk and how structural metrics could act as covariates for CKU in empirical models.

Research question. This paper asks whether measurable structural properties of codebases especially compressibility-based opacity metrics can serve as functional covariates for the irreducible component of residual digital risk implied by undecidability.

Practical motivation. If such irreducible uncertainty exists and varies systematically across codebases, it affects how organizations interpret assurance results, how regulators define checkable compliance constraints, and how insurers and acquirers price software-intensive assets.

1.1 Structure of the argument

To calibrate the claims, we separate the argument into three layers. Only the first layer is a formal consequence of computability theory; the second and third are empirical hypotheses about economic materiality and measurement.

Existence (theoretical layer) : For sufficiently expressive software systems those that are at least Turing-complete undecidability results guarantee that some non-trivial semantic properties relevant to safety, security, and compliance cannot be decided algorithmically. At the firm level, this implies the existence of a structural component of residual digital risk that cannot be eliminated by perfect information or unlimited classical computation. We refer to this component as computational Knightian uncertainty (CKU) and formally show in Section 2 that it is strictly positive in principle for such systems. This component is distinct from residual risk due to incomplete data, model misspecification, or adversarial behavior, all of which are, at least in principle, reducible.

Plausible economic relevance (conceptual layer) : Theory alone does not determine whether this CKU component is economically material. It may be negligible in small, tightly controlled systems and more significant in large, structurally opaque ones. We argue that in large, complex codebases—especially those that embed interpreters, domain-specific languages, and configuration-as-code—undecidable and practically unverifiable behaviors inhabit a rich, irreg-

ular space of possibilities that cannot be exhaustively ruled out. This step relies on structural opacity and architecture, not on new theorems.

Empirical proxies and hypotheses (empirical layer): Finally, we propose approximate Kolmogorov complexity and related structural metrics as covariates that may correlate with CKU and, in turn, with total residual risk. This yields testable hypotheses about the relationships among structural opacity, incident severity, and market outcomes, such as insurance losses and M&A valuation discounts. The simulations in Sections 2.4 and 3 are deliberately simple. Still, they illustrate how these metrics behave in empirical models and how a CKU-related structural covariate might be identified statistically alongside standard software metrics

1.2 Contributions

Within this structure, the paper contributes four elements.

First, we provide a formal definition of CKU as the component of uncertainty about software behavior that persists even under complete code visibility and unlimited classical computation due to the undecidability of relevant semantic properties. This definition distinguishes CKU from conventional residual, model, or parameter uncertainty, which could, in principle, be reduced through improved measurements or better models.

Second, we give a worked example in an automated trading context, where a regulatory invariant is informally shown to be undecidable for a reasonably expressive class of systems, with the assumptions explicitly stated and the implications for regulatory practice spelled out.

Third, we conceptualize structural opacity and verifiable cores and argue that using approximate KC, alongside traditional metrics such as lines of code and cyclomatic complexity [3]–[6], serves as a proxy for structural opacity. A small synthetic experiment shows that a crude gzip-based compressibility index sharply separates highly regular from highly irregular code.

Fourth, we develop a firm-level conceptual model and empirical logic that relate CKU and structural metrics to residual risk and observable outcomes, together with a focused empirical agenda and a simulation study that demonstrates that a KC-like covariate can be recovered in regression when it truly affects incident severity. In this framework, structural metrics are treated as structural covariates that may modulate CKU's impact on observable outcomes, rather than as direct risk measures.

These contributions address the research question stated in the Introduction:

Can measurable structural properties of codebases especially compressibility based opacity metrics serve as useful covariates for CKU in firm-level digital risk models?

The remainder of the paper develops the theoretical framing for this question and sketches an empirical agenda for answering it.

1.3 Scope and positioning

The paper is conceptual and methodological. It does not report empirical analyses on real firms. The goal is to articulate a theoretically grounded notion of computational Knightian uncertainty, connect it to the structural properties of codebases, and outline empirical designs for future work.

The intended audience is computer science, software engineering, and information systems researchers interested in the limits of software assurance; economists and risk modelers concerned with digital assets; and practitioners in cyber insurance and technology M&A who must reason about software-driven enterprise risk. The argument is deliberately modest: it identifies a formally grounded component of residual risk. It proposes structural metrics as potential covariates, rather than claiming that undecidability alone determines observed loss experience.

2 Methods

This section describes the conceptual framework, simulation designs, and analytical approach.

To avoid over-claiming, we distinguish between (i) theoretical claims that follow from computability theory (e.g., the existence of CKU in sufficiently expressive systems) and (ii) empirical hypotheses about economic materiality and correlations with observable outcomes. The simulations are illustrative and are not evidence from real organizations.

2.1 Conceptual framework

We adopt a simple taxonomy of uncertainty.

Aleatory risk refers to randomness within a known or assumed probabilistic model.

Epistemic uncertainty reflects a lack of information that can, in principle, be reduced by observation or analysis. Model uncertainty concerns disagreements about which of several plausible models is appropriate.

Structural or Knightian uncertainty refers to ignorance about the relevant state space or causal structure such that no well-founded probability distribution over outcomes can be specified.

The notion of Knightian uncertainty is interpreted in various ways. Some authors equate it with unknowable probabilities; others emphasize "unknown unknowns" or profound ambiguity. Following Frank Knight's original distinction [12] and later ambiguity-based treatments [13], [14], as well as robust-control perspectives on model uncertainty [15], we reserve the term for situations in which the relevant outcomes and mechanisms cannot be exhaustively enumerated or assigned stable probabilities, even in principle. In this paper, computational Knightian uncertainty (CKU) is introduced as a computationally grounded subtype of this structural uncertainty.

In the remainder of the paper, we use the term "uncertainty" as defined in the taxonomy summarized in Table I.

Table 1: TAXONOMY OF UNCERTAINTY USED IN THIS PAPER

Type	**Description**	**Reducible in principle?**	**Example in software / cyber risk**
Aleatory risk	Randomness within a known or assumed probabilistic model; outcomes vary even if the model is correct.	Yes, only in the sense of variance reduction (e.g., diversification), not elimination.	Random arrival times of independent phishing emails under a fixed intensity model.
Epistemic uncertainty	Lack of information that can, in principle, be reduced by observation, measurement, or analysis.	Yes, with better data, measurement, and analysis.	Unknown prevalence of a specific vulnerability in a fleet of servers before they are scanned.
Model uncertainty	Disagreement or ambiguity about which of several plausible models or parameterizations is appropriate.	Often partly reducible via model comparison, validation, and expert judgment.	Competing loss models for cyber incidents (e.g., different frequency–severity or dependency structures).
Knightian / structural uncertainty	Ignorance about the relevant state space or causal structure; outcomes and mechanisms cannot be exhaustively enumerated or assigned stable probabilities.	Not generally reducible within a fixed probabilistic modelling framework.	Novel classes of attacks or failure modes not captured in existing risk taxonomies or scenarios.

Type	Description	Reducible in principle?	Example in software / cyber risk
Computational Knightian uncertainty (CKU)	Component of structural uncertainty that persists even under complete information about code and environment and unlimited classical computation, because relevant semantic properties are undecidable.	No, unless system expressiveness or architecture is changed (e.g., via verifiable cores or expressiveness restrictions).	Residual uncertainty about whether any admissible input sequence could trigger a regulatory breach in an expressive trading system, given the undecidability of the associated invariant.

Computability theory provides the formal backdrop. Turing's seminal work introduced the notion of a Turing machine and showed that there is no general algorithm that can decide all mathematical questions of a certain class. Rice's theorem strengthens this insight: any non-trivial semantic property of programs is undecidable in general. Many properties that matter for security, safety, and regulatory compliance are of exactly this form. These undecidability results are robust to hardware changes and persist even when arbitrarily fast or parallel classical computation is assumed. We define CKU as follows.

Definition (Computational Knightian uncertainty):

CKU is the component of uncertainty about a software system's future behaviors that persists even under complete information about its code and environment and unlimited classical computational resources, because no algorithm exists that can decide all relevant semantic properties of the system.

Interpretation of "risk" under Knightian uncertainty. In a Knightian framing, a firm may not have a single well-founded probability distribution over future cyber losses; uncertainty may instead be represented as a set of plausible models or scenarios [12] - [15]. In this paper, I use "risk" in the operational sense of residual loss exposure. When a scalar summary is needed for decision-making (e.g., pricing or capital allocation), it can be obtained by applying an appropriate risk functional (e.g., worst-case, robust, or scenario-based measures) to that exposure set. CKU identifies a component of this exposure that persists even under complete information about code and environment and unbounded classical computation.

Let R_i denote the residual digital risk faced by firm i after applying all feasible assurance, monitoring, and governance measures. Conceptually, we decompose R_i as.

$$R_i = R_i^{\mathrm{CKU}} + R_i^{\mathrm{NC}} \tag{1}$$

where R_i^{CKU} captures loss scenarios rooted in behaviors associated with undecidable properties that no algorithm could fully rule out, and R_i^{NC} captures residual risk from non-computational sources such as implementation errors, organizational failures, measurement noise, model misspecification, and adaptive adversaries within decidable regions.

Computability results guarantee that, for sufficiently expressive systems those that are at least Turing complete $R_i^{\mathrm{CKU}} > 0$ in principle. They do not, by themselves, establish that R_i^{CKU} is large relative to R_i^{NC} for any firm. Whether CKU is economically material is a system dependent question that depends on the codebase, architecture, operational environment, and threat model. It is important to separate the three levels:

- Undecidability as a *logical property* of a class of programs.
- Structural opacity as a *structural property* of a particular code base within such a class.
- Compressibility and related quantities are *measurement choices* that may correlate with opacity.

CKU, as defined above, is determined solely by the first level. It does not depend on how we measure code structure. Structural metrics, such as approximate Kolmogorov complexity, are introduced later as empirical covariates that may correlate with variation in R_i^{CKU} across systems. The core argument proceeds in three steps:

i. undecidability guarantees that $R_i^{\mathrm{CKU}>0}$ in expressive systems;

ii. in large, structurally opaque systems, CKU is plausibly non-negligible relative to other residual risk components;

iii. structural metrics, including approximate KC, are hypothesized to capture aspects of codebase structure that influence the magnitude of R_i^{CKU} and hence may correlate with observable outcomes.

What CKU does not cover

CKU does not exhaust all forms of digital risk. It is a structurally motivated lower bound on residual uncertainty that persists even under idealized information and computation, not a substitute for other sources of uncertainty.

First, there can be substantial cyber risk in systems with very low CKU, for example, when simple misconfigurations, well-understood coding errors, or social engineering attacks dominate the loss profile. Second, many important uncertainties arise from adversary behavior, organizational incentives, and changing legal or market environments; these may be difficult to model but are conceptually distinct from undecidability. Third, practical verification and testing are constrained by time, resources, and available tools; these constraints can leave significant residual risk even in systems whose relevant properties are, in principle, decidable.

The role of CKU in this paper is therefore not to explain all residual digital risk, but to highlight a formally grounded component that is often overlooked when software is treated as an ordinary insurable asset whose failure modes are, in principle, fully quantifiable.

What CKU adds beyond residual and model risk

Standard notions of residual or model risk typically arise from limitations of practice: imperfect data, finite computational resources, mis-specified models, or disagreement among experts. In principle, such risks could be reduced by collecting more data, improving models, or investing in more powerful computation. CKU differs in three ways.

First, CKU implies a hard lower bound on reducible uncertainty for sufficiently expressive software systems. Even under perfect measurement of code and environment, and even with unbounded classical computation, undecidability guarantees that some economically relevant semantic properties cannot be algorithmically decided. This implies that no amount of additional data or model refinement can drive R_i^{CKU} to zero.

Second, CKU motivates architectural interventions that are qualitatively different from those suggested by standard residual risk. If part of the risk stems from undecidable properties, then constraining expressiveness, isolating verifiable cores, and restricting how highly expressive components interact with safety-critical logic become central design moves. These interventions change the structure of the state space itself, rather than merely improving estimates within a fixed model.

Third, CKU provides a computational justification for incorporating structural metrics as covariates in firm-level risk models. If undecidability-driven uncertainty is plausibly more significant in large, structurally opaque codebases, then metrics that proxy for opacity such as approximate KC offer a way to capture variation in R_i^{CKU} across firms that is orthogonal to standard exposure and control variables.

In short, CKU does not replace residual or model risk. Still, it adds a logically grounded layer of uncertainty that persists even under idealized modelling assumptions and that naturally points to specific architectural and measurement strategies.

2.2 Structural opacity and verifiable cores

Algorithmic information theory defines the Kolmogorov complexity K(x) of a finite string x as the length of the shortest program that outputs x on a fixed universal machine. K(x) is not computable in general, but compression-based methods (including normalized compression distance) and block decomposition techniques provide practical approximations [3], [16]. We distinguish three related notions:

- Undecidability concerns the logical limits of what can be decided about arbitrary programs.
- Structural opacity concerns how a particular codebase is organized within that undecidable design space.
- Approximate KC and compressibility metrics are measures that aim to summarize aspects of the structure.

Structural opacity is the extent to which a codebase's behavior resists compression into a small set of regular patterns under a chosen description language, so that any faithful description must be comparatively long and irregular. Intuitively, a structurally opaque codebase inhabits a wide and irregular space of potential behaviors, many of which are difficult to characterize or rule out. Within a given undecidable program class, different systems can be structurally opaque, and CKU is plausibly more economically significant in the opaque tail.

A verifiable core is a subsystem or architectural region whose structure and expressiveness are intentionally constrained so that essential properties fall within decidable or tractable fragments, enabling strong assurance. High-assurance microkernels such as seL4 [7], which provide formally verified functional correctness with a small, trusted computing base, are canonical examples. Similar ideas appear in formally verified cryptographic libraries and proof-carrying code sandboxes that restrict untrusted modules to a carefully specified interface. These cores are typically small relative to the overall system that depends on them. CKU is plausibly small within verifiable cores and larger in the structurally opaque regions surrounding them.

Traditional metrics such as lines of code (LOC) and cyclomatic complexity (CYC) are local or module-level: they characterize control flow within functions, dependencies between modules, or histories of change. They can miss global patterns and long-range dependencies that matter for undecidability-driven risk. Examples include codebases that embed interpreters or domain-specific languages, extensive metaprogramming or reflection that ties together distant modules, and configuration-driven systems that encode behavior declaratively across many files.

Approximate KC, estimated via compression or block decomposition, is sensitive to repeated structure across the entire codebase. It reflects how succinctly the code can be described globally under a fixed encoding scheme. A large codebase that is highly regular and dominated by templates, boilerplate, or generated code will compress strongly; a large, idiosyncratic codebase will compress less. Holding other metrics constant, higher approximate KC indicates a wider and more irregular space of behaviors, within which undecidable, untestable, and hard-to-approximate properties are more likely to have economically relevant consequences.

Although Kolmogorov complexity is defined for individual finite strings, real codebases are versioned, multi-language repositories that also include configuration files, build scripts, schemas, and even data. In practice, one works with normalized snapshots of the monorepo and applies approximate measures such as Normalized Compression Distance over the concatenated artefacts, or block decomposition methods (BDM) rather than simple gzip. These constructions are, from a strict algorithmic information-theoretic perspective, impure, and they can be confounded by factors such as language choice, formatting conventions, extensive code generation, or the inclusion of large binary artefacts. Empirically, these factors can be addressed through controls (e.g., language fixed effects), per-module normalization, or by separating generated code from handwritten code.

In this context, compressibility is proposed as a structural covariate rather than a direct measure of CKU: undecidability guarantees that $R_i^{\mathrm{CKU}} > 0$; structural opacity, which influences how large that component may be in practice, and approximate KC provides one way to measure opacity variation across systems.

2.3 Example: undecidable regulatory invariant

A stylized example from automated trading illustrates how undecidability can attach to an economically meaningful regulatory property. Consider a class $\mathcal{C}$ of trading systems that process potentially unbounded sequences of market events and client orders. Let Σ be a finite event alphabet and let $E \subseteq \Sigma^*$ denote the set of admissible event sequences (i.e., sequences that satisfy basic syntactic and regulatory constraints). We assume that E is sufficiently expressive to contain encodings of arbitrary bit strings, in particular encodings $\mathrm{enc}(M, x)$ of Turing machines M and inputs x.

We also assume that the strategy language and execution environment are Turing-complete: there exists a computable mapping g that takes a pair (M, x) and produces a configuration $S_{M,x}$ in this system class such that, under some admissible event sequence $e_{M,x} \in E$, the system's internal computation simulates $M(x)$. These assumptions are standard in undecidability reductions and are meant to capture the fact that many real trading stacks embed interpreters, scripting languages, or configuration-as-code layers.

Formally, the reduction relies on three assumptions:

(A1) The admissible language E admits an effective encoding of (M, x).

(A2) The system class $\mathcal{C}$ can simulate $M(x)$ for some admissible event sequence.

(A3) Account balances can be made to depend on whether the embedded computation halts (e.g., via a designated account whose update rule references an internal state flag).

For each client account a and event sequence e, let $B(a, t_{\text{close}}(e))$ denote the net capital position of a at the end of the trading day induced by e. Consider the following regulatory invariant:

$$P : \forall e \in E,\ \forall a,\ B(a, t_{\text{close}}(e)) \geq 0 \tag{2}$$

i.e., for all admissible event sequences $e \in E$ and all accounts a, the closing balance $B(a, t_{\text{close}}(e))$ is non-negative.

We sketch a standard reduction from the halting problem to deciding whether P holds for systems in this class.

1. **Encode the computation.**

   Given a Turing machine M and input x, construct a trading strategy configuration $S_{M,x}$ in the expressive strategy language such that some admissible event sequence $e_{M,x} \in E$ drives a faithful simulation of $M(x)$.

2. **Introduce a special account.**

   Add a designated client account $A_{M,x}$ whose balance is decremented by a fixed amount if and only if the simulated computation of $M(x)$ halts. For all other event sequences and all other accounts, the system's behavior ensures that no end-of-day balances become negative.

3. **Relate halting to violation of the invariant.**

   By construction, $M(x)$ halts if and only if there exists an admissible event sequence $e \in E$ leading to a negative end-of-day balance for $A_{M,x}$. Thus, $M(x)$ halts if and only if the property P fails for the system instance $S_{M,x}$.

If there were a universal procedure that decided P for all systems in this class, i.e., that could determine, for any candidate system, whether all accounts end each day with non-negative capital on all admissible sequences, then that procedure could be used to decide, for any M and x, whether $M(x)$ halts. This would solve the halting problem, contradicting its undecidability. Thus, invariant P is undecidable for this expressive class of trading systems. Equivalently, P is a *non-trivial semantic property* of programs; by *Rice's theorem*, such properties are undecidable on Turing-comp

The construction is deliberately stylized, but it explains why regulators and system designers often restrict the expressiveness of trading algorithms and maintain verifiable cores: without such restrictions, even natural regulatory invariants may fall into undecidable territory. In practice, these restrictions take the form of bounded loop constructs, limited state, domain-specific languages with constrained semantics, and strict segregation between highly expressive components and safety-critical settlement logic.

2.4 Simulation design

To illustrate and test parts of the empirical layer, we design two small synthetic simulations. The goal is not to emulate any real-world system, nor to simulate undecidability directly, but to show how

1. a compressibility-based opacity index
2. a KC-like structural covariate behave in simple, controlled settings and in standard regression models.

We are interested in whether these metrics:

Implementation and Reporting

I implemented both simulations in Python 3.11.2 using `numpy` 1.24.0, `pandas` 2.2.3, and `statsmodels` 0.14.3. To address robustness, Simulation 1 was repeated across 15 random seeds and Simulation 2 across 100 random seeds. I report 95% confidence intervals across seeds for key quantities, including compression ratios, regression coefficients, and R^2. For Simulation 2, I additionally report variance inflation factors (VIFs) as a basic collinearity diagnostic and the distribution of p-values for the opacity coefficient under the null hypothesis ($\beta_K = 0$).

- behave sensibly as proxies for structural opacity
- can be statistically identified alongside conventional metrics such as LOC and CYC when they truly matter, without generating spurious effects when they do not.

Simulation 1: gzip-based opacity index

Simulation 1 examines whether a simple gzip-based compressibility index serves as an opacity proxy when considering code structure alone.

We construct 30 synthetic "regular" repositories. Each consists of many functions with identical if-ladders and predictable returns, so that the overall structure is highly repetitive and easily described.

We construct 30 synthetic "irregular" repositories of comparable size, each with random function names, identifiers, and call targets, and with little repeated structure. These are designed to mimic large amounts of idiosyncratic "glue" code.

For each repository, we compute a crude opacity index as the gzip compression ratio of the UTF-8 source text:

$$\text{Opacity ratio} = \frac{\text{Compressed size}}{\text{Original size}} \tag{3}$$

Python code is given in Appendix A. We treat this index as a simple, baseline approximation to structural opacity: lower ratios indicate highly regular code that compresses well; higher ratios indicate more idiosyncratic structure. This simulation speaks only to structural regularity; it does not model CKU or incident processes directly, but it tests whether even a rudimentary compressor can sharply distinguish "regular" from "irregular" code under controlled conditions.

2.5 Simulation 2: opacity covariate in regression

Simulation 2 explores how a KC-like opacity covariate behaves in regression alongside lines of code (LOC) and cyclomatic complexity (CYC), when the covariates are realistically correlated. We generate n=400 synthetic systems with:

- LOC drawn lognormally,
- CYC drawn lognormally
- An opacity index Kconstructed as a noisy linear combination of LOC and CYC.

Log incident severity is generated as

$$\log(\text{severity}) = \beta_0 + \beta_{\text{LOC}} \cdot \text{LOC} + \beta_{\text{CYC}} \cdot \text{CYC} + \beta_K \cdot K + \varepsilon \tag{4}$$

Scenario A uses $\beta_K > 0$, so opacity truly matters in the data-generating process. Scenario B uses $\beta_K = 0$, so opacity has no effect. In both scenarios, the true coefficients on LOC and CYC are small but positive. We deliberately allow K to be correlated with LOC and CYC so that collinearity is realistic, mirroring the empirical situation in which larger, more complex systems also tend to be more structurally opaque.

For each scenario, I estimate linear regressions of log severity on structural metrics: a baseline model with LOC and CYC only, and a model that additionally includes K. Appendix B reports executable Python code for a single reproducible run given a fixed seed. To quantify sampling variability and robustness, I also repeat the simulation 100 times with random seeds and summarize the resulting uncertainty intervals and diagnostics in Section 3.2.

This simulation is not an empirical test of CKU itself, but a check that an opacity like covariate (i) has a recoverable effect when it truly influences severity, and (ii) does not systematically appear significant when it does not, despite realistic correlations with other structural metrics.

3 Results

3.1 Simulation 1: structural opacity from gzip ratios

Across 15 random seeds (each generating n = 30 repositories per group), Simulation 1 yields a sharp separation between the two synthetic code regimes. The regular repositories have a gzip compression ratio of 0.00887 (constant across seeds in this construction).

- For irregular repositories, the mean compression ratio is 0.51623, with a 95% interval across seeds of [0.51619, 0.51628]. Within a seed, the standard deviation of ratios across the n = 30 irregular repositories typically ranges from 0.00012 to 0.00019.
- Thus, even a crude compressor-based index can distinguish highly regular from highly irregular synthetic code under controlled conditions. The point is not the absolute level of the ratio, which varies with language, formatting, and repository layout, but the ability of compression to serve as a coarse proxy for structural regularity.

Real repositories will lie between these extremes and will mix regular patterns with idiosyncratic "glue" code. Empirical applications would therefore require careful normalization (e.g., with respect to language and build artifacts) before treating compressibility as an opacity covariate.

This toy experiment is deliberately stylized and is used only to motivate the use of compressibility-based proxies; it does not measure CKU directly, and it does not substitute for empirical validation on real codebases.

3.2 Simulation 2: opacity covariate in regression

I report Simulation 2 results aggregated over 100 random seeds (n = 400 synthetic systems per seed), which provides uncertainty intervals and a basic robustness check.

- Scenario A ($\beta_K = 0.12$). In the baseline model with LOC and CYC only, the median R^2 is 0.312, with a 95% interval of [0.230, 0.378] across seeds.
- When K is added as a covariate, the median R^2 increases to 0.318, with a 95% interval of [0.233, 0.383]. The median R^2 gain from including K is 0.0054 (95% interval [0.0000, 0.0270]).
- Across seeds, the estimated K coefficient has a mean of 0.1160, with a 95% interval of [0.0016, 0.2441].
- The median p-value for the K coefficient is 0.079, and the coefficient is significant at the 5% level in 43% of runs, reflecting a moderate effect size and collinearity with other structural metrics.

Omitting K inflates correlated coefficients: the mean CYC coefficient is 0.0546 without K, but 0.0408 with K (true $\beta_{\text{CYC}} = 0.04$). The mean LOC coefficient remains close to its true value ($\beta_{\text{LOC}} = 0.000008$) in both specifications.

- Scenario B ($\beta_K = 0$). The estimated K coefficient is centered near zero: mean -0.0040 with a 95% interval of $[-0.1184, 0.1241]$.
- Under the null, K is significant at the 5% level in 5% of runs, as expected under nominal Type I error. The median p-value for K is 0.533.
- Adding K under the null yields only a negligible fit gain: the median R^2 increase is 0.0008 (95% interval [0.0000, 0.0091]).
- Collinearity diagnostics are consistent with the design: variance inflation factors are approximately 1.02 for LOC and have a median of approximately 3.77 for CYC and K (95% intervals of 3.12–4.41 and 3.14–4.42, respectively).
- Because the disturbance term ε is Gaussian by construction, OLS assumptions hold in this synthetic setting; the goal is therefore not to validate OLS, but to illustrate how an opacity-like covariate behaves under controlled collinearity.

Overall, Simulation 2 suggests that opacity-like structural covariates can be statistically identified when they matter, but inference can be noisy when K they are correlated with standard metrics supporting their use as a complementary covariate rather than a standalone predictor.

In empirical applications, this motivates reporting uncertainty intervals, assessing collinearity, and testing robustness (e.g., by considering alternative specifications or tail-focused models) when integrating opacity proxies into cyber risk models.

These simulations remain illustrative; they do not test CKU directly, but they demonstrate that the proposed measurement layer behaves as expected in controlled settings.

4 Discussion

The simulations are deliberately simple, but they perform two useful methodological functions.

First, Simulation 1 shows that a basic gzip-based compressibility index behaves as expected when comparing highly regular and highly irregular synthetic code. Although authentic codebases are more complex and involve many factors, this supports treating compressibility as a first-order proxy for structural opacity. In more realistic settings, one would use stronger compressors or block decomposition methods and would normalize for language, formatting, generated code, and repository layout. Still, the basic rank ordering is already evident.

Second, Simulation 2 demonstrates that the regression specification proposed for empirical work behaves sensibly in a controlled setting: it recovers an opacity effect when one exists and does not systematically produce one when it does not. This matters because one might worry that a KC-like covariate is so correlated with other metrics that regression analysis would either always show a significant effect (a collinearity artefact) or never identify it. In real data, one would expect opacity to matter most in the high-LOC, high-complexity tail, suggesting the value of interaction terms and tail-focused methods such as quantile regression.

More broadly, the paper's conceptual contribution is to show how undecidability results translate into a structural, computational form of Knightian uncertainty for software-intensive firms. CKU is not merely a philosophical anecdote; it is a logical limit on what automated assurance processes can deliver, even under ideal information and unlimited classical computation. This limit is small but strict in verifiable cores, and plausibly larger and economically meaningful in structurally opaque systems, particularly where safety and security depend on complex emergent behavior.

4.1 Practical implications for organizations

- CKU matters in practice because many organizational processes implicitly assume that, with enough effort, software behavior can be fully characterized, and residual risk can be translated into calibrated probabilities. Undecidability does not imply that assessment is futile; instead, it means that some uncertainty remains even under ideal information and computation, which affects how assurance results should be interpreted.
- Decision-making and governance. CKU motivates interventions that change the system structure, rather than merely adjusting estimates within a fixed-risk model. Examples include isolating verifiable cores, constraining expressiveness in safety- critical logic, and using domain-specific languages or bounded resources to keep key invariants within decidable fragments.
- Regulation and compliance. CKU provides a rationale for regulatory approaches that favor checkable constraints (e.g., bounded strategy languages, auditable interfaces, and precise separation between expressive components and settlement or safety-critical modules). It also clarifies why compliance regimes based purely on ex post testing can leave irreducible gaps.
- Risk assessment and insurance. For cyber risk modelling and underwriting, CKU suggests that model outputs should be interpreted as conditional on a structural lower bound: two firms with similar controls can still differ in irreducible uncertainty due to architecture and codebase opacity. Structural metrics such as compressibility-based indices can therefore complement control inventories and help prioritize deeper reviews and due diligence.
- Technology M&A and due diligence. In acquisitions, CKU helps explain why software complexity can translate into valuation discounts or larger escrows and representations: even with full repository access, some semantic properties of large systems cannot be fully determined. Lightweight opacity metrics provide one screening signal to triage where deeper technical diligence is most valuable.

4.2 Empirical agenda and data requirements

Validating the empirical layer requires real-world data that link code structure to security outcomes and losses. The objective is not to measure CKU directly, but to test whether structural opacity metrics add explanatory power beyond conventional metrics (e.g., LOC, cyclomatic complexity) and beyond control-based assessments.

One line of work is to study open-source ecosystems in which code histories and vulnerability disclosures are publicly available. A practical design would use project–release snapshots as the unit of analysis, compute structural metrics and opacity proxies for each snapshot, and model outcomes, such as the appearance of severe vulnerabilities in subsequent releases, using logistic, count, or survival models with project fixed effects.

A complementary line of work is to use anonymized cyber insurance portfolios or software-focused M&A datasets to evaluate whether opacity proxies correlate with observed loss experience, underwriting judgments, or valuation discounts, controlling for firm size, sector, and governance controls. In both settings, careful normalization for language, generated code, and repository layout is essential.

4.3 Limitations

Several limitations of the present work warrant note. The simulations use synthetic code and stylized loss processes; they do not provide evidence on real-world firms. The proposed opacity metrics are coarse and can be confounded by language choice, code generation, formatting practices, and repository layout. CKU addresses only the undecidability-driven component of residual risk and does not capture many important adversarial, organizational, and environmental uncertainties. Finally, the trading example is deliberately stylized; actual regulatory systems use a variety of engineering and institutional mechanisms to keep critical properties within decidable fragments. Most importantly, the paper does not use real-world organizational datasets; assembling such data is essential for testing the empirical hypotheses.

Despite these limitations, the structure of the argument and the simulations make the empirical agenda concrete. The claim is not that CKU renders quantitative risk modelling impossible, but that it identifies a formally grounded layer of residual uncertainty that is invisible to models which implicitly assume that all relevant properties of large software systems are, in principle, decidable and quantifiable.

4.4 Conclusion

This paper introduced computational Knightian uncertainty (CKU) to describe the component of software-related uncertainty arising from formal undecidability, even under perfect code visibility and unlimited classical computation. Using Rice's theorem and a stylized trading example, we argued that no algorithm can, in general, determine many economically meaningful properties of large software systems, and that this implies a non-zero, structurally grounded component of residual digital risk.

We proposed structural opacity, approximated via Kolmogorov complexity, as a conceptual link between undecidability and firm-level risk. In large, structurally opaque systems, the region of behaviors governed by undecidable or practically unverifiable properties is plausibly large enough to have economic significance. KC-based compressibility measures are imperfect and sensitive to implementation details. Still, they provide a practical starting point for quantifying structural opacity alongside traditional software metrics such as lines of code and cyclomatic complexity.

Two simple simulations showed, first, that a gzip-based opacity index sharply separates regular from irregular synthetic code, and second, that a KC-like covariate can be recovered by regression when it truly affects incident severity, without appearing to do so when it does not. These simulations do not replace empirical studies on real codebases, but they indicate that the proposed empirical specification is workable and that opacity effects are, in principle, statistically detectable.

Future work should conduct pilot studies on open-source ecosystems to test whether compressibility-based metrics add predictive power for defects and vulnerabilities beyond standard metrics; explore alternative structural measures of opacity, such as graph-based architecture descriptors and richer approximations to Kolmogorov complexity; and integrate structural metrics into cyber risk and insurance models in ways that respect their limitations. Case studies of firms that deploy verifiable cores and large-scale formal methods would also help quantify the extent to which CKU can be reduced in practice. Doing so will require real-world data that pairs repository snapshots with incident and loss outcomes.

If these strands converge, CKU and structural opacity could become applicable concepts not only for theorizing about the limits of assurance, but also for structuring discussions between engineers, risk managers, insurers, and regulators about the kinds of risk that remain even under best practice—and about how much complexity premium or discount should attach to software-intensive assets in markets and regulatory regimes.

5 Appendix A – Synthetic compressibility experiment (Simulation 1)

This appendix provides reproducible Python code for Simulation 1, which compares gzip compression ratios for synthetic regular and irregular repositories.

```
import numpy as np
import pandas as pd
import gzip
import io
import random
import string

np.random.seed(1234)
random.seed(1234)

def gzip_ratio(text: str) -> float:
    data = text.encode("utf-8")
    buf = io.BytesIO()
    with gzip.GzipFile(fileobj=buf, mode="wb") as f:
        f.write(data)
    compressed_size = len(buf.getvalue())
    return compressed_size / len(data)

def make_regular_repo(n_funcs=200, n_cases=20):
    lines = []
    for f_id in range(n_funcs):
        lines.append(f"def␣f_{f_id}(x):")
        for i in range(n_cases):
            lines.append(f"␣␣␣␣if␣x␣==␣{i}:")
            lines.append(f"␣␣␣␣␣␣␣␣return␣{i}")
        lines.append("␣␣␣␣return␣-1")
    return "\n".join(lines)

def random_ident(n=8):
    return ''.join(random.choice(string.ascii_letters) for _ in range(n
        ))

def make_irregular_repo(n_funcs=200, n_cases=20):
    lines = []
    for f_id in range(n_funcs):
        fname = random_ident()
        lines.append(f"def␣{fname}(x):")
        for i in range(n_cases):
            a = random_ident()
            b = random_ident()
            lines.append(f"␣␣␣␣if␣{a}␣==␣{b}:")
            lines.append(f"␣␣␣␣␣␣␣␣{random_ident()}_{i}({random_ident()
                })")
        lines.append(f"␣␣␣␣return␣{random_ident()}")
    return "\n".join(lines)

# generate 30 regular and 30 irregular repos
regular_repos = [make_regular_repo() for _ in range(30)]
irregular_repos = [make_irregular_repo() for _ in range(30)]

# compute gzip ratios
regular_ratios = [gzip_ratio(r) for r in regular_repos]
```

```
irregular_ratios = [gzip_ratio(r) for r in irregular_repos]

# summarise
regular_summary = pd.Series(regular_ratios).describe()
irregular_summary = pd.Series(irregular_ratios).describe()

print("Regular repos opacity summary:\n", regular_summary)
print("\nIrregular repos opacity summary:\n", irregular_summary)
```

Running this code reproduces the statistics discussed in Section 3.1.

6 Appendix B – Regression simulation with an opacity covariate (Simulation 2)

This appendix provides Python code for Simulation 2, which studies the behavior of an opacity covariate alongside LOC and cyclomatic complexity in a linear regression.

```
import numpy as np
import pandas as pd
import statsmodels.api as sm

def simulate_opacity_dataset(n=400, beta_k=0.12, noise_sd=0.5,
                             seed=5678, k_noise_sd=0.4):
    np.random.seed(seed)

    # structural metrics
    loc = np.random.lognormal(mean=10, sigma=0.6, size=n)
    cyc = np.random.lognormal(mean=2.5, sigma=0.4, size=n)

    # opacity index K correlated with LOC and CYC
    k_true = 0.00003 * loc + 0.12 * cyc + np.random.normal(0,
       k_noise_sd, size=n)

    # true coefficients
    beta_loc = 0.000008
    beta_cyc = 0.04

    epsilon = np.random.normal(0, noise_sd, size=n)

    log_severity = (
        -6
        + beta_loc * loc
        + beta_cyc * cyc
        + beta_k   * k_true
        + epsilon
    )

    df = pd.DataFrame({
        "LOC": loc,
        "CYC": cyc,
        "K": k_true,
        "log_severity": log_severity
    })
    return df

# Scenario A: K has a true positive effect
df_A = simulate_opacity_dataset(beta_k=0.12)
```

```
X_base = sm.add_constant(df_A[["LOC", "CYC"]])
X_k    = sm.add_constant(df_A[["LOC", "CYC", "K"]])

model_base = sm.OLS(df_A["log_severity"], X_base).fit()
model_k    = sm.OLS(df_A["log_severity"], X_k).fit()

print("Scenario A: KC-like opacity has a true effect\n")
print("Baseline model (no K):")
print(model_base.summary())
print("\nModel with K:")
print(model_k.summary())

# Scenario B: K has no true effect
df_B = simulate_opacity_dataset(beta_k=0.0)

X_B = sm.add_constant(df_B[["LOC", "CYC", "K"]])
model_B = sm.OLS(df_B["log_severity"], X_B).fit()

print("\nScenario B: KC-like opacity has no true effect\n")
print("Model with K included:")
print(model_B.summary())
```

7 Declarations

Funding
This research received no external funding.

Conflicts of Interest
The author declares that there are no conflicts of interest.

Ethical Considerations
The author states that all work related to this research was conducted in accordance with institutional, national, and international guidelines and in compliance with recognized ethical standards. This article does not contain any studies with human participants or animals performed by the author.

AI Usage Statement
Generative AI tools were used in the preparation of this manuscript in the following way: Grammarly was used for language editing.

Data Availability Statement
All data generated or analyzed during this study are included in this published article.

Code Availability Statement
All the important software codes developed for this study are included in this published article.

SDG Alignment
This research is aligned with the following United Nations Sustainable Development Goals (SDGs):
SDG 8 – Decent Work and Economic Growth;
SDG 9 – Industry, Innovation & Infrastructure;
SDG 12 – Responsible Consumption & Production;
SDG 16 – Peace, Justice & Strong Institutions;

References

[1] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," Proc. London Math. Soc., ser. 2, vol. 42, pp. 230–265, 1937, doi: 10.1112/plms/s2-42.1.230.

[2] H. G. Rice, "Classes of recursively enumerable sets and their decision problems," Trans. Amer. Math. Soc., vol. 74, no. 2, pp. 358–366, 1953, doi: 10.1090/S0002-9947-1953-0053041-6.

[3] M. Li and P. Vitányi, An Introduction to Kolmogorov Complexity and Its Applications, 3rd ed. New York, NY, USA: Springer, 2008, doi: 10.1007/978-0-387-49820-1.

[4] T. J. McCabe, "A complexity measure," IEEE Trans. Softw. Eng., vol. SE-2, no. 4, pp. 308–320, Dec. 1976, doi: 10.1109/TSE.1976.233837.

[5] N. E. Fenton, "Software metrics: successes, failures and new directions," J. Syst. Softw., vol. 47, nos. 2–3, pp. 149–157, 1999, doi: 10.1016/S0164-1212(99)00035-7.

[6] N. E. Fenton and M. Neil, "A critique of software defect prediction models," IEEE Trans. Softw. Eng., vol. 25, no. 5, pp. 675–689, May 1999, doi: 10.1109/32.815326.

[7] G. Klein et al., "seL4: formal verification of an OS kernel," in Proc. 22nd ACM Symp. Operating Systems Principles (SOSP), Big Sky, MT, USA, Oct. 2009, pp. 207–220, doi: 10.1145/1629575.1629596.

[8] C. Biener, M. Eling, and J. H. Wirfs, "Insurability of cyber risk: an empirical analysis," Geneva Papers Risk Insur. – Issues Pract., vol. 40, no. 1, pp. 131–158, 2015, doi: 10.1057/gpp.2014.19.

[9] R. He, Z. Jin, and J. S.-H. Li, "Modeling and management of cyber risk: a cross disciplinary review," Ann. Actuarial Sci., vol. 18, no. 1, pp. 1–32, 2024, doi: 10.1017/S1748499523000258.

[10] J. Kay and M. King, Radical Uncertainty: Decision Making Beyond the Numbers. New York, NY, USA: W. W. Norton, 2020.

[11] Veracode, Inc., "State of Software Security 2023: Annual Report on the State of Application Security," Veracode, Burlington, MA, USA, 2023. [Online]. Available: https://info.veracode.com/rs/790-ZKW-291/images/Veracode_State_o_Software_Security_2023.pdf. .

[12] F. H. Knight, Risk, Uncertainty, and Profit. Boston, MA, USA: Houghton Mifflin, 1921.

[13] . Ellsberg, "Risk, ambiguity, and the Savage axioms," Q. J. Econ., vol. 75, no. 4, pp. 643–669, 1961, doi: 10.2307/1884324.

[14] I. Gilboa and D. Schmeidler, "Maxmin expected utility with non-unique prior," J. Math. Econ., vol. 18, no. 2, pp. 141–153, 1989, doi: 10.1016/0304-4068(89)90018-9.

[15] L. P. Hansen and T. J. Sargent, Robustness. Princeton, NJ, USA: Princeton Univ. Press, 2008, doi: 10.1515/9781400829385.

[16] R. Cilibrasi and P. M. B. Vitányi, "Clustering by compression," IEEE Trans. Inf. Theory, vol. 51, no. 4, pp. 1523–1545, Apr. 2005, doi: 10.1109/TIT.2005.844059.

[17] R. Anderson and T. Moore, "The economics of information security," Science, vol. 314, no. 5799, pp. 610–613, 2006, doi: 10.1126/science.1130992.