Statistical Companion Document

Formal Statistical Foundations for the javai Methodology

Contents

Statistical Companion Document **3**

Formal Statistical Foundations for the javai Methodology 3

Introduction: The Assumption of Certainty in Software Testing 3

The LLM Inflection Point 3

Two Dimensions of Stochasticity 4

The Need for Statistical Rigor 5

The javai Project Family 5

Document Purpose and Audience 6

Reference Scenarios 6

Scenario A: Compliance Testing with a Payment Processing API 6

Scenario B: Regression Testing with an LLM-Based Customer Service . 6

Two Testing Scenarios 7

Compliance Testing 7

Regression Testing 7

Mathematical Structure is Identical 8

1. Statistical Model 8

1.1 Bernoulli Trial Framework 8

1.2 Binomial Aggregation 8

1.3 Assumptions and Limitations 9

2. Baseline Estimation (Experiment Phase) 10

2.1 Point Estimation 10

2.2 Standard Error 10

2.3 Confidence Intervals 11

2.4 Sample Size Determination 13

3. Threshold Derivation for Regression Testing 13

3.1 The Problem 13

3.2 One-Sided Hypothesis Testing Framework 13

3.3 One-Sided Lower Confidence Bound 14

3.4 Threshold Calculation (Wilson Score Lower Bound) 14

3.5 Reference Table: Wilson Score Lower Bounds 15

3.6 Testing Against a Given Threshold (Compliance) 15

4. The Perfect Baseline Problem ($\hat{p} = 1$) 16

4.1 Problem Statement 16

4.2 Interpretation of 100% Observed Success 17

4.3 The Wilson Lower Bound Solution 17
4.4 Extended Example: Highly Reliable API 18
4.5 Theoretical Note: Beta-Binomial Alternative 19
5. Test Execution and Interpretation 19
5.1 Decision Rule 19
5.2 Type I and Type II Errors 19
5.3 Statistical Power 20
5.4 Sample Size for Desired Power 20
5.5 Sample Size for SLA Verification 21
5.6 The Role of Minimum Detectable Effect 22
5.7 Test Intent: VERIFICATION vs SMOKE 22
6. The Three Operational Approaches: Mathematical Formulation 24
6.1 Approach 1: Sample-Size-First (Cost-Driven) 25
6.2 Approach 2: Confidence-First (Risk-Driven) 25
6.3 Approach 3: Direct Threshold (Threshold-First) 26
7. Reporting and Interpretation 27
7.1 Transparent Statistics Output 27
7.2 Confidence Statement 28
7.3 Multiple Testing Considerations 28
7.4 Threshold Provenance 29
8. Assumptions and Validity Conditions 30
8.1 When Normal Approximation is Valid 30
8.2 Independence Violations 30
8.3 Non-Stationarity 30
8.4 Guardrails for Assumption Validity 32
9. Summary of Key Formulas 35
Estimation 35
Wald Confidence Interval (two-sided) 35
Wilson Score Interval 35
One-Sided Lower Bound (for threshold derivation) 35
Wilson Lower Bound (for $\hat{p} = 1$) 35
Rule of Three (quick approximation for zero failures) 35
Sample Size for Precision 36
Sample Size for Power 36
Empirical Percentile (nearest-rank) 36
Latency Threshold Derivation (conservative percentile tolerance) . . . 36
10. Transparent Statistics Mode 36
10.1 Purpose 36
10.2 Output Structure 36
10.3 Example Output: Regression Paradigm 37
10.4 Example Output: Compliance Paradigm 38
10.5 Mathematical Notation 39
10.6 Validation by Statisticians 39
11. Statistical Discipline Through Design 39
11.1 The Problem with ”Just Run More Tests” 39
11.2 How the Methodology Encourages Good Practice 40
11.3 Explicit Over Implicit 40
11.4 Warnings as Statistical Honesty 40

11.5 Stationarity as a Managed Condition 41
11.6 The Audit Trail 41
11.7 Summary: Statistics in Service of Engineering 42
12. Latency: Empirical Percentile Analysis 42
12.1 The Statistical Challenge of Latency 42
12.2 Empirical Percentile Estimation 43
12.3 Latency Assertions 44
12.4 Threshold Derivation from Baselines 45
12.5 Sample Size Requirements for Percentile Estimation 46
12.6 Enforcement Modes 48
12.7 Relationship to Pass-Rate Testing 48
References 48

Statistical Companion Document

Formal Statistical Foundations for the javai Methodology

Introduction: The Assumption of Certainty in Software Testing

For decades, the dominant paradigm in software testing has rested on an implicit assumption: **systems under test behave with certainty**. Given the same input, a correctly functioning system produces the same output. This assumption has shaped our tools, our practices, and our intuitions about what ”correct” means.

Under this paradigm, tests produce **binary outcomes**—pass or fail—and a single failure is definitive evidence of a defect. Test frameworks report success as a count of green checkmarks, and any red mark warrants investigation. The entire edifice of continuous integration, test-driven development, and quality gates is built on this foundation.

Of course, uncertainty has always existed. Network timeouts, race conditions, clock skew, and resource contention introduce variability into even carefully designed systems — both in *what* they produce and in *how long* they take to produce it. But the traditional response has been to **treat uncertainty as a defect to eliminate**—a source of ”flaky tests” to be fixed, mocked away, or suppressed. The goal was to restore the deterministic ideal.

The LLM Inflection Point

As long as undesirable glitches or crashes occur sufficiently rarely, and their consequences remain relatively harmless, the sloppy handling of indeterminism may go unnoticed, be ignored, or be downplayed. Such phenomena are effectively normalized.

However, this posture cannot possibly work in the context of Large Language Models, in which uncertainty is not a bug to fix but an **intrinsic characteristic of the technology**. Even with identical input parameters, a model will likely produce different outputs. This is the system working as designed. The same input genuinely produces different outputs, and the distribution of those outputs *is* the system's behavior.

This represents a qualitative shift in the testing challenge:

Traditional Testing	Testing Under Uncertainty
Accidental (bugs)	Intentional (sampling)
To be eliminated	To be characterized
Failure is binary	Failure is probabilistic
Single test is definitive	Single test is a sample

The average software developer has encountered uncertainty, but typically as an annoyance to work around. LLMs bring it into the foreground as a **permanent feature** of the systems we build. A customer service chatbot that ”usually” gives correct answers is not buggy—it's behaving exactly as its underlying model does.

Two Dimensions of Stochasticity

The uncertainty introduced by non-deterministic systems manifests along two independent dimensions:

1. **Functional stochasticity** — whether the system produces a correct result. Given identical input, an LLM may generate valid JSON in 95 out of 100 invocations and malformed output in the remaining 5. The *correctness* of the output is a random variable.

2. **Temporal stochasticity** — how long the system takes to respond. Even among successful invocations, response times vary substantially. A service that averages 200ms may spike to 2 seconds on any given call. Latency is not a fixed property of the system; it is a *distribution*.

These dimensions are independent. A fast response can be incorrect; a slow response can be correct. A service can meet its reliability target while routinely breaching its latency SLA, or vice versa. Treating either dimension in isolation misses half the picture.

Traditional testing addresses neither dimension statistically. A single successful response says nothing about the success *rate*; a single fast response says nothing about the *tail latency* that affects the worst-served users. Both require repeated observation and distributional reasoning — the same shift from binary to probabilistic thinking.

The javai methodology addresses both dimensions within a unified framework. Pass-rate assertions (Sections 1–5) model functional stochasticity as a binomial process. Latency assertions (Section 12) characterise temporal stochasticity through empirical percentile distributions. Both are evaluated independently and both must pass for the test to pass, reflecting the reality that a system must be both correct *and* responsive.

Other observable properties of stochastic services — memory consumption, token usage, cost per call — also vary across invocations. The javai methodology focuses on functional correctness and latency because these two dimensions have the most direct impact on end users and developers: does the service work, and is it fast enough? Resource consumption is orthogonal and, while worth monitoring, is typically managed through infrastructure tooling rather than test assertions.

The Need for Statistical Rigor

This shift demands a corresponding evolution in testing methodology. We cannot simply run a test once and declare success. Nor can we run it many times and hope for the best. We need:

1. **Principled sample sizes**: How many trials provide sufficient evidence?
2. **Quantified uncertainty**: What confidence can we place in our conclusions?
3. **Controlled error rates**: How do we balance false positives against false negatives?
4. **Reproducible thresholds**: How do we set pass/fail criteria that are defensible?

These are fundamentally **statistical questions**, and they deserve statistical answers. The javai project family exists to bring the rigor of hypothesis testing, confidence intervals, and power analysis to the practical problem of testing systems characterized by uncertainty.

This document provides the formal foundations for that approach. And while the hot topic of LLMs brings the reality of uncertainty into the foreground, the principles developed here are applicable to any system that exhibits uncertain behavior.

The javai Project Family

The statistical methodology described in this document is implemented across multiple language-native frameworks:

Framework	Language	Repository
punit	Java	JUnit 5 extension — the reference implementation
feotest	Rust	Idiomatic Rust, not a port
baseltest	Python	(planned)

Each framework implements the same statistical core independently, in its own language and idiom. The mathematics described in this document is the shared specification; the implementations are independent.

Conformance Verification The javai-R project generates language-agnostic reference datasets using R — the gold standard for statistical computing. Each framework verifies that its statistics engine produces results matching the R-generated reference data within stated tolerances. This ensures that all implementations deliver identical statistical verdicts, regardless of language.

Document Purpose and Audience

This document provides a rigorous statistical treatment of the methods employed by the javai methodology for probabilistic testing of systems characterized by uncertainty. It is intended for:

- **Professional statisticians** validating the mathematical foundations
- **Quality engineers** with statistical training designing test strategies
- **Auditors** who need to verify that testing methodology is sound
- **Technical leads** establishing organizational testing standards
- **Framework implementors** building javai-compatible statistics engines in new languages

---

Reference Scenarios

The methodology accommodates two distinct testing scenarios. Both are equally valid; they differ only in where the threshold comes from.

Scenario A: Compliance Testing with a Payment Processing API

Application: A third-party payment processing API with a contractual uptime guarantee.

Service Level Agreement (SLA) Requirement: The contract states: ”The API shall successfully process transactions at least 99.5% of the time.”

Success Criterion: A trial is successful if the API returns a successful response (HTTP 2xx with valid transaction confirmation).

Key parameters:

- $p_{\text{SLA}} = 0.995$ (given by contract)
- No baseline experiment required
- Statistical question: ”Does the system meet its contractual obligation?”

Scenario B: Regression Testing with an LLM-Based Customer Service

Application: A customer service system that uses a Large Language Model (LLM) to generate structured JSON responses from natural language queries.

Use Case: Given a query (e.g., ”What's the shipping status for order #12345?”), the system should return valid JSON with the required fields (`orderId`, `status`, `estimatedDelivery`).

Success Criterion: A trial is successful if:

1. The response is syntactically valid JSON
2. The response contains all required fields
3. The field values are semantically appropriate (not hallucinated)

Key parameters:

- $\hat{p}_{\text{exp}} = 0.951$ (measured from 1000-sample experiment)
- Threshold derived from empirical baseline
- Statistical question: ”Has the system degraded from its measured baseline?”

Two Testing Scenarios

The methodology supports two distinct testing scenarios. They share the same statistical foundations but differ in where the threshold comes from and how results are interpreted.

Scenario	Threshold Source	Statistical Question
Compliance Testing	Contract, SLA, SLO, policy	”Does the system meet the mandat
Regression Testing	Empirical estimate from experiment	”Has the system degraded from its

Compliance Testing

The threshold is a **normative claim**—a business or contractual requirement:

- The threshold p_{SLA} is given, not estimated.
- No baseline experiment is required.
- The test verifies conformance to an external standard.
- **Verification vs. Smoke**: For an evidential claim (VERIFICATION intent), the sample size must be sufficient to support the threshold. If not, the developer must explicitly declare the intent as SMOKE.
- Failure means: ”Evidence that the system does not meet the requirement” (in VERIFICATION) or ”Potential regression detected” (in SMOKE).

When to use: Third-party APIs with SLAs, internal services with SLOs, compliance requirements, contractual obligations.

Regression Testing

The threshold is an **empirical estimate** derived from measurement:

- The threshold is derived from experimental data $(\hat{p}_{\text{exp}}, n_{\text{exp}})$
- A baseline experiment (a ”measure experiment”) is required first
- The test detects regression from the measured baseline
- Failure means: ”Evidence that the system has degraded”

When to use: LLM-based systems, ML models, any system where developers need to discover acceptable performance through experimentation.

Mathematical Structure is Identical

Both paradigms use the same hypothesis test structure:

$$H_0 : p \geq p_{\text{threshold}} \quad \text{(acceptable)}$$

$$H_1 : p < p_{\text{threshold}} \quad \text{(unacceptable)}$$

The difference is in:

1. **Source** of $p_{\text{threshold}}$ (given vs. derived)
2. **Interpretation** of results (SLA violation vs. regression)
3. **Prerequisite steps** (none vs. MEASURE experiment)

---

1. Statistical Model

1.1 Bernoulli Trial Framework

Each invocation of the use case is modeled as an independent Bernoulli trial:

$$X_i \sim \text{Bernoulli}(p)$$

where:

- $X_i \in \{0, 1\}$ is the outcome of the *i*-th trial (1 = success, 0 = failure)
- $p \in [0, 1]$ is the true (unknown) success probability
- Trials are assumed independent and identically distributed (i.i.d.)

1.2 Binomial Aggregation

For *n* independent trials, the total number of successes follows a binomial distribution:

$$K = \sum_{i=1}^{n} X_i \sim \text{Binomial}(n, p)$$

The sample proportion $\hat{p} = K/n$ is an unbiased estimator of *p*:

$$E[\hat{p}] = p, \quad \text{Var}(\hat{p}) = \frac{p(1-p)}{n}$$

1.3 Assumptions and Limitations

The Bernoulli model assumes:

1. **Independence**: Each trial is independent. In practice, this may be violated if, for example:
   - The LLM provider implements request-level caching
   - Rate limiting causes correlated delays
   - Model state persists across requests (generally not the case for stateless APIs)

2. **Stationarity**: The success probability p is constant across trials. This may be violated if, for example:
   - A baseline is created at a time when the system was under load, but a test performed later, while the system IS NOT under load. Such a test will likely miss a drop in performance of the system.
   - Contextual factors differ between baseline creation and test execution (time of day, day of week, deployment region, etc.)

   The javai methodology addresses stationarity through two complementary mechanisms:

   - **Covariates** (see Section 8.4.1): Explicit declaration and tracking of contextual factors that may influence success rates, with warnings when baseline and test contexts differ.

   - **Baseline expiration** (see Section 8.4.2): Time-based validity tracking that alerts operators when baselines may no longer represent current system behavior.

   These features do not *guarantee* stationarity—that is impossible in practice—but they make non-stationarity **visible and auditable** rather than silently undermining inference.

3. **Binary outcomes**: Each trial has exactly two outcomes. Complex quality metrics may require more sophisticated models.

Developer Responsibility for Trial Independence While the framework provides the statistical machinery, **developers share responsibility** for ensuring that the independence assumption holds in practice. Many LLM service providers offer features that can introduce correlation between trials:

Feature	Risk to Independence	Mitigation
Cached system prompts	Cached prompts may produce more consistent outputs	Disable cachin
Conversation context	Prior exchanges influence subsequent responses	Use fresh sess
Request batching	Batched requests may share internal state	Submit reques
Seed parameters	Fixed seeds produce identical outputs	Use random or

Example: An LLM provider's ”cache system prompt” feature improves latency by reusing parsed prompts. While beneficial in production, this can reduce output variance during testing. The developer should either:

- Disable the cache via configuration during experiments
- Document that the measured success rate reflects cached behavior
- Understand that the true variance may be higher than observed

No framework can detect or correct for these effects—they require domain knowledge and deliberate configuration choices. When trial independence is uncertain, developers should consider:

- Running sensitivity analyses with different configurations
- Documenting assumptions in test annotations or comments
- Increasing sample sizes to account for potential correlation (see Section 8.2)

Recommendation: For most LLM-based systems accessed via stateless APIs, the independence assumption is reasonable when caching and context features are disabled. Monitor for temporal drift in long-running experiments.

---

2. Baseline Estimation (Experiment Phase)

2.1 Point Estimation

Given n experimental trials with k successes, the maximum likelihood estimator (MLE) for p is:

$$\hat{p} = \frac{k}{n}$$

Example: In our JSON generation use case, an experiment with n = 1000 trials yields k = 951 successes.

$$\hat{p} = \frac{951}{1000} = 0.951$$

2.2 Standard Error

The standard error of $\hat{p}$ quantifies the precision of the estimate:

$$\text{SE}(\hat{p}) = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

Example:

$$\text{SE} = \sqrt{\frac{0.951 \times 0.049}{1000}} = \sqrt{0.0000466} \approx 0.00683$$

2.3 Confidence Intervals

The javai methodology uses the Wilson score interval exclusively for all confidence interval calculations. This section documents the Wilson method and, for completeness, includes the Wald (normal approximation) method that statisticians may encounter in textbooks.

2.3.1 Wilson Score Interval (the javai Method)

The Wilson score interval is the sole method for confidence interval construction across all javai framework implementations. It has superior coverage properties across all conditions encountered in probabilistic testing:

- Correct coverage for all sample sizes (including $n < 40$)
- Valid for proportions near 0 or 1 (including $\hat{p} = 1$)
- Never produces bounds outside [0, 1]

The Wilson interval endpoints are:

$$\frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

Example (95% CI for $\hat{p} = 0.951$, $n = 1000$):

$$\text{Lower} = \frac{0.951 + \frac{1.96^2}{2000} - 1.96\sqrt{\frac{0.951 \times 0.049}{1000} + \frac{1.96^2}{4000000}}}{1 + \frac{1.96^2}{1000}} \approx 0.937$$

$$\text{Upper} \approx 0.963$$

Why Wilson Exclusively? The javai methodology uses Wilson for all calculations because:

1. **Wilson is never worse**: For large samples and moderate proportions, Wilson produces results nearly identical to the Wald approximation. There is no penalty for using Wilson universally.
2. **Wilson avoids pathologies**: For small samples, extreme proportions, or perfect baselines ($\hat{p} = 1$), alternative methods produce incorrect or degenerate results. Wilson handles all cases correctly.
3. **Consistency**: A single method ensures results are always comparable across tests. No edge cases where method switching affects verdicts.
4. **LLM testing reality**: High success rates ($p > 0.85$) are common in probabilistic testing of LLM-based systems. This is precisely where Wilson provides the most benefit.

What This Means for Developers

- Developers do not need to choose a method or configure thresholds for method switching
- Statistical calculations are consistent across all sample sizes
- The formulas shown above are what every javai framework uses—always

Conformance Verification The javai-R project generates reference Wilson score interval values using R's `qnorm` function. Each framework implementation verifies its Wilson computation matches the R-generated reference data. See `inst/cases/wilson_ci.json` and `inst/cases/wilson_lower.json` in the javai-R repository.

2.3.2 Wald Interval (For Completeness) For completeness, this section documents the Wald interval (normal approximation), which statisticians will encounter in many textbooks. **The javai methodology does not use this method**, but understanding it helps explain why Wilson is preferred.

For large *n*, by the Central Limit Theorem:

$$\hat{p} \overset{a}{\sim} N\left(p, \frac{p(1-p)}{n}\right)$$

The $(1-\alpha)$ Wald confidence interval is:

$$\hat{p} \pm z_{\alpha/2} \cdot \text{SE}(\hat{p})$$

where $z_{\alpha/2}$ is the $(1-\alpha/2)$ quantile of the standard normal distribution.

Example (95% CI, $z_{0.025} = 1.96$):

$$0.951 \pm 1.96 \times 0.00683 = [0.938, 0.964]$$

Why the javai Methodology Does Not Use Wald Statistics textbooks often present guidelines for when Wald is acceptable:

Condition	Textbook Guidance
$n \geq 40$ and $0.1 \leq \hat{p} \leq 0.9$	Wald acceptable
$n \geq 20$ and ($\hat{p} < 0.1$ or $\hat{p} > 0.9$)	Wilson preferred
$n < 20$	Wilson strongly recommended
$n < 10$	Wilson required; Wald inappropriate

Rather than implement conditional method selection, the methodology uses Wilson universally. This simplifies implementation while providing correct results in all cases—including the edge cases where Wald fails.

2.4 Sample Size Determination

To achieve a desired margin of error *e* with confidence $(1-\alpha)$, the required sample size is approximately:

$$n = \frac{z_{\alpha/2}^2 \cdot \hat{p}(1-\hat{p})}{e^2}$$

Example: To estimate *p* ≈ 0.95 with ±2% margin at 95% confidence:

$$n = \frac{1.96^2 \times 0.95 \times 0.05}{0.02^2} = \frac{0.1825}{0.0004} \approx 456$$

Target Precision (95% CI)	Required *n* (for *p* ≈ 0.95)
±5%	73
±3%	203
±2%	456
±1%	1,825

---

3. Threshold Derivation for Regression Testing

3.1 The Problem

The experiment established $\hat{p}_{\text{exp}} = 0.951$ from $n_{\text{exp}} = 1000$ samples. For cost reasons, regression tests will use $n_{\text{test}} = 100$ samples.

Question: What threshold $p_{\text{threshold}}$ should the regression test use?

Naive approach: Use $p_{\text{threshold}} = 0.951$.

Problem with naive approach: With only 100 samples, the standard error is:

$$\text{SE}_{\text{test}} = \sqrt{\frac{0.951 \times 0.049}{100}} \approx 0.0216$$

Even if the true *p* equals the experimental rate, observed rates will vary. At $\pm 2\sigma$, we'd expect observations between 0.908 and 0.994. Using 0.951 as the threshold would cause frequent false positives.

3.2 One-Sided Hypothesis Testing Framework

Both compliance and regression testing use a **one-sided hypothesis test**:

$$H_0 : p \geq p_{\text{threshold}} \quad \text{(acceptable)}$$

$$H_1 : p < p_{\text{threshold}} \quad \text{(unacceptable)}$$

The difference lies in how $p_{\text{threshold}}$ is determined and interpreted:

Paradigm	Threshold	H_0 Interpretation	H_1 Interpretation
Compliance	p_{SLA} (given)	System meets requirement	System violates SLA
Regression	Derived from $\hat{p}_{\text{exp}}$	No degradation from baseline	Regression has occurred

We seek a decision rule that:

- Controls the Type I error rate (false positive) at level α
- Maximizes power to detect true violations/degradation

3.3 One-Sided Lower Confidence Bound

The $(1-\alpha)$ one-sided lower confidence bound is:

$$p_{\text{lower}} = \hat{p} - z_\alpha \cdot \text{SE}$$

Note: For one-sided bounds, we use z_α (not $z_{\alpha/2}$).

Confidence Level	z_α	Interpretation
90%	1.282	10% false positive rate
95%	1.645	5% false positive rate
99%	2.326	1% false positive rate

3.4 Threshold Calculation (Wilson Score Lower Bound)

Thresholds are derived using the Wilson one-sided lower bound, consistent with the exclusive use of Wilson for all statistical calculations (see Section 2.3.1).

Given experimental results $(\hat{p}_{\text{exp}}, n_{\text{exp}})$ and test configuration $(n_{\text{test}}, \alpha)$, the threshold is the one-sided Wilson lower bound:

$$p_{\text{threshold}} = \frac{\hat{p} + \frac{z^2}{2n} - z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

where $z = z_\alpha$ is the one-sided critical value.

Example ($\hat{p}_{\text{exp}} = 0.951$, $n_{\text{test}} = 100$, $\alpha = 0.05$, $z = 1.645$):

$$p_{\text{threshold}} = \frac{0.951 + \frac{2.706}{200} - 1.645\sqrt{\frac{0.0466}{100} + \frac{2.706}{40000}}}{1 + \frac{2.706}{100}}$$

$$= \frac{0.951 + 0.0135 - 1.645 \times 0.0218}{1.027} = \frac{0.9286}{1.027} \approx 0.904$$

Interpretation: A 100-sample test with threshold 0.904 will have a 5% false positive rate if the true success probability equals the experimental rate.

Conformance Verification The javai-R project generates reference threshold derivation values. See `inst/cases/threshold_derivation.json` in the javai-R repository.

3.5 Reference Table: Wilson Score Lower Bounds

For baseline success-rate estimate $\hat{p} = 0.951$:

Test Samples	95% Lower Bound	99% Lower Bound
50	0.874	0.826
100	0.902	0.874
200	0.919	0.902
500	0.933	0.923

Observation: Smaller test samples require lower bounds (and hence lower thresholds) to maintain the same false positive rate.

3.6 Testing Against a Given Threshold (Compliance)

For compliance testing, the threshold is **given**, not derived:

$$p_{\text{threshold}} = p_{\text{SLA}}$$

There is no experimental baseline—the threshold comes directly from a contract, SLA, SLO, or organizational policy.

Example: A payment processing SLA states "99.5% transaction success rate."

- $p_{\text{SLA}} = 0.995$ (given by contract)
- No $\hat{p}_{\text{exp}}$ to estimate
- Test directly verifies: does $p \geq 0.995$?

Why This Changes the Statistics In regression testing, we derive a *lowered* threshold from $\hat{p}_{\text{exp}}$ to account for sampling variance. In compliance testing, the threshold is fixed—but this creates a different challenge:

The False Positive Problem: If a test uses $p_{\text{threshold}} = p_{\text{SLA}} = 0.995$ and the true system rate is exactly 0.995, then approximately 50% of tests will fail purely due to sampling variance. This is not a bug—it's statistics.

Solutions:

1. **Sample-Size-First**: Accept a fixed sample count and let the framework compute what confidence this achieves against the SLA threshold.
2. **Confidence-First**: Specify required confidence and `minDetectableEffect`. The framework computes the sample size needed to detect violations of at least that magnitude.
3. **Direct Threshold**: Use the exact SLA threshold and accept the statistical consequences (high false positive rate near threshold boundary).

Reference Table: Sample Sizes for SLA Verification To verify $p \geq p_{\text{SLA}}$ with 95% confidence and 80% power:

p_{SLA}	Min Detectable Effect (δ)	Required Samples
0.95	0.05 (detect drop to 90%)	150
0.95	0.02 (detect drop to 93%)	822
0.99	0.02 (detect drop to 97%)	236
0.99	0.01 (detect drop to 98%)	793
0.999	0.005 (detect drop to 99.4%)	548
0.999	0.001 (detect drop to 99.8%)	8,031

Key insight: Higher SLAs and smaller detectable effects require dramatically more samples. This is why `minDetectableEffect` is essential for the confidence-first approach—without it, the question ”how many samples to verify 99.9%?” has no finite answer.

---

4. The Perfect Baseline Problem ($\hat{p} = 1$)

4.1 Problem Statement

A critical pathology arises when the baseline experiment observes **zero failures**:

$$k = n \implies \hat{p} = 1$$

This commonly occurs when testing highly reliable systems (e.g., well-established third-party APIs) where failures are rare but not impossible.

Example: An experiment with $n = 1000$ trials against a payment gateway API yields $k = 1000$ successes.

Why this matters: With $\hat{p} = 1$, naive threshold derivation would set $p_{\text{threshold}} = 1$, meaning any single failure causes test failure—regardless of sample size or confidence level. This is statistically unsound.

The javai solution: The Wilson score method (Section 2.3.1) handles this case correctly. This is another reason for using Wilson exclusively—it remains valid at the boundaries where other methods fail.

4.2 Interpretation of 100% Observed Success

An observed rate of $\hat{p} = 1$ from n trials does **not** mean $p = 1$. Rather, it provides evidence that:

$$P(p \geq p_{\text{lower}} \mid k = n, n) = 1 - \alpha$$

where p_{lower} is derived using methods that remain valid at the boundary.

The Rule of Three (quick approximation): With n trials and zero failures, we can be approximately 95% confident that:

$$p \geq 1 - \frac{3}{n}$$

Baseline Samples	95% Lower Bound (Rule of Three)
100	0.970
300	0.990
1000	0.997
3000	0.999

(Side note: this assumes conditions are stable and runs are independent.)

4.3 The Wilson Lower Bound Solution

The javai methodology resolves this pathology using the **Wilson score lower bound**, which remains well-defined when $\hat{p} = 1$.

Key implementation requirement: Baselines store (k, n), not merely $\hat{p}$. The observed rate can be computed ($\hat{p} = k/n$), but raw counts are essential for proper statistical treatment of boundary cases.

Procedure:

1. Compute the one-sided Wilson lower bound p_0 from the baseline (k, n)
2. Use p_0 (not $\hat{p}$) as the effective baseline for threshold derivation
3. Apply the standard threshold formula using p_0

4.3.1 Wilson Lower Bound Formula

The general Wilson one-sided lower bound is:

$$p_{\text{lower}} = \frac{\hat{p} + \frac{z^2}{2n} - z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

When $\hat{p} = 1$, this simplifies to:

$$p_{\text{lower}} = \frac{n}{n + z^2}$$

4.3.2 Worked Example **Baseline**: $n = 1000$ trials, $k = 1000$ successes (100% observed)

Step 1: Compute Wilson lower bound (95% one-sided, $z = 1.645$):

$$p_0 = \frac{1000}{1000 + 2.706} = \frac{1000}{1002.706} \approx 0.9973$$

Step 2: Derive test threshold for $n_{\text{test}} = 100$:

$$\text{SE}_{\text{test}} = \sqrt{\frac{0.9973 \times 0.0027}{100}} \approx 0.0052$$

$$p_{\text{threshold}} = 0.9973 - 1.645 \times 0.0052 = 0.9973 - 0.0086 \approx 0.989$$

Interpretation: For 100-sample tests, the threshold is 0.989 (at most 1 failure permitted). Compare this to the naive approach, which would set threshold = 1.0 (zero failures permitted), producing an undefined false positive rate.

4.3.3 Reference Table: Thresholds for 100% Baselines

Baseline n	p_0 (Wilson 95%)	Test $n = 50$ threshold	Test $n = 100$ threshold
100	0.9737	0.936	0.947
300	0.9911	0.969	0.976
1000	0.9973	0.985	0.989
3000	0.9991	0.992	0.994

4.4 Extended Example: Highly Reliable API

Scenario: Testing a payment gateway integration.

Baseline experiment: 2000 transactions, 0 failures ($\hat{p} = 1$).

Goal: Configure regression tests with 95% confidence.

Calculation:

1. Wilson lower bound: $p_0 = \frac{2000}{2000+2.706} = 0.9986$
2. For 100-sample test:
   - $\text{SE} = \sqrt{0.9986 \times 0.0014/100} = 0.0037$
   - $p_{\text{threshold}} = 0.9986 - 1.645 \times 0.0037 = 0.993$
3. For 50-sample test:

- $\text{SE} = \sqrt{0.9986 \times 0.0014/50} = 0.0053$
- $p_{\text{threshold}} = 0.9986 - 1.645 \times 0.0053 = 0.990$

Result: Even for this highly reliable system, the methodology produces statistically principled thresholds with valid confidence level interpretation.

4.5 Theoretical Note: Beta-Binomial Alternative

For statisticians reviewing this methodology: the **Beta-Binomial posterior predictive** approach offers a theoretically superior treatment that fully propagates baseline uncertainty and produces integer thresholds. However, the javai methodology uses the Wilson bound because:

- It requires no prior specification
- It is simpler to implement and audit
- It produces results that are practically equivalent for typical sample sizes
- It remains within the frequentist paradigm familiar to most practitioners

Organizations with strong Bayesian infrastructure or specific requirements for integer thresholds may wish to implement their own threshold derivation using the posterior predictive:

$$K_t \mid k, n \sim \text{BetaBinomial}(n_t, a + k, b + n - k)$$

where (a, b) are prior hyperparameters (Jeffreys: $a = b = 0.5$).

See Gelman et al. (2013) for a complete treatment.

---

5. Test Execution and Interpretation

5.1 Decision Rule

Given a test with n_{test} samples and threshold $p_{\text{threshold}}$:

1. Execute use case n_{test} times
2. Count successes k_{test}
3. Compute observed rate $\hat{p}_{\text{test}} = k_{\text{test}}/n_{\text{test}}$
4. Decision:
    - If $\hat{p}_{\text{test}} \geq p_{\text{threshold}}$: **PASS** (no evidence of degradation)
    - If $\hat{p}_{\text{test}} < p_{\text{threshold}}$: **FAIL** (threshold not met—evidence of degradation)

5.2 Type I and Type II Errors

	True state: No degradation	True state: Degradation
Test passes	Correct (True Negative)	Type II Error (False Negative)
Test fails	Type I Error (False Positive)	Correct (True Positive)

- **Type I error rate** (α): Controlled by threshold derivation. If threshold is set at $(1-\alpha)$ confidence, then $P(\text{False Positive}) = \alpha$.
- **Type II error rate** (β): Depends on:
  - True effect size (how much degradation occurred)
  - Sample size
  - Threshold

5.3 Statistical Power

Power is the probability of correctly detecting degradation when it exists:

$$\text{Power} = 1 - \beta = P(\text{Reject } H_0 | H_1 \text{ true})$$

For a one-sided test detecting a shift from p_0 to p_1 (where $p_1 < p_0$):

$$\text{Power} = \Phi\left(\frac{p_0 - p_1 - z_\alpha \cdot \text{SE}_0}{\text{SE}_1}\right)$$

where:

- $\text{SE}_0 = \sqrt{p_0(1-p_0)/n}$
- $\text{SE}_1 = \sqrt{p_1(1-p_1)/n}$
- Φ is the standard normal CDF

Example: Detecting a drop from $p_0 = 0.95$ to $p_1 = 0.90$ with $n = 100$ at $\alpha = 0.05$:

$$\text{SE}_0 = \sqrt{0.95 \times 0.05/100} = 0.0218$$

$$\text{SE}_1 = \sqrt{0.90 \times 0.10/100} = 0.0300$$

$$\text{Power} = \Phi\left(\frac{0.95 - 0.90 - 1.645 \times 0.0218}{0.0300}\right) = \Phi\left(\frac{0.0141}{0.0300}\right) = \Phi(0.47) \approx 0.68$$

With 100 samples, we have only 68% power to detect a 5-percentage-point degradation.

Conformance Verification The javai-R project generates reference power analysis values. See `inst/cases/power_analysis.json` in the javai-R repository.

5.4 Sample Size for Desired Power

To achieve power $(1-\beta)$ for detecting effect size $\delta = p_0 - p_1$:

$$n = \left(\frac{z_\alpha\sqrt{p_0(1-p_0)} + z_\beta\sqrt{p_1(1-p_1)}}{\delta}\right)^2$$

Example: 80% power to detect 5% drop from 95% to 90% at $\alpha = 0.05$:

$$n = \left(\frac{1.645 \times 0.218 + 0.842 \times 0.300}{0.05}\right)^2 = \left(\frac{0.359 + 0.253}{0.05}\right)^2 = (12.24)^2 \approx 150$$

Effect Size	Power 80%	Power 90%	Power 95%
5% (95%→90%)	150	200	250
10% (95%→85%)	40	55	70
3% (95%→92%)	410	550	700

5.5 Sample Size for SLA Verification

When verifying an SLA threshold p_{SLA} (rather than detecting degradation from a baseline), the sample size formula adapts:

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

Where:

- p_{SLA} is the given SLA threshold (e.g., 0.995)
- δ is the minimum detectable effect—the smallest violation worth detecting
- α is the significance level (Type I error rate)
- β is the Type II error rate ($1-\beta$ is power)

Example: Verify 99.5% SLA with 95% confidence, 80% power, detecting drops of 1% or more:

- $p_{\text{SLA}} = 0.995$, $\delta = 0.01$, $\alpha = 0.05$, $\beta = 0.20$

$$n = \left(\frac{1.645 \times \sqrt{0.995 \times 0.005} + 0.842 \times \sqrt{0.985 \times 0.015}}{0.01}\right)^2$$

$$= \left(\frac{1.645 \times 0.0705 + 0.842 \times 0.1215}{0.01}\right)^2 = \left(\frac{0.116 + 0.102}{0.01}\right)^2 = (21.8)^2 \approx 477$$

Reference Table: Sample Sizes for High SLAs

SLA Threshold	Effect Size (δ)	95% Confidence, 80% Power
99.0%	2% (detect ≤97%)	236

SLA Threshold	Effect Size (δ)	95% Confidence, 80% Power
99.0%	1% (detect ≤98%)	793
99.5%	1% (detect ≤98.5%)	477
99.5%	0.5% (detect ≤99%)	1,597
99.9%	0.5% (detect ≤99.4%)	548
99.9%	0.1% (detect ≤99.8%)	8,031

Key insight: Verifying high SLAs with small detectable effects requires substantial sample sizes. A 99.9% SLA with 0.1% detection requires about 8,031 samples.

5.6 The Role of Minimum Detectable Effect

The `minDetectableEffect` parameter answers a critical question:

> ”What's the smallest degradation worth detecting?”

Without this parameter, the question ”how many samples to verify $p \geq 0.999$?” has no finite answer. Here's why:

Mathematical necessity: To detect an arbitrarily small degradation from 99.9% to 99.89% would require millions of samples. To detect a drop to 99.899% requires even more. For infinitesimal effects, infinite samples are required.

Practical reality: No organization needs to detect every possible degradation. There's always a threshold below which degradation doesn't matter operationally:

System Type	Typical δ	Rationale
E-commerce checkout	1-2%	1% drop = significant revenue loss
Internal tooling	5-10%	User productivity impact
Safety-critical systems	0.1-0.5%	Regulatory requirements
High-frequency trading	0.01%	Financial impact per transaction

When `minDetectableEffect` is required:

In the **Confidence-First approach**, developers must specify `minDetectableEffect` for the framework to compute the required sample size. This applies to both compliance and regression testing.

Without `minDetectableEffect`, no framework can compute a finite sample size and will use the default sample count instead.

5.7 Test Intent: VERIFICATION vs SMOKE

The javai methodology distinguishes between two epistemic intentions when running a probabilistic test. The choice of intent governs whether the framework enforces a statistical feasibility gate and how verdicts are framed.

Intent	Purpose	Feasibility gate	Verdict
VERIFICATION	Evidential — produce statistically defensible verdict	Enforced	Full co
SMOKE	Sentinel — detect gross regressions cheaply	Bypassed	Soften

The default intent is **VERIFICATION**. Developers may opt into SMOKE when appropriate.

5.7.1 The Feasibility Gate (VERIFICATION only) Before any samples execute, the framework checks whether the configured sample size is **sufficient** for the test to produce a meaningful PASS verdict. The criterion uses the Wilson score one-sided lower bound (the same method used throughout for confidence bounds — see Section 4.3.1):

> For a perfect observation ($k = n$), the Wilson lower bound is $n/(n + z^2)$. The sample is feasible if this bound $\geq p_0$.

Solving for the minimum sample size:

$$N_{\text{min}} = \left\lceil \frac{p_0 \cdot z^2}{1 - p_0} \right\rceil$$

where $z = \Phi^{-1}(1 - \alpha)$ and $\alpha = 1 -$ confidence.

Reference table: N_{min} **at default confidence (0.95,** $\alpha = 0.05$**,** $z \approx 1.645$**)**

Target (p_0)	N_{min}	Interpretation
0.50	3	Almost any sample size suffices
0.80	11	Low bar
0.90	25	Moderate
0.95	52	Common threshold — needs at least 52 samples
0.99	268	High reliability — needs substantial samples
0.999	2,704	Very high reliability
0.9999	27,058	Extreme reliability — impractical for most test suites

What happens when infeasible: A VERIFICATION test with $N < N_{\text{min}}$ fails immediately with a configuration error. The failure message includes:

- The configured sample size and target
- The minimum required sample size
- A suggestion to use SMOKE intent if the test is a sentinel check

This failure is **distinct from a SUT failure** — it indicates a configuration problem, not a system defect. It is non-ignorable in CI.

Conformance Verification The javai-R project generates reference feasibility values. See `inst/cases/feasibility.json` in the javai-R repository.

5.7.2 SMOKE Intent: When and Why SMOKE tests are appropriate when:

- **Quick feedback** is more valuable than statistical defensibility (e.g. pre-commit hooks, nightly canaries)
- **The sample budget is fixed** by cost constraints and falls below N_{min}
- **The test is exploratory** — developers are still discovering the system's performance characteristics
- **The target is aspirational** — the threshold expresses an SLA that will later be verified with a properly sized test

5.7.3 FAIL Asymmetry for SMOKE Tests A key statistical insight governs how SMOKE verdicts should be interpreted:

> A **FAIL** verdict from an undersized test is directionally reliable — the observed rate fell below the threshold, and the direction of the evidence is clear even if the magnitude is uncertain. A **PASS** verdict from an undersized test provides weak evidence — the confidence interval is too wide to exclude values below the threshold.

In other words: small samples can reliably detect gross failures but cannot provide strong evidence of compliance. Framework output reflects this asymmetry:

- **SMOKE PASS**: Softened language — ”The observed rate is consistent with the target.”
- **SMOKE FAIL**: Still clear — ”The observed rate is inconsistent with the target.”
- **VERIFICATION PASS**: Full compliance language — ”The system meets its SLA requirement.”

5.7.4 Intent-Aware Caveats When a SMOKE test runs against a normative threshold (SLA, SLO, or POLICY):

Condition	Caveat
$N < N_{\text{min}}$	”Sample not sized for verification ($N = x$, need y). A PASS is a directional signal, not
$N \geq N_{\text{min}}$	”Sample is sized for verification. Consider setting `intent = VERIFICATION` for evident

These caveats appear in both summary and verbose output modes.

---

6. The Three Operational Approaches: Mathematical Formulation

The three operational approaches apply to **both** paradigms. The key difference is the source of the threshold:

Paradigm	Threshold Source	Symbol Used
Compliance	Given by contract/policy	p_{SLA}
Regression	Derived from experimental basis	$\hat{p}_{\text{exp}}$

Below, we present each approach with formulations for both paradigms.

6.1 Approach 1: Sample-Size-First (Cost-Driven)

Fix the sample count based on budget constraints; compute the implied threshold or confidence.

Regression Formulation **Given**: n_{test}, α, experimental basis $(\hat{p}_{\text{exp}}, n_{\text{exp}})$

Compute: $p_{\text{threshold}}$

$$p_{\text{threshold}} = \hat{p}_{\text{exp}} - z_\alpha \sqrt{\frac{\hat{p}_{\text{exp}}(1-\hat{p}_{\text{exp}})}{n_{\text{test}}}}$$

Trade-off: Fixed cost; confidence is controlled; threshold (sensitivity) is determined.

Compliance Formulation **Given**: n_{test}, p_{SLA}

Compute: Implied confidence level

Since the threshold is fixed ($p_{\text{threshold}} = p_{\text{SLA}}$), we compute what confidence the test achieves:

$$z = \frac{\hat{p}_{\text{observed}} - p_{\text{SLA}}}{\sqrt{p_{\text{SLA}}(1-p_{\text{SLA}})/n_{\text{test}}}}$$

The achieved confidence is $1 - \Phi(-z)$ where Φ is the standard normal CDF.

Trade-off: Fixed cost and threshold; confidence is determined by the data.

6.2 Approach 2: Confidence-First (Risk-Driven)

Fix the confidence and power requirements; compute the required sample size.

Regression Formulation **Given**: α, desired power $(1-\beta)$, minimum detectable effect δ, experimental basis $(\hat{p}_{\text{exp}}, n_{\text{exp}})$

Compute: n_{test}

$$n_{\text{test}} = \left(\frac{z_\alpha \sqrt{\hat{p}_{\text{exp}}(1-\hat{p}_{\text{exp}})} + z_\beta \sqrt{(\hat{p}_{\text{exp}} - \delta)(1 - \hat{p}_{\text{exp}} + \delta)}}{\delta} \right)^2$$

Trade-off: Fixed confidence and detection capability; cost (sample size) is determined.

Compliance Formulation **Given**: α, desired power $(1-\beta)$, minimum detectable effect δ, SLA threshold p_{SLA}

Compute: n_{test}

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

Example: Verify 99.5% SLA, detecting drops of 1% or more with 95% confidence and 80% power:

- $p_{\text{SLA}} = 0.995$, $\delta = 0.01$, $z_\alpha = 1.645$, $z_\beta = 0.842$

$$n = \left(\frac{1.645 \times 0.0705 + 0.842 \times 0.1215}{0.01} \right)^2 \approx 477$$

Trade-off: Fixed confidence and detection capability; cost (sample size) is determined.

Critical: The `minDetectableEffect` (δ) is essential. Without it, verifying any SLA requires infinite samples.

6.3 Approach 3: Direct Threshold (Threshold-First)

Use an explicit threshold directly; compute the implied confidence.

Regression Formulation **Given**: n_{test}, $p_{\text{threshold}}$ (often = $\hat{p}_{\text{exp}}$), experimental basis

Compute: Implied α

$$z_\alpha = \frac{\hat{p}_{\text{exp}} - p_{\text{threshold}}}{\sqrt{\hat{p}_{\text{exp}}(1-\hat{p}_{\text{exp}})/n_{\text{test}}}}$$

$$\alpha = 1 - \Phi(z_\alpha)$$

Example: Using threshold = 0.951 with $n = 100$:

$$z_\alpha = \frac{0.951 - 0.951}{0.0216} = 0$$

$$\alpha = 1 - \Phi(0) = 0.50$$

Interpretation: A 50% false positive rate—half of all test runs will fail even with no degradation.

Compliance Formulation **Given**: n_{test}, p_{SLA}

Compute: Implied α (using observed data)

The test directly uses the SLA threshold. After running, we compute:

$$z = \frac{\hat{p}_{\text{observed}} - p_{\text{SLA}}}{\sqrt{p_{\text{SLA}}(1 - p_{\text{SLA}})/n_{\text{test}}}}$$

The implied Type I error rate depends on where the true system rate lies relative to p_{SLA}:

- If true $p = p_{\text{SLA}}$: approximately 50% of tests will fail (the threshold boundary problem)
- If true $p > p_{\text{SLA}}$: fewer false positives
- If true $p < p_{\text{SLA}}$: the test correctly detects the violation

Trade-off: Fixed cost and threshold; confidence (reliability of verdicts) is determined—often poorly when the true rate is near the threshold.

When to use: Learning the trade-offs, strict compliance requirements where the SLA threshold is non-negotiable.

---

7. Reporting and Interpretation

7.1 Transparent Statistics Output

When transparent statistics mode is enabled, the framework outputs a structured report containing the following sections:

Section	Contents	P
HYPOTHESIS TEST	H_0, H_1, test type	F
OBSERVED DATA	Sample size n, successes k, observed rate $\hat{p}$	R
BASELINE REFERENCE	Source, empirical basis or SLA threshold, derivation method	T
STATISTICAL INFERENCE	Standard error, confidence interval, z-score, p-value	F
VERDICT	Result (PASS/FAIL), plain English interpretation, caveats	H
THRESHOLD PROVENANCE	Threshold origin, contract reference (if specified)	A

Key Metrics in the Report

Metric	Formula/Value	Interpretation
Sample size	n	Number of trials executed
Successes	k	Number of passing trials
Observed rate	$\hat{p} = k/n$	Point estimate from test
Standard error	$\text{SE} = \sqrt{\hat{p}(1-\hat{p})/n}$	Precision of the estimate
Confidence interval	Wilson score bounds	Range of plausible true values

Metric	Formula/Value	Interpretation
Z-score	$z = (\hat{p} - p_{\text{threshold}})/\text{SE}_0$	Standardized deviation from threshold
p-value	$P(Z > z)$	Probability of observing this or worse

Example Output For a test observing 87/100 successes against threshold 0.904:

```
OBSERVED DATA
  Sample size (n):     100
  Successes (k):       87
  Observed rate (p̂):   0.870

STATISTICAL INFERENCE
  Standard error:      SE = √(p̂(1-p̂)/n) = √(0.87 × 0.13 / 100) = 0.0336
  95% Confidence interval: [0.790, 0.926]

  Test statistic:      z = (p̂ - π₀) / √(π₀(1-π₀)/n)
                       z = (0.87 - 0.904) / √(0.904 × 0.096 / 100)
                       z = -1.15

  p-value:             P(Z < -1.15) = 0.125

VERDICT
  Result:              FAIL
 Interpretation:     The observed success rate of 87% is below the threshold
                       of 90.4%. Under threshold-comparison semantics, this
                       test fails verification.
```

See Section 10 for complete example outputs including both compliance and regression paradigms.

Conformance Verification The javai-R project generates reference verdict evaluation values. See `inst/cases/verdict.json` in the javai-R repository.

7.2 Confidence Statement

Every failure report should include a plain-language confidence statement:

> ”This test was configured with 95% confidence. There is a 5% probability that this failure is due to sampling variance rather than actual system degradation. The observed p-value of < 0.0001 indicates the result is highly unlikely under the null hypothesis of no degradation.”

7.3 Multiple Testing Considerations

When running multiple probabilistic tests:

- **Per-test error rate**: Each test has false positive rate α

- **Family-wise error rate**: Probability of at least one false positive increases with number of tests

For m independent tests at level α:

$$P(\text{at least one false positive}) = 1 - (1 - \alpha)^m$$

Number of tests	Per-test α = 0.05	Per-test α = 0.01
5	22.6%	4.9%
10	40.1%	9.6%
20	64.2%	18.2%

Mitigation options:

- Bonferroni correction: Use $\alpha' = \alpha/m$
- Benjamini-Hochberg: Control false discovery rate
- Accept inflated family-wise rate with documentation

7.4 Threshold Provenance

For auditability and traceability, the methodology records the **source** of the threshold through two attributes:

Attribute	Purpose	Example Values
`thresholdOrigin`	Category of threshold origin	`SLA`, `SLO`, `POLICY`, `EMPIRICAL`
`contractRef`	Human-readable reference to source document	`"SLA v2.1 §4.3"`, `"Policy-2`

Target Source Values

Value	Meaning	Hypothesis Framing
`SLA`	Service Level Agreement (contractual)	”System meets SLA requirement”
`SLO`	Service Level Objective (internal target)	”System meets SLO target”
`POLICY`	Organizational policy or standard	”System meets policy requirement”
`EMPIRICAL`	Derived from baseline experiment	”No degradation from baseline”
`UNSPECIFIED`	Not specified (default)	”Success rate meets threshold”

Impact on Hypothesis Formulation The `thresholdOrigin` influences how the framework frames the hypothesis test in detailed reports:

`thresholdOrigin`	H_0 (Null Hypothesis)	H_1 (Alternative)
`SLA`	System meets SLA requirement	System violates SLA
`SLO`	System meets SLO target	System falls short of SLO

`thresholdOrigin`	H_0 (Null Hypothesis)	H_1 (Alternative)
`POLICY`	System meets policy requirement	System violates policy
`EMPIRICAL`	No degradation from baseline	Degradation from baseline
`UNSPECIFIED`	Success rate meets threshold	Success rate below threshold

This adaptation ensures that verdicts are framed in the appropriate business context, making reports immediately understandable to stakeholders.

---

8. Assumptions and Validity Conditions

8.1 When Normal Approximation is Valid

The normal approximation to the binomial is adequate when:

$$n \cdot p \geq 5 \quad \text{and} \quad n \cdot (1 - p) \geq 5$$

More conservatively (for confidence intervals):

$$n \cdot p \geq 10 \quad \text{and} \quad n \cdot (1 - p) \geq 10$$

For p = 0.95:

- Need $n \geq 200$ for conservative criterion
- Wilson interval recommended for $n < 200$

8.2 Independence Violations

If trials are not independent, the effective sample size is reduced:

$$n_{\text{eff}} = \frac{n}{1 + (n - 1)\rho}$$

where ρ is the intraclass correlation.

Detection: Run autocorrelation analysis on trial outcomes. Significant lag-1 autocorrelation suggests dependence.

Mitigation: Increase sample size or introduce delays between trials.

8.3 Non-Stationarity

Non-stationarity—when the success probability p is not constant—is perhaps the most insidious threat to probabilistic testing. Unlike independence violations, which can sometimes be detected through autocorrelation, non-stationarity may be invisible in aggregate statistics while fundamentally invalidating comparisons.

8.3.1 Forms of Non-Stationarity

Form	Example	Detection Difficulty
Within-experiment drift	Model updates during a long MEASURE run	Moderate (time-seri
Between-experiment drift	System changes between baseline and test	Hard (requires exte
Contextual variation	Different behavior on weekdays vs weekends	Easy (if factors are
Gradual degradation	Slow performance decay over months	Hard (no single dete

8.3.2 Statistical Consequences If p changes during the experiment:

- Point estimate $\hat{p}$ reflects time-averaged behavior, not current behavior
- Confidence intervals may understate true uncertainty
- Threshold derivations may be based on stale data
- Verdicts may systematically mislead in one direction

If p differs between baseline and test:

- The comparison is between **different populations**
- The hypothesis test answers the wrong question
- Type I and Type II error rates are no longer controlled
- Verdicts are statistically meaningless (though they appear valid)

8.3.3 Why This Is Hard Non-stationarity is difficult because:

1. **It's often invisible in aggregate data**: A 95% pass rate could arise from stable 95% behavior, or from 99% for half the samples and 91% for the other half.
2. **It can occur between experiments**: The system that generated the baseline may literally not exist anymore (different model version, different infrastructure).
3. **It can be caused by external factors**: Changes to dependencies, APIs, or infrastructure that the developer doesn't control or even know about.
4. **The statistical machinery assumes it away**: All the formulas in this document assume p is constant. When it isn't, the formulas still produce numbers—they're just wrong.

8.3.4 The javai Approach No framework can guarantee stationarity. Instead, the javai methodology provides **guardrails** that:

1. **Make context explicit**: Covariate declarations force developers to think about what factors might matter.
2. **Make drift visible**: Covariate non-conformance and expiration warnings surface potential violations.
3. **Preserve auditability**: Baseline provenance ensures the conditions of inference are always documented.

4. **Qualify rather than suppress**: Warnings accompany verdicts rather than replacing them.

See Section 8.4 for detailed descriptions of these guardrails.

8.3.5 Developer Responsibilities The framework provides tools; developers must use them wisely:

Responsibility	How the Framework Helps	What Developers M
Identify relevant factors	Standard covariates for common cases	Declare covariates
Track contextual changes	Automatic covariate resolution and matching	Ensure custom cov
Recognize baseline staleness	Expiration warnings	Set appropriate ex
Investigate warnings	Clear warning messages with specifics	Don't ignore non-c
Refresh stale baselines	Prominent expiration alerts	Run measure exper

8.4 Guardrails for Assumption Validity

The statistical validity of probabilistic testing depends on the assumptions outlined in Section 1.3. While no framework can guarantee these assumptions hold, the javai methodology provides **guardrails**—features that surface violations, qualify results, and encourage practices that preserve statistical validity.

These guardrails embody a key principle: **statistical honesty over silent convenience**. Rather than producing clean verdicts that hide uncertainty, the framework makes the conditions of inference explicit and auditable.

8.4.1 Covariate-Aware Baseline Matching **The problem**: A baseline represents the empirical behavior of a system under specific conditions. If a probabilistic test runs under different conditions—different time of day, different deployment region, different feature flags—the comparison may be invalid. The samples are drawn from **different populations**.

This is a violation of the **stationarity assumption**: the success probability p is not constant between baseline creation and test execution.

Example: A customer service LLM performs differently during peak hours (high load, queue delays) versus off-peak hours. A baseline measured at 2 AM may not represent behavior at 2 PM.

The javai solution: Developers declare **covariates**—exogenous factors that may influence success rates. The exact syntax varies by framework, but the concept is universal across all implementations.

This declaration is an explicit statement: ”These factors may affect performance. Track them.”

How it works:

1. During measure experiments, the framework records the covariate values as part of the baseline specification.

2. During probabilistic tests, the framework resolves the current covariate values and compares them against the baseline.
3. If values differ (non-conformance), the framework issues a **warning** that qualifies the verdict.

Statistical interpretation: Non-conformance does not change the pass/fail verdict. Instead, it **qualifies** the inference:

> ”Under the assumption that success rates are comparable across these conditions, the test passes. However, this assumption may not hold—the baseline was created on a weekday, but the test is running on a weekend.”

This transforms a hidden assumption into an explicit, auditable caveat.

Why this matters:

Without Covariates	With Covariates
Silent assumption that conditions don't matter	Explicit declaration of relevant conditions
Population mismatch is invisible	Population mismatch triggers warning
False confidence in verdicts	Qualified confidence with documented caveats
Statistical validity unknowable	Statistical validity auditable

8.4.2 Baseline Expiration **The problem**: Systems change over time. Dependencies update, models are retrained, infrastructure drifts. A baseline from six months ago may no longer represent current behavior—even if all declared covariates match.

This is **temporal non-stationarity**: the success probability p changes over calendar time in ways that cannot be captured by discrete covariates.

The javai solution: Developers declare a **validity period** for baselines.

This declaration is an explicit statement: ”I believe this baseline remains representative for N days.”

How it works:

1. The expiration value is recorded in the baseline specification along with the experiment end timestamp.
2. During probabilistic tests, the framework computes whether the baseline has expired.
3. As expiration approaches, the framework issues **graduated warnings**:

Time Remaining	Warning Level	Message
> 25% of validity period	None	—
≤ 25%	Informational	”Baseline expires soon”
≤ 10%	Warning	”Baseline expiring imminently”
Expired	Prominent	”BASELINE EXPIRED”

Statistical interpretation: Expiration does not change the pass/fail verdict. Instead, it signals:

> ”This baseline is old. The system may have changed in ways not captured by covariate tracking. Interpret results with appropriate caution.”

Complementary to covariates: Covariates catch **known, observable** context changes (weekday vs weekend, region). Expiration catches **unknown, gradual** drift (model updates, dependency changes, infrastructure evolution).

Guardrail	What It Catches	Mechanism
Covariates	Known context mismatch	Explicit factor declaration and matching
Expiration	Unknown temporal drift	Calendar-based validity period

Together, they provide **defense in depth** against non-stationarity.

8.4.3 Baseline Provenance **The problem**: A statistical verdict is only meaningful if its empirical foundation is known. ”The test passed” means little without knowing: against what baseline? Under what conditions? With what caveats?

The javai solution: Every test verdict includes explicit **baseline provenance**:

```
BASELINE REFERENCE
  File:         ShoppingUseCase-ax43-dsf2.yaml
  Generated:    2026-01-10 14:45 UTC
  Samples:      1000
  Observed rate: 95.1%
  Covariates:  day_of_week=WEEKDAY, time_of_day=08:00/4h, region=EU_CORE
  Expiration:  2026-02-09 (27 days remaining)
```

This ensures that **no inference result is ever detached from its empirical foundation**.

Why this matters for statistical validity:

- Auditors can verify that comparisons are appropriate
- Operators can investigate unexpected results by examining baseline conditions
- Historical analysis can account for which baseline was in effect when
- Reproducibility is supported by explicit documentation

8.4.4 Explicit Warnings Over Silent Failures A consistent design principle across the javai methodology's guardrails:

> **Warnings qualify verdicts; they do not suppress them.**

Condition	Verdict Impact	Warning
Covariate non-conformance	None	Yes
Expired baseline	None	Yes
Multiple suitable baselines	None	Yes

This principle reflects a statistical philosophy:

1. **The developer asked a statistical question** ("Does my system meet this threshold?"). The framework answers that question.
2. **The answer may have caveats** ("...but the baseline is old" or "...but the conditions differ"). The framework surfaces those caveats.
3. **The decision about whether to trust the answer remains with the human**. The framework provides information, not absolution.

This approach preserves statistical honesty without creating operational paralysis. Tests don't mysteriously skip or fail due to metadata issues—they run, and their limitations are documented.

---

9. Summary of Key Formulas

Estimation

$$\hat{p} = \frac{k}{n}, \quad \mathrm{SE}(\hat{p}) = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

Wald Confidence Interval (two-sided)

$$\hat{p} \pm z_{\alpha/2} \cdot \mathrm{SE}(\hat{p})$$

Wilson Score Interval

$$\frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

One-Sided Lower Bound (for threshold derivation)

$$p_{\text{threshold}} = \hat{p} - z_\alpha \cdot \mathrm{SE}$$

Wilson Lower Bound (for $\hat{p} = 1$)

$$p_{\text{lower}} = \frac{n}{n + z^2}$$

Rule of Three (quick approximation for zero failures)

$$p \geq 1 - \frac{3}{n} \quad \text{(95\% confidence)}$$

Sample Size for Precision

$$n = \frac{z_{\alpha/2}^2 \cdot p(1-p)}{e^2}$$

Sample Size for Power

$$n = \left(\frac{z_\alpha \sqrt{p_0(1-p_0)} + z_\beta \sqrt{p_1(1-p_1)}}{p_0 - p_1} \right)^2$$

Empirical Percentile (nearest-rank)

$$Q(p) = t_{(\lceil p \cdot n_s \rceil)}, \quad t_{(1)} \leq \cdots \leq t_{(n_s)}$$

Latency Threshold Derivation (conservative percentile tolerance)

$$\tau_j = \max\left(Q_{\text{baseline}}(p_j),\ \left\lceil Q_{\text{baseline}}(p_j) + z_\alpha \cdot \frac{s}{\sqrt{n_s}} \right\rceil \right)$$

10. Transparent Statistics Mode

10.1 Purpose

Transparent Statistics Mode exposes the complete statistical reasoning behind every test verdict. This feature serves:

- **Auditors**: Documented proof that testing methodology is statistically sound
- **Stakeholders**: Evidence that reliability claims are justified
- **Educators**: Teaching material for understanding probabilistic testing
- **Regulators**: Compliance documentation for AI system validation

10.2 Output Structure

When enabled, each test produces a structured explanation containing:

Section	Content	Statistical Pu
Hypothesis Test	H_0, H_1, test type	Frames the s
Observed Data	n, k, $\hat{p}$	Raw observa
Threshold Reference	Source (SLA/empirical), provenance, threshold derivation	Traces thres
Statistical Inference	SE, CI, z-score, p-value	Full calculat
Verdict	Pass/fail with interpretation and caveats	Human-read

The **Threshold Reference** section adapts based on the testing paradigm:

Paradigm	Content Displayed
Compliance	Threshold origin (SLA/SLO/POLICY), contract reference, normative threshold
Regression	Spec file, empirical basis (samples, rate), threshold derivation method

10.3 Example Output: Regression Paradigm

```
═══════════════════════════════════════════════════════════════════════════
STATISTICAL ANALYSIS: shouldReturnValidJson
═══════════════════════════════════════════════════════════════════════════

HYPOTHESIS TEST
  H₀ (null):       True success rate π ≥ 0.85 (no degradation from baseline)
  H₁ (alternative): True success rate π < 0.85 (degradation has occurred)
  Test type:        One-sided binomial proportion test

OBSERVED DATA
  Sample size (n):     100
  Successes (k):       87
  Observed rate (p̂):   0.870

THRESHOLD REFERENCE
  Source:              ShoppingUseCase.yaml (generated 2026-01-10)
  Empirical basis:     1000 samples, 872 successes (87.2%)
  Threshold derivation: Lower bound of 95% CI = 85.1%, rounded to 85%

STATISTICAL INFERENCE
  Standard error:      SE = √(p̂(1-p̂)/n) = √(0.87 × 0.13 / 100) = 0.0336
  95% Confidence interval: [0.804, 0.936]

  Test statistic:      z = (p̂ - π₀) / √(π₀(1-π₀)/n)
                       z = (0.87 - 0.85) / √(0.85 × 0.15 / 100)
                       z = 0.56

  p-value:             P(Z > 0.56) = 0.288

VERDICT
  Result:              PASS
  Interpretation:      The observed success rate of 87% is consistent with
                      the baseline expectation of 87.2%. The 95% confidence
                     interval [80.4%, 93.6%] contains the threshold of 85%.

  Caveat:              With n=100 samples, we can detect a drop from 87% to
                       below 85% with approximately 50% power. For higher
                       sensitivity, consider increasing sample size.

═══════════════════════════════════════════════════════════════════════════
```

10.4 Example Output: Compliance Paradigm

```
═══════════════════════════════════════════════════════════════════════════════
STATISTICAL ANALYSIS: verifyPaymentGatewaySla
═══════════════════════════════════════════════════════════════════════════════

HYPOTHESIS TEST
  H₀ (null):       True success rate π ≥ 0.995 (system meets SLA requirement)
  H₁ (alternative): True success rate π < 0.995 (system violates SLA)
  Test type:        One-sided binomial proportion test

OBSERVED DATA
  Sample size (n):     500
  Successes (k):       496
  Observed rate (p̂):   0.992

THRESHOLD REFERENCE
  Threshold origin:    SLA
  Contract ref:        Payment Gateway SLA v3.2 §5.1.2
  Threshold origin:    Normative claim from external contract
  Threshold value:     0.995 (99.5%)

STATISTICAL INFERENCE
  Standard error:      SE = √(p̂(1-p̂)/n) = √(0.992 × 0.008 / 500) = 0.0040
  95% Confidence interval: [0.980, 0.997]

  Test statistic:      z = (p̂ - π₀) / √(π₀(1-π₀)/n)
                       z = (0.992 - 0.995) / √(0.995 × 0.005 / 500)
                       z = -0.95

  p-value:             P(Z < -0.95) = 0.171

VERDICT
  Result:              PASS (but borderline)
  Interpretation:      The observed success rate of 99.2% is below the SLA
                       threshold of 99.5%, but the evidence is insufficient
                       to conclude a violation (p = 0.171 > 0.05).

                       The 95% confidence interval [98.4%, 100%] includes
                      the SLA threshold, so we cannot reject the hypothesis
                       that the system meets its contractual obligation.

  Caveat:              The observed rate (99.2%) is below the SLA threshold
                     (99.5%). The threshold is met at this confidence level,
                       but this warrants monitoring. Consider increasing
                       sample size for more conclusive evidence.

═══════════════════════════════════════════════════════════════════════════════
```

Key differences in compliance output:

- Hypothesis framing uses ”meets SLA requirement” / ”violates SLA”
- Threshold Reference shows provenance (`thresholdOrigin`, `contractRef`) instead of empirical basis
- Verdict interpretation is framed in terms of contractual compliance

10.5 Mathematical Notation

The output uses proper mathematical symbols where terminal capabilities allow:

Concept	Unicode	ASCII Fallback
Sample proportion	$\hat{p}$ (p̂)	p-hat
Population proportion	π	pi
Null hypothesis	H_0	H0
Alternative hypothesis	H_1	H1
Less than or equal	$\leq$	<=
Greater than or equal	$\geq$	>=
Square root	$\sqrt{}$	sqrt
Approximately	$\approx$	~=
Alpha (significance)	α	alpha

10.6 Validation by Statisticians

The transparent output enables statisticians to verify:

1. **Hypothesis formulation**: Is the one-sided test appropriate?
2. **Threshold derivation**: Was the Wilson lower bound correctly applied?
3. **Confidence interval**: Is the Wilson score interval used correctly?
4. **Sample size adequacy**: Are the caveats about power appropriate?
5. **Interpretation**: Does the plain-English summary accurately reflect the statistics?

---

11. Statistical Discipline Through Design

The javai methodology is designed not just to *perform* statistical tests, but to *encourage* statistical discipline. This section summarizes how the methodology's features embody principles of good statistical practice.

11.1 The Problem with ”Just Run More Tests”

A naive approach to non-deterministic testing is: ”Run the test many times and see if it mostly passes.” This approach fails because:

- **No principled sample size**: How many is ”many”? Without power analysis, sample sizes are guesses.

- **No controlled error rates**: What confidence do we have in verdicts? Without hypothesis testing, confidence is undefined.
- **No threshold derivation**: What pass rate is acceptable? Without baselines, thresholds are arbitrary.
- **No assumption checking**: Are comparisons valid? Without guardrails, violations are invisible.

The javai methodology addresses each of these failures with specific features.

11.2 How the Methodology Encourages Good Practice

Statistical Principle	Common Violation	javai Guardrail
Principled sample sizes	Arbitrary numbers (10, 100, 1000)	Power analysis, confid
Controlled error rates	Unknown false positive rates	Threshold derivation v
Empirically-grounded thresholds	Hardcoded guesses	MEASURE experimen
Assumption validity	Silent violations	Covariate tracking, ex
Reproducibility	Undocumented conditions	Baseline provenance,
Transparency	Black-box verdicts	Transparent statistics

11.3 Explicit Over Implicit

The javai methodology favors explicitness:

Aspect	Implicit (Hidden)	Explicit (javai)
Threshold origin	Hardcoded number	`thresholdOrigin`, `contractRef`
Baseline conditions	Unmarked file	Covariate profile in spec
Baseline age	Check file timestamp	Expiration period, expiration status
Comparison validity	Assumed	Non-conformance warnings
Statistical reasoning	Hidden in code	Transparent statistics output

This explicitness serves multiple audiences:

- **Developers**: Understand what they're testing and why
- **Reviewers**: Verify that test configuration is appropriate
- **Auditors**: Confirm that methodology is sound
- **Operators**: Investigate unexpected results with full context

11.4 Warnings as Statistical Honesty

The javai methodology's warning system reflects a commitment to statistical honesty:

> The purpose of statistical analysis is not to produce clean answers, but to quantify uncertainty and surface limitations.

When the framework issues a warning about covariate non-conformance or baseline expiration, it is saying:

> ”I performed the calculation you requested. Here is the answer. However, you should know that the conditions underlying this calculation may not be ideal. Here's specifically what concerns me.”

This is more useful than either:

- **Silent acceptance**: Producing a verdict without surfacing limitations
- **Silent rejection**: Refusing to produce a verdict due to imperfect conditions

In practice, conditions are rarely perfect. Statistical discipline means quantifying and documenting imperfection, not pretending it doesn't exist.

11.5 Stationarity as a Managed Condition

Traditional statistical frameworks treat stationarity as an **assumption**—something that must be true for the analysis to be valid, but that is typically asserted rather than verified.

The javai methodology treats stationarity as a **managed condition**:

Traditional Approach	javai Approach
Assume stationarity holds	Declare relevant factors (covariates)
Hope baselines are still valid	Set explicit validity periods (expiration)
Silently violate assumptions	Surface violations as warnings
Binary: valid or invalid	Graduated: valid, cautionary, expired

This shift—from hidden assumption to managed condition—is at the heart of the javai approach to statistical validity.

11.6 The Audit Trail

Every test verdict can be traced to its foundations:

```
Verdict: PASS (97/100 = 97% ≥ min pass rate 90.4%)
    ↓
Threshold: 90.4% (derived from baseline at 95% confidence)
    ↓
Baseline: ShoppingUseCase-ax43-dsf2.yaml
    ↓
Empirical Basis: 951/1000 = 95.1% (measured 2026-01-10)
    ↓
Conditions: weekday=Mo-Fr, time=09:03-09:25, region=EU
    ↓
Validity: 27 days remaining (expires 2026-02-09)
    ↓
Conformance: ⚠ time_of_day non-conforming (test ran at 14:30)
```

This audit trail enables:

- **Post-hoc investigation**: Why did this test fail last Tuesday?

- **Compliance documentation**: Proof that testing methodology is sound
- **Historical analysis**: How have pass rates changed over time?
- **Debugging**: Is this a real regression or a baseline mismatch?

11.7 Summary: Statistics in Service of Engineering

The javai methodology's statistical machinery serves a practical goal: **reliable, reproducible verdicts about non-deterministic systems**.

The features described in this section—covariate tracking, baseline expiration, provenance, transparent statistics—are not academic exercises. They address real problems that arise when testing systems like LLMs:

- ”Why does this test pass on Tuesday and fail on Saturday?”
- ”Is this failure real or just variance?”
- ”Can I trust a baseline from three months ago?”
- ”What assumptions am I making, and are they valid?”

By surfacing these questions—and providing frameworks for answering them—the javai methodology brings statistical discipline to probabilistic testing without requiring developers to be statisticians.

---

12. Latency: Empirical Percentile Analysis

12.1 The Statistical Challenge of Latency

Pass-rate testing models functional outcomes as Bernoulli trials drawn from a binomial distribution (Section 1). Latency presents a fundamentally different statistical challenge. Service latency distributions are typically:

- **Right-skewed**: A long right tail caused by cache misses, garbage collection pauses, network retransmission, or cold starts
- **Multimodal**: Distinct modes corresponding to fast paths (cached) and slow paths (database lookup, remote API call)
- **Heavy-tailed**: Extreme outliers that are orders of magnitude larger than the median

These characteristics violate the assumptions of parametric models such as the normal distribution. A system with a 200ms mean and 50ms standard deviation could have a p99 of 350ms (near-normal) or 2000ms (heavy-tailed) — the summary statistics cannot distinguish the two.

The javai approach: Rather than fitting a parametric distribution to latency data, the methodology uses **non-parametric empirical percentiles** exclusively. This distribution-free approach makes no assumptions about the shape of the latency distribution — it characterises the distribution directly from the observed order statistics.

12.2 Empirical Percentile Estimation

12.2.1 Population Definition Not all samples contribute to the latency distribution. The methodology defines the **latency population** as the distribution of **successful-response latency**:

$$\mathcal{L} = \{t_i : X_i = 1\}$$

Equivalently, the latency estimand is the conditional distribution:

$$T \mid X = 1$$

where t_i is the wall-clock execution time of the i-th trial and X_i is its Bernoulli outcome. Only successful samples ($X_i = 1$) are included.

Rationale: Failed samples produce execution times that are not directly comparable with successful response times. A fast failure (immediate validation rejection, $t \approx 0$) and a slow failure (timeout at $t = 30{,}000$ms) both reflect error paths, not the latency of successful operation. Including them would contaminate the distribution and produce percentile estimates that describe neither the successful nor the failed population.

Important semantic note: This is a **conditional latency distribution**, not the unconditional user-experienced response-time distribution over all attempts. Pass rate and latency therefore describe two complementary dimensions: the probability of success, and the latency distribution given success.

Let $n_s = |\mathcal{L}|$ denote the number of successful samples and let $t_{(1)} \leq t_{(2)} \leq \cdots \leq t_{(n_s)}$ be the order statistics of the successful latencies.

12.2.2 Nearest-Rank Interpolation The javai methodology computes percentiles using the **nearest-rank method**. For a percentile $p \in (0, 1]$ with n_s sorted observations:

$$\text{index}(p) = \lceil p \cdot n_s \rceil - 1 \quad \text{(clamped to } [0,\, n_s - 1])$$

$$Q(p) = t_{(\text{index}(p)+1)}$$

where $t_{(k)}$ denotes the k-th order statistic.

Worked example: For $n_s = 200$ successful samples:

Percentile	p	$\lceil p \cdot 200 \rceil - 1$	Order statistic
p50	0.50	99	$t_{(100)}$
p90	0.90	179	$t_{(180)}$
p95	0.95	189	$t_{(190)}$
p99	0.99	197	$t_{(198)}$

Why nearest-rank? Linear interpolation methods (e.g., Type 7 in R) produce fractional percentile estimates. For latency thresholds expressed in integer milliseconds and compared against SLA values, the discrete nearest-rank method produces integer-valued estimates that align naturally with how thresholds are specified and reported.

12.2.3 Summary Statistics In addition to percentiles, the framework computes:

- **Mean**: $\bar{t} = \frac{1}{n_s} \sum_{i=1}^{n_s} t_i$
- **Sample standard deviation**: $s = \sqrt{\frac{1}{n_s-1} \sum_{i=1}^{n_s} (t_i - \bar{t})^2}$
- **Maximum**: $t_{(n_s)}$

The mean and standard deviation are used in threshold derivation (Section 12.4). The percentiles and maximum characterise the distribution shape.

12.3 Latency Assertions

12.3.1 The Assertion Model A latency assertion specifies a set of percentile constraints:

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

where p_j is a percentile level (one of $\{0.50, 0.90, 0.95, 0.99\}$) and τ_j is the corresponding threshold in milliseconds.

For each constraint, the assertion evaluates:

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

The overall latency assertion passes if and only if all individual constraints pass:

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

12.3.2 Combined Verdict Pass-rate and latency are independent quality dimensions. The overall test verdict requires both to pass:

$$\text{PASS}_{\text{test}} = \text{PASS}_{\text{rate}} \wedge \text{PASS}_{\text{latency}}$$

This reflects the operational reality that 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 SLA just as surely as one that returns incorrect results.

12.3.3 Threshold Sources Latency thresholds can originate from two sources:

Source	Symbol	How specified	Statistical question
Explicit	τ_j (given)	Annotation or configuration	”Does the system meet the dec
Baseline-derived	$\hat{\tau}_j$ (estimated)	Automatic from spec	”Has latency degraded from th

This mirrors the compliance vs. regression dichotomy for pass-rate thresholds (Section 3). Explicit thresholds are normative claims; baseline-derived thresholds are empirical estimates with a statistical margin.

12.4 Threshold Derivation from Baselines

12.4.1 The Problem When a measure experiment records the latency distribution, the observed percentiles are point estimates subject to sampling variability. Using the raw baseline percentile as the threshold would cause frequent false positives — the same problem as using $\hat{p}_{\text{exp}}$ directly as the pass-rate threshold (Section 3.1).

Example: A baseline observes $Q_{0.95} = 480\text{ms}$ from $n_s = 935$ samples. A subsequent test with $n_s = 192$ samples observes $Q_{0.95} = 495\text{ms}$. Is this a degradation, or normal variance?

12.4.2 Upper Confidence Bound The methodology derives latency thresholds as **one-sided upper confidence bounds** on the baseline percentile. The formula uses the standard error of the mean as a proxy for percentile uncertainty:

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

where:

- $Q_{\text{baseline}}(p_j)$ is the baseline percentile value
- $z_\alpha = \Phi^{-1}(1-\alpha)$ is the one-sided critical value (e.g., $z_{0.05} = 1.645$)
- s is the baseline sample standard deviation
- n_s is the baseline sample count (successful samples only)

The derived threshold is then:

$$\tau_j = \max\left(Q_{\text{baseline}}(p_j),\ \lceil \hat{\tau}_j \rceil\right)$$

The ceiling ensures integer millisecond thresholds, and the max ensures the threshold is never tighter than the raw baseline observation.

12.4.3 Statistical Interpretation This derivation answers a more modest engineering question:

> ”What is a conservative upper tolerance for this baseline percentile, allowing for ordinary sampling variation?”

A test with observed $Q(p_j) \leq \tau_j$ means: the observed percentile is within the tolerance derived from the measured baseline and its overall spread.

A breach ($Q(p_j) > \tau_j$) means: the observed percentile exceeds that baseline-derived tolerance, providing evidence suggestive of latency degradation.

12.4.4 Limitations of this Approach The formula $\hat{\tau}_j = Q(p_j) + z \cdot s/\sqrt{n_s}$ uses the standard error of the *mean* as a proxy for the uncertainty of the *percentile*. It should therefore be understood as an engineering approximation, not as a formal non-parametric confidence interval for the quantile. For the true sampling distribution of an order statistic, the density of the distribution at the percentile point is required — which is unknown in the non-parametric setting.

The approximation is conservative in practice for the following reasons:

- The standard deviation s reflects the overall spread of the distribution, not just the local density at the percentile point
- The max floor ensures thresholds are never tighter than the raw observation
- The ceiling rounding provides an additional (small) buffer

For baseline sample sizes typical of measure experiments ($n_s \geq 500$), the approximation produces thresholds that are close to what a bootstrap confidence interval would yield, at far lower computational cost. For small baselines ($n_s < 100$), the derived thresholds may be wider than necessary — a conservative error that produces fewer false positives rather than more.

12.4.5 Worked Example **Baseline**: $n_s = 935$, $s = 145$ms, $Q_{0.95} = 580$ms, confidence $= 0.95$

$$z_{0.05} = 1.645$$

$$\text{SE} = \frac{145}{\sqrt{935}} = \frac{145}{30.58} = 4.74\text{ms}$$

$$\hat{\tau}_{0.95} = 580 + 1.645 \times 4.74 = 580 + 7.80 = 587.80$$

$$\tau_{0.95} = \max(580, \lceil 587.80 \rceil) = 588\text{ms}$$

A subsequent test observing $Q_{0.95} = 580$ms would pass ($580 \leq 588$). An observation of $Q_{0.95} = 600$ms would breach ($600 > 588$).

12.5 Sample Size Requirements for Percentile Estimation

12.5.1 The Problem An empirical percentile $Q(p)$ is computed from the $\lceil p \cdot n_s \rceil$-th order statistic. When n_s is small relative to p, the estimate is unreliable:

- For $p = 0.99$ with $n_s = 10$: $\lceil 0.99 \times 10 \rceil = 10$ — the ”99th percentile” is simply the maximum value. A single outlier determines the result.
- For $p = 0.99$ with $n_s = 50$: $\lceil 0.99 \times 50 \rceil = 50$ — still the maximum. The p99 only becomes distinct from the maximum when $n_s \geq 100$.

12.5.2 Minimum Sample Sizes The methodology enforces minimum sample counts for each percentile level based on the requirement that the percentile estimate be based on at least one observation *below* it in the sorted order:

Percentile	p	Minimum n_s	Rationale
p50	0.50	5	With $n_s = 5$: index = 2, two values below, two above
p90	0.90	10	With $n_s = 10$: index = 9, one value above
p95	0.95	20	With $n_s = 20$: index = 19, one value above
p99	0.99	100	With $n_s = 100$: index = 99, one value above. Below 100, p99 =

These thresholds ensure that the percentile estimate is not degenerate (i.e., not simply the minimum or maximum of the sample).

12.5.3 The Feasibility Gate For **VERIFICATION** intent with latency enforcement enabled, the framework checks *before any samples execute* whether the expected number of successful samples meets the minimum requirement:

$$n_{s,\text{expected}} = n_{\text{planned}} \times \hat{p}_{\text{baseline}}$$

If $n_{s,\text{expected}} < n_{s,\text{min}}(p_j)$ for any asserted percentile p_j, the framework raises a configuration error — the same mechanism used for the pass-rate feasibility gate (Section 5.7.1).

Example: A test with $n_{\text{planned}} = 50$ and baseline $\hat{p} = 0.80$ yields $n_{s,\text{expected}} = 40$. A p99 assertion requires $n_{s,\text{min}} = 100$. The test is infeasible and fails immediately with a diagnostic message.

12.5.4 Indicative Results When sample size falls below the minimum but the test is not subject to the feasibility gate (SMOKE intent, or advisory mode), the framework still evaluates the percentile but marks the result as **indicative**:

> ”The p99 result is based on $n_s = 40$ samples (minimum recommended: 100). This result is indicative — a directional signal, not a statistically reliable estimate.”

This mirrors the SMOKE/VERIFICATION asymmetry for pass-rate testing (Section 5.7.3). The percentile is computed and reported, but its epistemic weight is explicitly qualified.

12.6 Enforcement Modes

Latency assertions support two enforcement modes, reflecting the practical reality that latency profiles are environment-dependent:

Mode	Breach behaviour	Default	Rationale
Advisory	Warning in output; test passes	Yes	Baseline may have been recorded on differe
Enforced	Test fails	No	Latency is a first-class SLA dimension

Why advisory is the default: A baseline generated on CI hardware with dedicated resources may record $Q_{0.95} = 480\text{ms}$. The same workload on a developer laptop with competing processes may observe $Q_{0.95} = 650\text{ms}$ — a genuine environmental difference, not a regression. Failing the test by default would produce false positives that erode trust in the framework.

When enforcement is enabled, latency breaches fail the test, and the VERIFICATION feasibility gate is active. This is appropriate for environments where hardware consistency is controlled (dedicated CI, staging environments).

12.7 Relationship to Pass-Rate Testing

The table below summarises how the two quality dimensions parallel each other in statistical treatment:

Aspect	Pass Rate	Latency
Statistical model	Parametric (binomial)	Non-parametric (empirical percentiles)
Estimand	Success probability p	Percentile quantiles $Q(p_j)$
Threshold derivation	Wilson score lower bound	Upper confidence bound on percentile
Baseline storage	$(\hat{p}, k, n)$	$(Q_{0.50}, Q_{0.90}, Q_{0.95}, Q_{0.99}, s, n_s)$
Feasibility gate	N_{min} from Wilson bound	$n_{s,\text{min}}$ from percentile reliability
Indicative marking	Undersized sample note	Undersized sample note
Enforcement	Always enforced	Advisory by default; opt-in enforcement

The two dimensions are evaluated independently and combined with logical conjunction. This independence means that latency analysis can never compensate for a pass-rate failure, and vice versa — each dimension must meet its own threshold.

References

1. Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158), 209-212.
2. Agresti, A., & Coull, B. A. (1998). Approximate is better than ”exact” for interval estimation of binomial proportions. *The American Statistician*, 52(2), 119-126.

3. Brown, L. D., Cai, T. T., & DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical Science*, 16(2), 101-133.

4. Newcombe, R. G. (1998). Two-sided confidence intervals for the single proportion: comparison of seven methods. *Statistics in Medicine*, 17(8), 857-872.

5. Hanley, J. A., & Lippman-Hand, A. (1983). If nothing goes wrong, is everything all right? Interpreting zero numerators. *JAMA*, 249(13), 1743-1745. [The ”Rule of Three”]

6. Clopper, C. J., & Pearson, E. S. (1934). The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4), 404-413.

7. Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian Data Analysis* (3rd ed.). Chapman and Hall/CRC. [Beta-Binomial posterior predictive]

8. Jeffreys, H. (1946). An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London. Series A*, 186(1007), 453-461. [Jeffreys prior]

9. Hyndman, R. J., & Fan, Y. (1996). Sample quantiles in statistical packages. *The American Statistician*, 50(4), 361-365. [Percentile interpolation methods]

10. David, H. A., & Nagaraja, H. N. (2003). *Order Statistics* (3rd ed.). Wiley-Interscience. [Order statistics and empirical percentiles]

This document is intended for review by professional statisticians. For operational guidance, see the documentation in your framework of choice: punit, feotest. For the reference implementation of all statistical computations described here, see javai-R.