Distributional Contracts for Stochastic Systems

Contents

Distributional Contracts for Stochastic Systems

A very early draft, which is most definitely not ready for publication.

Abstract

In stochastic systems, correctness cannot in general be characterised solely by the outcome of a single execution, since individual executions may legitimately vary even when the system is behaving acceptably overall.

This paper extends Bertrand Meyer's Design by Contract [1][2] by lifting postconditions from Boolean predicates over individual executions to statistical assertions over repeated executions. We introduce the notion of a distributional contract, under which the quality of a stochastic system is characterised along two independent dimensions: functional stochasticity — whether the system produces acceptable results with sufficient probability — and temporal stochasticity — whether it responds within acceptable time bounds with sufficient probability. These dimensions require distinct statistical treatments but together address the two most fundamental quality concerns for a stochastic service.

Since the quantities of interest are not directly observable, they are estimated empirically from repeated executions and assessed using conservative statistical bounds. This yields an operational basis for making statistically grounded assertions, at a defined confidence level, that a stochastic system satisfies its contractual obligations.

Two Dimensions of Stochasticity

In deterministic software, correctness is defined pointwise: for any input satisfying the precondition, the postcondition must hold for the resulting output. This model is insufficient for stochastic systems, where individual executions may legitimately vary even when the system is behaving acceptably overall. The idea of assessing stochastic systems through repeated execution and statistical inference has precedent in the tradition of statistical model checking, where Younes and Simmons [9] showed that time-bounded probabilistic properties can be assessed by hypothesis testing over sampled executions, trading absolute certainty for bounded decision error, and Legay, Delahaye, and Bensalem [10] surveyed this approach as a scalable alternative to numerical model checking. We extend this line of reasoning from formal stochastic models to operational software services, and from temporal-logic property satisfaction to a contractual framing: the notion of a contract is lifted from individual executions to distributions over executions.

The uncertainty exhibited by stochastic systems manifests along two independent dimensions, each giving rise to a distinct form of distributional contract.

The first is **functional stochasticity**: whether the system produces an acceptable result. Given identical input, a stochastic system may produce correct output on some executions and incorrect output on others. The correctness of the output is a random variable. The distributional contract over this dimension asserts that the system succeeds with probability at least some minimum acceptable level.

The second is **temporal stochasticity**: how long the system takes to respond. Even among successful executions, response times vary substantially. A service that typically responds in 200 milliseconds may occasionally take several seconds. Latency is not a fixed property of the system; it is a distribution. The distributional contract over this dimension asserts that a given percentile of the response time distribution falls below an acceptable bound — for example, that the 95th percentile latency does not exceed 500 milliseconds.

These dimensions are independent. A fast response can be incorrect; a slow response can be correct. A system can satisfy its functional contract while routinely breaching its latency contract, or vice versa. Neither dimension subsumes the other, and a distributional contract that addresses only one leaves the other unexamined.

Crucially, the two dimensions require different statistical treatments. Functional stochasticity reduces, in the evaluable case, to a binary outcome for each execution — the quality predicate either holds or it does not — and is naturally modelled as a Bernoulli process. Not all executions are evaluable, a distinction addressed shortly. Temporal stochasticity, by contrast, is a continuous quantity; each execution yields a duration, and the contractual assertion is over the shape of the resulting distribution rather than over a binary success rate. The sections that follow develop the Bernoulli framework for functional stochasticity in detail. The statistical treatment of temporal stochasticity, which rests on empirical percentile analysis rather than binomial inference, is addressed separately.

Functional Stochasticity

Let $Q(x, y) \in \{0, 1\}$ be a quality predicate indicating whether an execution with input x and output y is acceptable. For a stochastic routine, the contractual obligation is not that every execution satisfies Q, but that the probability of satisfying Q exceeds a minimum acceptable level:

$$p = \mathbb{P}(Q(x, y) = 1) \geq p_{\text{min}}$$

Here, p denotes the unknown success probability and p_{min} the minimum acceptable success rate. The origin of p_{min} is deliberately left open: it may be stipulated externally, for example by an SLA or regulation, or derived empirically from baseline observations.

Since p is not directly observable, it must be estimated from repeated executions, yielding $\hat{p}$. This constraint — that the system is an opaque executable whose properties must be inferred from observed behaviour rather than from an inspectable internal model — is shared with the black-box probabilistic verification tradition. Sen, Viswanathan, and Agha [11] consider statistical model checking of black-box probabilistic systems where properties are inferred from observed traces, and Aichernig and Tappler [12] extend this to stochastic systems with probabilistic behaviour. The present work inherits their black-box constraint but reframes the objective: rather than establishing reachability or property satisfaction within automata-based frameworks, the goal is to assess whether the system meets a contractual quality obligation. A conservative lower confidence bound $L(\hat{p}, n, \alpha)$, such as the Wilson lower bound, is then used to assess whether this obligation is satisfied. The contract is taken to hold if

$$L(\hat{p}, n, \alpha) \geq p_{\min}$$

which provides statistical evidence, at confidence level $1 - \alpha$, that the system meets its contractual obligation.

Two operational modes arise naturally. In the first, a stipulated contract is verified against an externally defined threshold $p_{\min}$. In the second, an empirical baseline experiment is used to derive a conservative threshold, which is then enforced in subsequent evaluations. In this way, Design by Contract is generalised from pointwise correctness to distributional correctness for systems whose behaviour is inherently non-deterministic.

Operational Outcome Model

In operational settings, the raw outcome space of a service is not necessarily binary. Besides acceptable and unacceptable results, executions may also end in catastrophic outcomes such as resource exhaustion, process termination, or other infrastructure-level failures. These outcomes fall outside the ordinary semantic notion of a returned result, but they remain part of the observed behaviour of the system and must therefore be accounted for explicitly.

We therefore distinguish three classes of operational outcome for run i:

$$Y_i \in \{\text{pass}, \text{fail}, \text{catastrophic}\}$$

The quality predicate $Q(x, y)$ is a partial function: it is defined only when the execution produces a result that can be meaningfully evaluated. When $Y_i = \text{catastrophic}$, no such result exists, and Q is accordingly undefined. The binary variable X_i used for statistical analysis is therefore

$$X_i = \begin{cases} 1 & \text{if } Y_i = \text{pass} \\ 0 & \text{if } Y_i = \text{fail} \\ \text{undefined} & \text{if } Y_i = \text{catastrophic} \end{cases}$$

Catastrophic outcomes are not failed trials; they are non-evaluable observations. The evaluation predicate is never applied, because there is no result against which it could be applied. In practice, the evaluation is simply skipped.

The Bernoulli model is therefore defined only over the evaluable subset of executions — those for which Q is defined. This distinction matters: conflating catastrophic outcomes with semantic failures would distort the estimated success probability by attributing to the quality predicate outcomes that it was never in a position to judge.

This does not mean that catastrophic outcomes are ignored. Their frequency is itself an observable quantity of operational concern, but it is orthogonal to the distributional contract over Q and must be monitored separately.

Q as a Bernoulli Trial

To make stochastic quality statistically analysable, each repeated execution is evaluated against a fixed acceptance criterion — which is, in Meyer's terms, the routine's postcondition. The difference is that here the postcondition is not expected to hold on every execution, but its satisfaction on each execution is observed and recorded. This induces a binary observation $X_i \in \{0, 1\}$, where $X_i = 1$ if the execution satisfies the postcondition and $X_i = 0$ otherwise.

These observations may be modelled as Bernoulli trials with parameter p, provided several assumptions are approximately satisfied: the acceptance criterion is applied consistently, repeated executions are approximately independent, and the underlying success probability remains stable over the sampling window.

Under these conditions, the Bernoulli model provides the basis for estimating pass rates, constructing confidence bounds, and deriving probabilistic contractual verdicts.

In an operational setting, we can add guardrails, which go at least some of the way to ensuring Bernoulli trials' assumptions are satisfied. Concrete measures will be discussed in later sections.

Deriving Thresholds from Baselines

The core statement identified two operational modes for establishing p_{min}. In the first, the threshold is stipulated externally — by a contract, service level agreement, regulatory requirement, or organisational policy. Such a threshold is a normative claim: it expresses what the system *ought* to achieve, independently of what it has been observed to achieve. No experiment is required; the statistical question is simply whether the system meets the mandated requirement.

In the second mode, no external threshold exists. The system's acceptable performance level is not known in advance and must be discovered empirically. This is the common case for systems whose behaviour is inherently stochastic, where the developer cannot stipulate a success rate from first principles but must instead observe what the system actually achieves under controlled conditions. Services built on Large Language Models are natural candidates for this mode: an LLM invoked with a fixed system prompt, model version, and temperature setting may be reasonably expected to exhibit a consistent level of stochasticity over time, making the baseline estimate a stable foundation for subsequent evaluation — provided the operational profile remains comparable.

The Baseline Experiment

To establish an empirical threshold, a baseline experiment is conducted: the system is executed n_{exp} times under controlled conditions, and the observed success rate $\hat{p}_{\text{exp}} =$

k/n_{exp} is recorded. This point estimate serves as the best available characterisation of the system's current behaviour. It is important to note, as Musa [13] established in the context of software reliability engineering, that any such empirical estimate is meaningful only relative to the distribution of inputs under which the system is exercised. A baseline measured under one operational profile may not generalise to another — an insight that motivates the covariate tracking discussed in later sections.

However, $\hat{p}_{\text{exp}}$ cannot be used directly as a threshold for subsequent evaluations. The point estimate is subject to sampling variability: a different run of the same experiment would yield a different value of $\hat{p}$. Setting $p_{\text{min}} = \hat{p}_{\text{exp}}$ would amount to requiring the system to meet a standard that is an artefact of a particular sample, not a stable property of the system. In practice, this leads to frequent false alarms — tests that reject a system whose true success probability has not changed, simply because the test sample happened to fall below the experimental estimate.

Conservative Threshold Derivation

The solution is to derive p_{min} not from the point estimate itself, but from a conservative lower bound on the success probability that the baseline supports. Specifically, the one-sided Wilson lower bound is applied to the experimental results:

$$p_{\text{min}} = L(\hat{p}_{\text{exp}}, n_{\text{exp}}, \alpha)$$

This yields a threshold with the following property: if the system's true success probability is at least as high as the baseline estimate, a subsequent evaluation will reject falsely with probability at most α. The threshold is deliberately set below $\hat{p}_{\text{exp}}$, absorbing the uncertainty inherent in the estimate, so that only genuine degradation — not sampling noise — triggers a failure.

The gap between $\hat{p}_{\text{exp}}$ and the derived p_{min} depends on the baseline sample size. A larger experiment produces a tighter confidence bound and hence a threshold closer to the observed rate. A smaller experiment produces a wider bound and a correspondingly more conservative threshold. This is the correct behaviour: less evidence warrants less precision, and the threshold reflects this honestly.

Choice of Confidence Bound

The lower bound $L(\hat{p}, n, \alpha)$ used throughout this framework is the Wilson score interval [3]. This choice warrants brief justification, since alternative methods exist and differ in important respects.

The Wilson score interval is constructed by inverting the score test for a binomial proportion. For n observations with observed success rate $\hat{p}$ and critical value $z = z_\alpha$, the one-sided lower bound is:

$$L(\hat{p}, n, \alpha) = \frac{\hat{p} + \frac{z^2}{2n} - z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

This bound has several properties that make it well-suited to the present framework: it produces valid bounds in $[0, 1]$ for all values of $\hat{p}$ and all sample sizes, it maintains correct coverage for small n, and it remains well-defined at the boundary cases $\hat{p} = 0$ and $\hat{p} = 1$.

The most commonly encountered alternative is the Wald interval, which approximates the binomial distribution by a normal distribution centred on $\hat{p}$. For moderate success rates and large samples, the Wald interval is adequate. However, it degenerates at the extremes of the reliability range. When $\hat{p}$ is close to 0 or 1, the Wald interval produces bounds that fall outside $[0, 1]$ — a nonsensical result for a probability. When $\hat{p} = 1$ (all trials successful), the Wald standard error is exactly zero, collapsing the interval to a single point and providing no margin for uncertainty whatsoever. These are not obscure edge cases; they arise routinely in practice, precisely when a system is performing well. For a thorough comparative analysis of interval estimation methods for the binomial proportion, see Brown et al. [4] and Agresti and Coull [5].

The Wilson score interval avoids these pathologies entirely. Because it is derived by inverting the score test rather than substituting the point estimate into the variance, it produces valid bounds across the full parameter space, including the boundary cases where Wald fails.

At the other end of the spectrum, more sophisticated methods exist — exact Clopper-Pearson intervals [6], Bayesian credible intervals, and various higher-order corrections. These offer marginal improvements in coverage accuracy under specific conditions, but they add analytical and computational complexity that is difficult to justify in this setting. The Bernoulli model is itself an approximation: the assumptions of independence and stationarity are, as discussed, only approximately satisfied in operational environments. Investing in a more precise confidence bound while the underlying trial model is imperfect amounts to false precision — a more accurate answer to a question that was only approximately well-posed. The Wilson bound occupies the right point in this trade-off: it is correct across the full parameter space, simple to compute, and honest about the level of precision that the framework can actually support.

Stipulated and Derived Thresholds in a Common Framework

Despite their different origins, stipulated and empirically derived thresholds enter the same statistical framework. In both cases, the evaluation collects n observations, computes the lower confidence bound $L(\hat{p}, n, \alpha)$, and compares it against p_{min}. The statistical machinery is identical; what differs is the interpretation. A failure against a stipulated threshold constitutes evidence that the system does not meet an external requirement. A failure against an empirically derived threshold constitutes evidence that the system has degraded from its measured baseline.

Temporal Stochasticity

The preceding sections developed the distributional contract for functional stochasticity, in which each execution yields a binary outcome and the contractual assertion is over the success probability. Temporal stochasticity requires a fundamentally different treatment. Each execution yields a continuous quantity — its duration — and the contractual assertion is not over a success rate but over the shape of a distribution.

Why Latency Demands Distributional Reasoning

A single observed response time tells us almost nothing about a system's temporal behaviour. Service latency distributions are typically right-skewed, often multimodal, and frequently heavy-tailed. A system with a 200ms mean may have a 95th percentile of 350ms or of 2000ms — the mean alone cannot distinguish the two. What matters operationally is not the average case but the tail: the experience of the worst-served fraction of requests. This is why latency contracts are expressed not in terms of means but in terms of percentiles.

The Latency Population

Not all executions contribute to the latency distribution. Catastrophic outcomes, as discussed earlier, produce no evaluable result and are excluded. But failed executions — those for which $Q(x, y) = 0$ — are also excluded from the latency population. A fast failure (an immediate validation rejection at $t \approx 0$) and a slow failure (a timeout at $t = 30{,}000$ms) both reflect error paths, not the latency of successful operation. Including them would contaminate the distribution with durations that are artefacts of failure modes rather than measurements of service responsiveness.

The latency distribution is therefore conditional on success. Let $\mathcal{L} = \{t_i : X_i = 1\}$ be the set of durations from evaluable executions that satisfied the quality predicate, and let $n_s = |\mathcal{L}|$. The distributional contract over temporal stochasticity applies to this conditional distribution $T \mid X = 1$, not to the unconditional distribution over all attempts. Functional stochasticity and temporal stochasticity thus describe two complementary aspects of the same set of executions: the probability of success, and the latency distribution given success.

Empirical Percentile Estimation

Because latency distributions resist parametric characterisation, the contract is assessed using non-parametric empirical percentiles computed directly from the observed order statistics. No assumptions are made about the shape of the underlying distribution.

Let $t_{(1)} \leq t_{(2)} \leq \cdots \leq t_{(n_s)}$ be the order statistics of the successful latencies. For a percentile level $p \in (0, 1]$, the empirical percentile is:

$$Q(p) = t_{(\lceil p \cdot n_s \rceil)}$$

This is the nearest-rank method: the p-th percentile is the smallest observed value such that at least a proportion p of observations fall at or below it.

The Latency Assertion

A latency contract specifies a set of percentile constraints:

$$\mathcal{C} = \{(p_j, \tau_j) : j = 1, \ldots, m\}$$

where p_j is a percentile level (e.g., 0.50, 0.90, 0.95, 0.99) and τ_j is the maximum acceptable value at that percentile. For each constraint, the assertion evaluates:

$$\text{PASS}_j \iff Q(p_j) \leq \tau_j$$

The overall latency contract is satisfied if and only if all constraints are met:

$$\text{PASS}_{\text{latency}} = \bigwedge_{j=1}^{m} \text{PASS}_j$$

As with functional stochasticity, the threshold τ_j may be stipulated externally — by an SLA specifying, say, that the 95th percentile latency must not exceed 500ms — or derived empirically from a baseline experiment. The same dichotomy of mandated versus empirical thresholds applies, with the same interpretive distinction: a breach of a stipulated threshold is evidence of non-compliance; a breach of an empirically derived threshold is evidence of degradation.

Threshold Derivation for Latency

When τ_j is derived from a baseline, the observed percentile $Q_{\text{baseline}}(p_j)$ is, like $\hat{p}_{\text{exp}}$ in the functional case, a point estimate subject to sampling variability. Using it directly as a threshold would again invite false alarms.

The threshold is therefore set as a conservative upper bound on the baseline percentile:

$$\hat{\tau}_j = Q_{\text{baseline}}(p_j) + z_\alpha \cdot \frac{s}{\sqrt{n_s}}$$

where s is the sample standard deviation and n_s the number of successful observations in the baseline. The final threshold is $\tau_j = \max(Q_{\text{baseline}}(p_j),\ \lceil \hat{\tau}_j \rceil)$, ensuring integer-valued thresholds that are never tighter than the raw observation.

This formula uses the standard error of the mean as a proxy for the uncertainty of the percentile. It is an engineering approximation, not a formal non-parametric confidence interval for the quantile — which would require knowledge of the density at the percentile point, precisely the quantity that the non-parametric approach avoids assuming. The approximation is conservative in practice: the standard deviation s reflects the overall spread of the distribution, and the floor and ceiling adjustments provide additional margin. For baseline sample sizes typical of measurement experiments ($n_s \geq 500$), the resulting thresholds are close to what a bootstrap confidence interval would yield, at far lower computational cost.

A further consequence of the empirical approach is that small sample sizes trivially exclude higher percentiles from meaningful evaluation. The p-th percentile requires at least $\lceil 1/(1-p) \rceil$ observations before it is distinct from the sample maximum: for $p = 0.95$ this is 20, for $p = 0.99$ it is 100. When n_s falls below such a threshold, the corresponding percentile constraint cannot be assessed at all. And even at the boundary — $n_s = 100$ for $p = 0.99$, for instance — the percentile estimate rests on a single observation separating it from the maximum, which is hardly a basis for a meaningful contractual judgement. In practice, reliable percentile estimation requires n_s to be substantially larger than the theoretical minimum, so that the estimate is determined by a region of the distribution rather than by an isolated order statistic.

The Composite Verdict

With both dimensions defined, the overall distributional contract for a stochastic service is the conjunction of its functional and temporal contracts:

$$\text{PASS}_{\text{service}} = \text{PASS}_{\text{functional}} \wedge \text{PASS}_{\text{latency}}$$

A service must be both correct and responsive. A payment API that succeeds 99.5% of the time but takes 30 seconds for 1% of requests fails its contract just as surely as one that returns incorrect results. The two dimensions are assessed independently, using their respective statistical frameworks, and both must pass for the service to be judged acceptable.

Sample Size and Feasibility

Not every distributional contract can be meaningfully assessed with a given number of samples. The relationship between sample size, threshold, and confidence is not a free choice — it is a system of constraints in which fixing any two determines the third. This concern echoes a theme in the software reliability literature: Frankl, Hamlet, Littlewood, and Strigini [14] argue that testing methods should be evaluated in terms of the reliability they ultimately deliver in operation, not merely in terms of fault counts, and Hamlet and Taylor [15] demonstrate that success on a set of carefully chosen examples does not, by itself, justify confidence in future behaviour under realistic conditions. These results become sharper still for stochastic services, where the property of interest is inherently probabilistic and no single execution can be decisive. This section examines the three ways in which the constraint system can be resolved.

The Fundamental Constraint

A distributional contract involves three quantities that are bound together by the statistical framework: the sample size n, the threshold p_{min}, and the confidence level $1 - \alpha$. It is not possible to simultaneously achieve an arbitrarily tight threshold, high confidence, and a small sample. This is not a limitation of any particular method; it is a fundamental property of statistical inference. Any operational approach to parameterising a distributional contract must fix two of these quantities and accept that the third is determined.

Three natural approaches emerge, each corresponding to a different practical starting point.

Sample-Size-First: Cost-Driven

In many operational settings, the sample size is fixed by external constraints: the cost of each execution, the time available in a continuous integration pipeline, or the rate limits imposed by an external service. The question becomes: given a fixed n and a threshold p_{min}, what confidence does the resulting evaluation achieve?

The confidence is determined implicitly by the Wilson lower bound. For a given n and p_{min}, the achievable confidence is the level $1-\alpha$ at which the lower bound $L(1, n, \alpha) = p_{\text{min}}$ — that is, the confidence at which even a perfect observation ($\hat{p} = 1$) would just barely satisfy the contract. If the achievable confidence is unacceptably low, the evaluation remains valid but its evidential weight is limited. It may detect gross violations but will lack the sensitivity to distinguish small degradations from sampling noise.

Confidence-First: Risk-Driven

When the consequence of a false verdict is severe — in safety-critical systems, compliance audits, or contractual obligations — the confidence level and detection capability are the primary requirements. The question becomes: given a threshold p_{min}, a desired confidence $1 - \alpha$, and a minimum detectable effect δ, how many samples are required?

The minimum detectable effect is essential: it specifies the smallest degradation worth detecting, expressed as a drop from p_{min} to $p_{\text{min}} - \delta$. Without it, the sample size has no finite answer — detecting an arbitrarily small degradation requires an arbitrarily large sample.

Given these parameters, the required sample size for a one-sided test with power $1 - \beta$ is:

$$n = \left(\frac{z_\alpha \sqrt{p_{\text{min}}(1 - p_{\text{min}})} + z_\beta \sqrt{(p_{\text{min}} - \delta)(1 - (p_{\text{min}} - \delta))}}{\delta} \right)^2$$

where z_α and z_β are the critical values corresponding to the desired significance and power levels respectively.

The practical consequence is that stringent contracts demand large samples. Verifying a 99.9% threshold with the ability to detect a 0.1% degradation at 95% confidence and 80% power requires on the order of 8,000 samples. This is not excessive; it is the honest cost of making a statistically defensible claim about a system operating at that level of reliability.

Threshold-First: Baseline-Anchored

In the third approach, the threshold is taken directly from the empirical baseline — $p_{\text{min}} = \hat{p}_{\text{exp}}$ — without the conservative adjustment described in the section on threshold derivation. This is statistically aggressive: because the threshold equals the point estimate, a subsequent evaluation has approximately a 50% chance of falling below it by sampling variation alone, even if the system has not changed.

This approach is included for completeness. It is useful in contexts where the operator is deliberately exploring the trade-offs between threshold, sample size, and false positive rate, and is willing to accept the consequences. It should not be used where the verdict carries operational weight.

The Feasibility Gate

Regardless of which approach is taken, there exists for any given threshold and confidence level a minimum sample size below which no observation — not even a perfect one — can satisfy the contract. If every execution succeeds ($\hat{p} = 1$), the Wilson lower bound is:

$$L(1, n, \alpha) = \frac{n}{n + z^2}$$

For this bound to reach p_{min}, the sample size must satisfy:

$$n \geq N_{\text{min}} = \left\lceil \frac{p_{\text{min}} \cdot z^2}{1 - p_{\text{min}}} \right\rceil$$

If $n < N_{\text{min}}$, the contract is infeasible: the evaluation cannot produce a positive verdict under any outcome. This can and should be detected before any samples are collected. Proceeding with an infeasible configuration wastes resources on an evaluation whose conclusion is predetermined.

The feasibility gate scales steeply with the threshold. At the default confidence level of 95% ($z \approx 1.645$):

p_{min}	N_{min}
0.80	11
0.90	25
0.95	52
0.99	268
0.999	2,704
0.9999	27,058

A contract requiring $p \geq 0.9999$ cannot be verified with fewer than $27{,}058$ samples. This is not a deficiency of the method; it is the inescapable cost of making a credible assertion about a system operating at four nines of reliability.

The javai Project Family: Implementations of the Distributional Contract

The preceding sections developed the distributional contract as a statistical model. This section describes the javai project family, which implements the model as practical tooling for developers working with stochastic systems.

The javai project was created in response to a conspicuous gap in the testing landscape. The shortcomings of existing approaches to testing stochastic systems have been recognised in the literature. Christensen et al. [8] observe that ”in practice, testers resort to running probabilistic programs an arbitrary number of times; hoping to trigger existing bugs, but without any guarantees,” and propose ProbTest, a black-box unit testing method that uses the coupon collector's problem to determine sample sizes for bug detection in probabilistic programs. Their work confirms the need for statistically principled approaches and demonstrates that the problem admits tractable solutions, though their focus is on bug-finding rather than on contractual quality assurance over deployed services.

The advent of Large Language Models brought stochastic behaviour from the margins of software engineering — where it had been treated as accidental and undesirable — to the centre. Services built on LLMs are inherently non-deterministic: the same input genuinely produces different outputs, and the distribution of those outputs *is* the system's behaviour. Yet the available testing tools offered no principled way to express, measure, or enforce quality expectations over such systems. The javai project family addresses this gap by providing direct operational realisations of the distributional contract across multiple language ecosystems.

The methodology is implemented as language-native frameworks, each idiomatic to its ecosystem:

Framework	Language	Integration	Repository
punit	Java	JUnit 5 extension	The reference implementation
feotest	Rust	Cargo test integration	Idiomatic Rust, not a port
baseltest	Python	pytest plugin (planned)	—

Each framework provides a direct operational realisation of the distributional contract as described in this paper. Their features correspond to the elements of the model.

Expressing a Distributional Contract

Each framework gives the developer the means to express a distributional contract as a declarative specification. The functional dimension is defined through a service postcondition: a quality predicate Q that may be a conjunction of arbitrarily many conditions — syntactic validity, semantic correctness, presence of required fields, adherence to a schema — evaluated as a single composite assertion over the output of each execution. The developer may additionally declare latency thresholds — percentile-level bounds to which the service must adhere — expressing the temporal dimension of the contract.

All frameworks evaluate both dimensions using the statistical methods described in this paper: the Bernoulli model with Wilson score lower bounds for functional stochasticity, and empirical percentile analysis for temporal stochasticity. The composite verdict requires both dimensions to pass.

Stipulated and Empirical Thresholds

Both threshold sources are supported across all implementations. A developer may declare a stipulated threshold directly — from an SLA, regulatory requirement, or organisational policy — or may conduct a baseline measurement experiment from which the framework derives a conservative threshold using the Wilson lower bound. The baseline result is recorded as a persistent specification artefact, capturing the observed success rate, sample size, and derived threshold together with the conditions under which the baseline was established.

The Feasibility Gate

All implementations enforce the feasibility constraint described in this paper. Before any samples are executed, the framework determines whether the configured sample size can in principle support a positive verdict at the required confidence level. If it cannot, the evaluation is rejected as infeasible — a configuration error, distinct from a system failure. This prevents the waste of resources on evaluations whose conclusion is predetermined.

Early Termination

In some cases, the final verdict can be determined before all planned samples have been executed. If enough failures have been observed that the lower bound cannot reach p_{min} even if all remaining samples pass, continued execution is pointless. Conversely, if enough successes have been observed that the lower bound will exceed p_{min} regardless of remaining outcomes, the verdict is already determined. The frameworks detect both conditions and terminate early, which is of particular practical value when each sample involves an expensive operation such as an LLM invocation.

Operational Safeguards

The Bernoulli model assumes independence between executions and stationarity of the success probability across the sampling window. These assumptions are not given by the operational environment; they must be actively maintained. The frameworks provide several mechanisms toward this end. A warmup facility allows initial observations to be discarded, mitigating cold-start non-stationarity caused by caches, connection pools, or runtime compilation.

A particular threat to stationarity arises from exogenous covariates: factors external to the system under test that influence the success probability without being part of the input. In any population sampling exercise — and Bernoulli trial sampling is no exception — the validity of comparisons between samples depends on the conditions under which those samples were drawn. If the baseline was established under one set of conditions and the evaluation is conducted under another, the two samples are drawn from different populations, and the comparison is statistically meaningless regardless of the sample sizes involved. Deployment region, time of day, model version, infrastructure configuration, and upstream service behaviour are all examples of exogenous covariates that can shift the success probability between a baseline and a subsequent evaluation. The frameworks address this by associating covariate declarations with baselines, making contextual factors explicit and auditable, and issuing warnings when conditions at evaluation time differ from those recorded in the baseline.

Baseline expiration tracking alerts operators when a specification may no longer represent current system behaviour. These mechanisms do not guarantee that the Bernoulli assumptions hold — that is impossible in practice — but they make violations visible rather than allowing them to silently undermine inference.

The covariate problem points to a deeper reality: for many stochastic systems, behaviour is environmentally dependent in ways that cannot be fully captured at baseline time. A service that meets its distributional contract in a test environment may violate it in production, not because the system has changed but because the operating conditions have. It is therefore insufficient, in many cases, to rely on one-time evaluations conducted in controlled settings. What is needed is a mechanism for probing the system's behaviour in situ — in the environment where it actually operates, under the conditions it actually encounters. Sentinel capabilities within the javai family address this requirement, providing lightweight runtime agents that evaluate the distributional contract continuously against the live system, detecting degradation as it occurs rather than after the fact.

The Experiment-to-Test Workflow

The frameworks operationalise the measure-then-enforce workflow through a structured progression. A measurement experiment executes the system a large number of times under controlled conditions, producing a specification. Subsequent evaluations run with fewer samples and are assessed against the threshold derived from the specification. The use case — the reusable unit that invokes the system and evaluates the result — is shared between experiments and tests, ensuring that the quality predicate Q is applied consistently.

Conformance Across Implementations

Because the distributional contract is a statistical model, not a language-specific construct, all javai framework implementations must produce identical statistical results for the same inputs. The javai-R project provides the verification mechanism: it generates canonical reference datasets using R — the gold standard for statistical computing — against which each framework validates its statistics engine. This ensures that a distributional contract evaluated by punit in Java produces the same verdict as one evaluated by feotest in Rust, within stated floating-point tolerances.

Conclusion

Meyer's Design by Contract gave software engineering a powerful organising principle: the obligations between caller and callee, expressed as preconditions, postconditions, and invariants, and enforced deterministically at runtime. For over three decades this model has served well, because the systems it governed were, by and large, deterministic. A postcondition either held or it did not, and a single counterexample was definitive evidence of a defect.

Yet stochastic systems have always existed — network services subject to variable latency, randomised algorithms, probabilistic data structures, simulations — and Design by Contract has never had a satisfactory answer for them. A deterministic postcondition cannot express ”succeeds at least 95% of the time” or ”responds within 500ms at the 99th percentile”. These are inherently distributional claims, and no amount of Boolean logic over individual executions can capture them. This was always a shortcoming of the model, but one that could be overlooked as long as stochastic behaviour remained marginal. The rise of Large Language Models has made it impossible to ignore: when non-determinism is not a defect but the defining characteristic of the technology, the absence of a principled contractual framework for stochastic systems becomes an acute gap.

This gap can only be addressed by a probabilistic approach. When a system's behaviour is inherently variable, a single execution cannot characterise its quality, a single failure cannot condemn it, and a single success cannot vindicate it. The postcondition must be lifted from a Boolean predicate over an individual execution to a statistical assertion over a population of executions.

This paper has proposed such a lifting, drawing on established traditions — statistical model checking [9][10][16], black-box probabilistic verification [11][12], and software reliability engineering [13][14][15] — while recasting their insights in a contractual form suited to day-to-day software engineering practice. The distributional contract preserves the contractual structure of Design by Contract — the notion that a service has obligations to its consumers that can be formally stated and objectively assessed — while replacing pointwise evaluation with statistical inference. The contract is expressed along two independent dimensions: functional stochasticity, assessed via a Bernoulli model and the Wilson score lower bound, and temporal stochasticity, assessed via empirical percentile analysis. Both dimensions admit thresholds that are either stipulated externally or derived conservatively from empirical baselines, and both are subject to the same feasibility constraints that govern any honest statistical claim.

The model is deliberately conservative in its statistical choices. The Wilson bound was preferred not because it is the most sophisticated available technique, but because it is correct across the full parameter space and honest about the level of precision that the underlying Bernoulli approximation can actually support. False precision — a more elaborate statistical treatment applied to a model whose assumptions are only approximately satisfied — was judged a greater risk than the modest loss of tightness that conservatism entails.

The javai project family demonstrates that the distributional contract is not merely a theoretical construct but a practical tool. The frameworks translate each element of the model — the quality predicate, the threshold, the confidence bound, the feasibility gate, the baseline specification — into declarative constructs that a developer can express and a runtime can enforce, across multiple language ecosystems. Sentinel capabilities extend the model beyond one-time evaluation to continuous in-situ monitoring, acknowledging that stochastic systems are environmentally dependent and that a contract verified under controlled conditions may not hold in production.

Several questions remain open. The treatment of exogenous covariates in this paper is operational rather than formal: covariates are declared and tracked, but there is no statistical framework for adjusting inference when covariate drift is detected. The latency threshold derivation uses the standard error of the mean as a proxy for percentile uncertainty — an engineering approximation whose adequacy depends on sample size and distributional shape. And the independence assumption, while supported by operational safeguards, is ultimately unverifiable from the data alone.

These are not deficiencies to be apologised for; they are honest boundaries of what a practical framework can achieve. The distributional contract does not claim to resolve the fundamental difficulty of reasoning about stochastic systems. It claims only to provide a disciplined, statistically grounded basis for doing so — one that makes its assumptions explicit, its limitations visible, and its verdicts defensible.

References

[1] Meyer, B. ”Applying 'Design by Contract'.” *IEEE Computer*, vol. 25, no. 10, October 1992, pp. 40–51.

[2] Meyer, B. *Object-Oriented Software Construction*, 2nd ed. Prentice Hall, 1997.

[3] Wilson, E. B. ”Probable inference, the law of succession, and statistical inference.” *Journal of the American Statistical Association*, vol. 22, no. 158, 1927, pp. 209–212.

[4] Brown, L. D., Cai, T. T., and DasGupta, A. ”Interval estimation for a binomial proportion.” *Statistical Science*, vol. 16, no. 2, 2001, pp. 101–133.

[5] Agresti, A. and Coull, B. A. ”Approximate is better than 'exact' for interval estimation of binomial proportions.” *The American Statistician*, vol. 52, no. 2, 1998, pp. 119–126.

[6] Clopper, C. J. and Pearson, E. S. ”The use of confidence or fiducial limits illustrated in the case of the binomial.” *Biometrika*, vol. 26, no. 4, 1934, pp. 404–413.

[7] The javai project family: probabilistic testing frameworks. Source code and documentation available at https://javai.org and https://github.com/javai-org.

[8] Christensen, K., Varshosaz, M., and Pardo, R. ”ProbTest: Unit Testing for Probabilistic Programs.” In: Bianculli, D. and Gómez-Martínez, E. (Eds.): SEFM 2025, LNCS 16192, pp. 91–109. Springer, 2026.

[9] Younes, H. L. S. and Simmons, R. G. ”Statistical probabilistic model checking with a focus on time-bounded properties.” *Information and Computation*, vol. 204, no. 9, 2006, pp. 1368–1409.

[10] Legay, A., Delahaye, B., and Bensalem, S. ”Statistical model checking: an overview.” In: *Runtime Verification*, LNCS 6418, Springer, 2010, pp. 122–135.

[11] Sen, K., Viswanathan, M., and Agha, G. ”Statistical model checking of black-box probabilistic systems.” In: *Computer Aided Verification*, LNCS 3114, Springer, 2004, pp. 202–215.

[12] Aichernig, B. K. and Tappler, M. ”Probabilistic black-box reachability checking.” In: *Runtime Verification*, LNCS 10548, Springer, 2017, pp. 50–67.

[13] Musa, J. D. ”Operational profiles in software-reliability engineering.” *IEEE Software*, vol. 10, no. 2, 1993, pp. 14–32.

[14] Frankl, P. G., Hamlet, D., Littlewood, B., and Strigini, L. ”Evaluating testing methods by delivered reliability.” *IEEE Transactions on Software Engineering*, vol. 24, no. 8, 1998, pp. 586–601.

[15] Hamlet, D. and Taylor, R. ”Partition testing does not inspire confidence.” *IEEE Transactions on Software Engineering*, vol. 16, no. 12, 1990, pp. 1402–1411.

[16] Agha, G. and Palmskog, K. ”A survey of statistical model checking.” *ACM Transactions on Modeling and Computer Simulation*, vol. 28, no. 1, 2018, pp. 1–39.