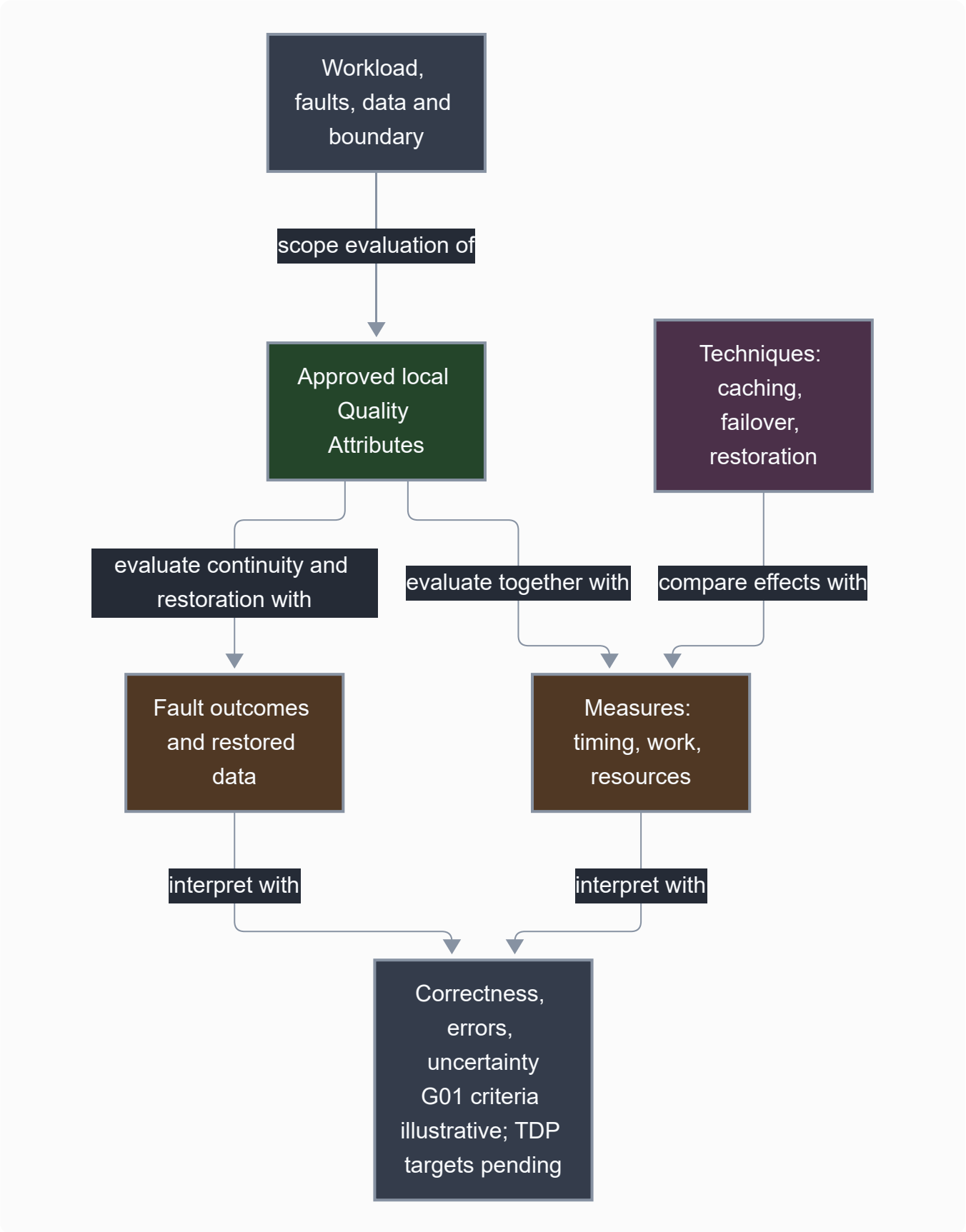
Quality attributes: technical narrative

Document review: **pending** after the interaction additions. Earlier editorial approvals remain historical; semantic, lifecycle and target states retain their separate scope.

This narrative helps developers and architects scope a performance evaluation, reconcile its observations and compare candidate techniques without overlooking correctness or cost. The five-concept comparison distinguishes the approved local qualities; Performance guidance holds their canonical measures, scenarios and worked calculations. G02 guidance extends evaluation to faults, restoration and abnormal inputs. Both scenarios describe hypothetical services, not this portal or observed TDP endpoints.

Conceptual overview



Text equivalent: Workload and boundaries scope the five local approved local quality concepts. Resource techniques need joint timing, throughput, resource, correctness and error evidence. Interpret results with uncertainty and the criteria's explicit status: G01 uses illustrative, unapproved criteria; real TDP targets

require a separate owner decision. Arrows show evaluation dependencies, not causal guarantees or taxonomy. Availability and reliability additionally require continuity, restoration and acknowledged-data checks under defined faults.

Provenance: editorial synthesis of the linked approved local concepts and canonical G01 methods below and synthetic G02. The source revision and reviewed paths are retained in the repository's review record. No backend or external framework source is newly verified; the recorded editions and access limits remain applicable. The arrows organize evaluation, not observed architecture, domain membership or a framework hierarchy. They establish no causal guarantee, measured TDP result or approved target.

Performance and capacity

Performance benefits from a defined workload and boundary; apply it where a response matters. Instrument clients and service boundaries, create representative data, and measure distributions under load. Building representative environments takes development time; operations must maintain monitoring and repeat tests. Synthetic runs cannot establish production behavior. Verify with the G01 scenario and compare observations to separately approved targets.

Capacity helps identify a sustainable workload before constraints fail. Apply it where demand or concurrency varies; require a workload model, saturation indicators and an explicit success criterion. Load generation and environment fidelity cost time and money. Limits depend on data, dependencies and configuration; increasing capacity may increase cost or mask a bottleneck. Verify by stepping load and recording errors, throughput, latency and resource consumption over a defined window.

Response time is useful for user-visible journeys: instrument start and complete response at a consistent boundary, include successful populations and errors, and verify percentiles under representative load. Maintain the instrumentation and repeat measurements as the journey changes; client, network and server delays all contribute. Latency helps localize a hop or operation when its event boundaries and clocks are known. Collecting traces and synchronizing clocks require setup and maintenance; a short hop does not prove a short journey. Compare hop measurements with end-to-end observations. Throughput helps identify completed work capacity: count successes over a fixed window and record offered load and errors. Maintain the load generator and counters for repeat comparisons; batching and slow or failed work can distort them, so verify with latency and correctness. All three are measures, not independent quality attributes.

> **Example: define the interval before comparing timings.** In hypothetical G01, client dispatch to complete receipt excludes request preparation and result presentation. A shorter dependency span cannot establish the complete user wait. Use a monotonic clock at each measurement boundary and correlate spans without subtracting unrelated host timestamps. The canonical journey explains the nested intervals; it reports no executed TDP trace.

Efficiency and time

Performance efficiency helps compare useful work against resources. Establish comparable workload, resource accounting and boundaries first; maintain the accounting instrumentation and repeat comparisons after tuning. A cheaper result can harm latency or resilience. Verify work completed and resources consumed together, not a single utilization value.

Time behaviour helps expose tail delays. Use synchronized measurement points, a defined population and percentile window. Maintain those measurement points and the test environment; averages conceal outliers and clocks or network hops can distort conclusions. Reducing work may help if redundant work exists, but can alter results; verify correctness as well as timing. Fast and Responsive are scoped prompts, not thresholds.

Responsive is useful as a prompt for interactive journeys, but needs a defined interaction and timing measure before testing; Fast similarly needs a task, boundary and threshold. Work with stakeholders to specify the interaction or task and build a representative test; neither label establishes a confirmed mapping or threshold. Verify proposed measures with users and compare observed distributions under representative load.

Resources and techniques

Resource utilization helps find finite-resource pressure. Collect CPU, memory and network counters with units and time windows; maintain the counters and alerts as resource limits change. Utilization alone does not prove saturation or efficiency. Caching may reduce repeated work when reuse and invalidation are understood, at memory and consistency cost. Scheduling can smooth demand when deferral is acceptable, but adds queueing and operational complexity. Compare baseline and treatment under equivalent workloads and check correctness, latency and failure rates. Measures and channel capacity describe boundaries and limitations.

For CPU, collect core-normalized usage over stated windows to find compute pressure; profiling and sampling cost time and may add overhead. Verify suspected saturation against queues and latency, since high CPU can be useful work. For memory, collect working-set and limit data to expose pressure; maintain monitoring and investigate heap use when pressure appears. Caches can increase memory to reduce CPU or latency. Verify peaks and failures under representative data. For bandwidth, measure bits per second and link limits to identify transfer pressure; maintain collection and repeat network tests as conditions change. Protocol overhead or contention complicates interpretation. Compare transfer timing and errors with link measurements. Channel capacity is a conditional communication limit, not observed application throughput or a separate quality attribute. Establish link conditions and competing traffic, budget for measurement, then test achieved transfer rates and latency; protocol overhead and other bottlenecks prevent assuming the theoretical limit.

Controlled evaluation of G01

Use the canonical G01 and variants as one experiment family. All values here are synthetic, criteria are illustrative and unapproved, and the read service is hypothetical. No backend test was run. The five-concept comparison keeps their questions distinct; this method evaluates scenarios and techniques, rather than treating an attribute as an intervention.

1. **Fix conditions and boundaries.** G01 uses 100,000 synthetic records, up to 100 concurrent users, two service cores in the worked runs, an empty-cache one-minute uniform ramp to 50 requests/s, then a 600 s hold including warming. Time client dispatch to complete receipt. Complete user wait additionally includes preparation and presentation; service/dependency spans are nested intervals, not replacements for that client measurement. Correlate traces without subtracting unsynchronized host clocks.
2. **Reconcile the population.** Record arrivals, generator shortfalls, correct authorized completions, errors and in-flight work at window edges. Count incorrect content, timeouts (5 s in the example), rejections and unauthorized results as errors. The worked holds have no edge work and 30,000 arrivals. Run A has 29,850 successes and 150 errors: 49.75 successful requests/s and 0.50% errors. Run B has 29,400 successes and 600 errors: 49.00 requests/s and 2.00% errors. Throughput alone would conceal B's failure.
3. **Evaluate the joint criteria.** G01 requires successful p95 ≤ 500 ms, throughput ≥ 49 requests/s, errors < 1%, each one-minute CPU average < 80% and peak service memory < 2 GiB. A's p95 420 ms, highest CPU minute 65% and peak memory 1.2 GiB satisfy the example. B's p95 680 ms and CPU 85% fail alongside its errors. The canonical results retain the full observations and interpretation. Preserve errors in their own report, and aggregate observations rather than averaging percentiles; the population and nearest-rank calculation explain how to reproduce the success-only p95.

4. **Find a candidate demand boundary.** G01-C resets the cache and repeats the ramp/hold at 40, 50 and 60 requests/s. Its throughput criterion is 98% of the offered rate, retaining G01's other criteria. At 60, the synthetic hold has 36,000 arrivals, 35,100 correct completions and 900 errors: 58.50 requests/s, 2.50% errors, p95 760 ms, CPU 88%, memory 1.5 GiB. It fails timing, errors, CPU and the relative throughput criterion: 58.50 is below the required 58.8 requests/s at this offered rate. In G01-C, 50 is the highest tested passing rate, not an exact maximum or safe operating limit. Refine steps and repeat with representative dependencies before selecting headroom.
5. **Compare technique effects.** G01-E replays identical seeded read traces and ten controlled updates at fixed times under equivalent conditions, alternating run order and repeating trials. Reset data/cache before each ramp. Check all responses against a version/authorization oracle. Reads dispatched after an acknowledged update must see that version or newer in this example. Retain G01's criteria, add zero stale/unauthorized results and at least 20% lower CPU-seconds per correct read. These remain unapproved example rules. This G01-E comparison changes the technique under equivalent conditions; G01-C changes demand in separate runs. Neither is an additional observation of G01 Run A or B.

   Example: a success percentile leaves errors in a separate population. Synthetic Run A's p95 of 420 ms describes its 29,850 correct, authorized successes during the 600 s hold. Its 150 errors remain in the arrivals-based error report, and 450 successes take 900 ms. A favorable percentile cannot replace the error check or promise every user's wait. See the canonical population.

   Example: the comparison needs the same work. Hypothetical G01-E fixes read-key popularity, update times, payloads, dependency version, authorization controls and instrumentation across variants. Without those controls, lower CPU could reflect a different workload or omitted checks. Alternating run order and repeating trials are proposed methods, not executed evidence; the synthetic rows do not quantify run-to-run uncertainty. See the controlled method.

The no-cache baseline and invalidation-aware treatment each complete 29,850 correct reads over equivalent 600 s holds. The canonical CPU accounting shows a 40% reduction in service CPU-seconds per correct read. Baseline p95 is 480 ms, highest CPU minute 95% and peak memory 0.8 GiB; treatment is 420 ms, 65% and 1.2 GiB. The baseline fails CPU; the treatment meets the joint illustrative criteria with 50% more peak memory. These CPU totals cover the service and cache only; dependency resource costs remain separately recorded. The example network peak is 12 Mbit/s on a 100 Mbit/s channel, with no network acceptance threshold. Cost shifting must remain visible.

> **Example: a lower CPU ratio does not waive other criteria.** In synthetic G01-E, the treatment reduces service CPU per correct read while peak service memory increases. The ratio describes resource cost per useful result; the one-minute CPU percentage constrains utilization across two allocated cores. Dependency resources and invalidation effort still require separate accounting. Neither a better ratio nor lower service CPU establishes a total-cost saving. See the canonical comparison.

Cache evaluation and rejection criteria

A TTL-only alternative uses 600 CPU-s and shows p95 400 ms, but returns 120 stale responses. Its correct completions fall to 29,730 and its errors rise to 270 (0.90%). It fails the zero-stale rule even though its aggregate error rate and timing satisfy G01. Cache hits and lower CPU do not establish correctness.

> **Example: freshness can reject an apparently faster alternative.** The synthetic TTL-only alternative's 120 stale responses fail G01-E's illustrative zero-stale criterion despite favorable timing and an aggregate error fraction below 1%. They count as errors and are excluded from correct useful work, even if HTTP reports success. The example's post-update rule is not an approved TDP freshness obligation.

Use the alternative comparison to select a candidate. No cache minimizes retained memory and invalidation code but repeats dependency work. A bounded TTL cache introduces key/entry overhead, expiry, evictions and stale-result risk. Update-coordinated invalidation adds event handling, races, missed-event recovery and cross-instance coordination. All variants need permission-scoped keys and tests of authorization changes. Neither TTL nor invalidation alone proves the intended consistency semantics.

Record hit/miss rates, evictions, duplicate fills, invalidation delay, CPU, memory and dependency costs. Exercise cold/warm states, low reuse, hot keys, concurrent updates, lost/delayed invalidations, restarts and permission changes. Apply the oracle to every result, including HTTP successes. Retain uncertainty and repeated-run variation; these single synthetic holds cannot quantify it. If the hypothetical journey proves relevant, the proposed next decision is whether to investigate the invalidation-aware candidate under representative conditions. Applicability, freshness semantics, targets, acceptable headroom, budgets, priorities and owner remain pending; this example authorizes no implementation or reduction of authorization controls.

Availability and reliability

Use the G02 guide and family comparison as the canonical scope. This section retains enough method for the standalone PDF. All service behavior, observations and criteria below are hypothetical/synthetic and unapproved for TDP; no fault injection or backend investigation was run.

Measurement and G02 method

Define an up state for the declared service, eligible users and required-use schedule. In G02 a valid read must return the expected authorized version within 2 s; continuous idealized monitoring gives exact interval boundaries. Real probes sample and must report resolution and unknown intervals. Uptime is total up seconds; downtime is the union of down intervals in the same window. Time availability is up / (up + down), with no silent maintenance or unknown-time exclusions. Count service failures separately from rejected invalid requests.

G02 uses a controller (stimulus source) to stop one instance at t=1,200 s (stimulus), within the hypothetical service/routing/store/recovery boundary (artifact). The environment fixes 100,000 synthetic records, 10 valid reads/s, one update every 60 s and a 3,600 s required-use window. The instance stays down. The response is continued correct service where possible, detection, restoration and reconciliation of acknowledged changes. Its illustrative response measures are at least 99% time availability, restoration within 30 s, zero lost acknowledged updates, duplicate effects or unauthorized results. A valid read counts as successful only when it returns the expected authorized version within 2 s. Failed reads contribute to the declared down intervals. These are the six scenario components, not observed architecture.

Schedule 60 distinct update keys at t=30, 90, …, 3,570 s. In synthetic Run B, the controller holds the t=1,230 s update until restoration and submits it before the next scheduled update; all others are submitted on schedule. The runs acknowledge all 60 outside their down intervals. This controller rule implies no service-side queue or retry guarantee. Reads dispatched after acknowledgment must return that version or newer under the example oracle. Reconcile every key and effect after recovery. Failed reads complete immediately; no in-flight work crosses the window. Preserve configuration, trace, fault time, monitoring and resources when comparing interventions; reset state, alternate order and repeat.

Synthetic observation	A: shorter restoration	B: longer restoration
Down interval and total	[1,200, 1,220) s; 20 s	[1,200, 1,280) s; 80 s
Up time / availability	3,580 s / 99.444…%	3,520 s / 97.777…%
Correct / failed / offered reads	35,800 / 200 / 36,000	35,200 / 800 / 36,000
Preserved / acknowledged updates	60 / 60	60 / 60

Synthetic observation	A: shorter restoration	B: longer restoration
Lost updates / duplicates / unauthorized results	0 / 0 / 0	0 / 0 / 0
Result against G02	Joint pass	Fail both time criteria

> **Example: data and time both constrain recovery.** B preserves updates but misses the time bounds. Conversely, restoring in 20 s with one lost acknowledged update would fail the zero-loss criterion. A process restart and an HTTP health check do not establish correct recovered state.

A separate strict continuity variant C requires zero failed valid reads under the same instance fault: synthetic 36,000/36,000 correct reads and 3,600 s uptime pass it. A fails that stronger check despite passing G02. Extend the fault model explicitly before claiming coverage of shared-store, network or correlated failure. Time availability does not establish failure-free interval reliability.

For MTBF, declare observation/exposure and service-failure counting rules. A separate repairable-service sample with 35,800 s uptime, 200 s down and two failures estimates MTBF as 17,900 s and mean down duration as 100 s. Its sample time fraction is 17,900 / 18,000 = 99.444…%. Predictive use needs additional model assumptions and uncertainty; zero observed failures gives no finite estimate and no proof of defect absence. Demand-weighted request success and time availability can differ. See the canonical measurement limits.

Techniques, costs and rejection criteria

Quality question	Candidate benefit, prerequisites and cost	Checks and limits
Availability	Standby routing/failover may shorten interruption when health signals, routing control and spare capacity exist; costs duplicate resources, telemetry and tuning	Compare restart-only B with failover A under the same declared capacity except routing intervention; record detection/switch delay and correct outcomes. Reject late recovery, stale state or authorization failures
Reliability	Defect prevention, regression and sustained representative trials may reduce interruption risk; requires a service-failure definition and trustworthy oracle, and costs delivery time	Record exposure, events and release/environment versions. One clean run establishes no probability or universal guarantee; software faults are not hardware wear
Fault tolerance	Isolation/redundancy may preserve work with independent failure domains and enough residual capacity; adds coordination, spare-instance and test cost	Exercise strict C, health detection, fencing and split-brain paths; reject any failed/incorrect valid read under that criterion. Shared dependencies can defeat independence; check latency under reduced capacity
Recoverability	Backup/restore and replay may recover affected state with retained logs, key access and a rehearsed reconciliation oracle; costs storage, drills and operator attention	Restore into isolation and reconcile all acknowledged keys, duplicates, permissions and version freshness. Reject lost effects, unsafe state or missed time bound; backup age and replay delay constrain loss/time
Robustness	Bounded validation and isolation can protect valid work from abnormal input; requires defined boundaries and safe errors; adds validation work and risks rejecting legitimate traffic	G02-R sends 100 malformed/oversized requests: require 100 controlled rejections, zero partial writes/disclosure and unchanged valid-work criteria. A crash on request 100 fails even with acceptable aggregate availability
Faultlessness	A normal-operation control and defect-focused regressions can expose failures;	Synthetic control: 36,000 correct reads, 60 preserved updates, zero observed failures.

Quality question	Candidate benefit, prerequisites and cost	Checks and limits
	requires representative traces and costs maintenance	This demonstrates no latent-defect absence and confirms no ISO definition
Historical Maturity	Source/requirements review can recover the intended historical meaning before choosing tests; requires edition-specific evidence and owner review time	Meaning and lifecycle stay unresolved; neither normal control nor label replacement proves equivalence with Faultlessness or organizational maturity

The A/B comparison illustrates an intervention, without claiming executed causality. Record resource allocation, spare-instance time, restore labor, monitoring overhead and run-to-run variation alongside service outcomes. Additional retries can amplify load; stale replicas and split brain threaten data integrity; recovery credentials and backup access retain security controls. Never buy apparent availability by returning unauthorized or incorrect data. Check the existing Performance criteria with residual capacity, rather than assuming redundancy preserves timing. The executive decisions remain pending.

Terminology and approval limits

Available, Fault-tolerant, fault-tolerance, Recoverable and Robust/robustness map through scoped related proposals, not inferred equivalents. Dependable and dependability provide broad trust framing that can include security and safety; they create no duplicate quality leaf or combined score. Consistent remains ambiguous across data, behavior and policy; the example's version rule does not settle that label. Security's availability concern reuses qa:availability. Security guidance is linked below; safety and operational diagnosis remain separate work.

The ISO catalogue confirms a published 2023 edition, not its full definitions. Q42 is primary for its own tags and secondary for its ISO attributions. The historical preview fetch timed out; retain separate Maturity/Faultlessness records with their stated definition limits. Six new qualities independently propose runtime and development-time reading; Maturity remains unassessed, with no inherited assignments. No owner, budget, priority, target or lifecycle approval is supplied by this documentation. Source limits and mappings remain visible in the portal; this PDF retains the same limits.

Sources and further reading

The canonical G02 source record retains consultation dates, scope and access limits. The ISO/IEC 25010:2023 catalogue provides publication metadata; full definitions remain unverified. The 2011 catalogue documents a withdrawn edition, without proving Maturity/Faultlessness equivalence. The Q42 model is authority for its own tags, not primary ISO wording. The SEI report CMU/SEI-2003-TR-004 supplies the historical six-component scenario method, not a runtime guarantee.

Crosswalk and decisions

The Q42 dimensions are overlapping tags; performance decisions interact with maintainability, operability, reliability, safety, security, flexibility, suitability and usability. Do not infer ISO taxonomy from them. The secondary labels efficiency and responsiveness retain proposed related mappings. Review their source meanings and intended context with the owner before selecting resource or timing measures; this requires source review and an owner decision rather than treating the labels as synonyms. Runtime observation and development evaluation complement one another; generated lifecycle groups are author assessments, not inherited requirements. Use the checklist and rubric before proposing implementation or priorities.

The Q42 tags are reading prompts rather than additional local attributes. For **Efficient**, compare useful work and resources; maintain resource accounting alongside latency and error measurements. For **Flexible**, test changes in demand and configuration; abstractions require implementation and

maintenance and can add delay. For **Maintainable**, review tuning complexity and repeat profiling after a change; specialized optimizations cost maintenance and may obscure defects. For **Operable**, instrument queues and saturation and exercise alerts; telemetry costs storage and attention and can itself add overhead. For **Reliable**, run sustained and failure tests with dependencies; redundancy costs resources and may affect latency. For **Safe**, establish the relevant harm and deadline with an owner, then test late and failed responses; safeguards cost implementation and may limit throughput. For **Secure**, measure with required controls enabled and test their failure paths; controls consume resources but cannot be omitted on timing evidence alone. For **Suitable**, validate the actual task and success criteria with stakeholders; a fast irrelevant result is not useful. For **Usable**, observe representative users and feedback during delays; user studies cost time and synthetic timings alone do not establish usability. Each assessment requires its own context and evidence; these interactions do not define the other quality slices.

Runtime grouping supports monitoring of real workloads but needs telemetry, operational effort and careful interpretation of changing conditions. Development-time grouping supports repeatable comparisons before release but needs representative fixtures and maintenance; verify both against each other where production evidence is available. These are reading dimensions, not attributes or inherited framework classifications.

> **Example: testing before release still measures the service.** A hypothetical pre-release G01 evaluation measures service behavior under load, not developer productivity or build duration. It can inform development feedback without establishing production behavior or confirming lifecycle assignments. See the canonical lifecycle explanation.

The generated groups below derive from individual published assessment records. A concept assigned both dimensions appears in both groups with one identity; navigation supplies no assignment. `None.` means no published member matches that group, not absence of the concern from TDP or complete corpus coverage. Follow Lifecycle reading for assessed, unassessed and empty-group distinctions. Editorial approval of this narrative does not confirm those assessments or their assignments.

Security goals and controls

Evaluate confidentiality, integrity and accountability independently using the synthetic G03 scenario: check responses, stored state and attributable decisions for the declared identities and policy. Authentication verifies a claimed identity; authorization decides permitted actions. Neither alone proves the outcome. Authenticity requires evidence of genuineness; non-repudiation needs evidence appropriate to a dispute; resistance requires a specified attack and conditions. The example cannot demonstrate those properties for every case.

Compare a policy change against a controlled baseline and account for policy maintenance, evidence storage, privacy, false denials and latency. Reject a treatment that meets one criterion while disclosing information or altering data improperly. The workload, threat model and criteria are illustrative and unapproved; no TDP security assessment is claimed. Availability has its own service-window boundary.

Interaction and accessibility

Define the task, participant configurations, outcome denominator and error recovery criteria before comparing changes. Test keyboard and declared assistive technology, retain individual barriers and verify that validation messages are associated with fields and preserve entered data. Inline validation and separate help have different implementation and maintenance costs; see the technique comparison. An isolated test does not establish WCAG conformance or production behavior.

For a comparable evaluation, keep the task, data, participant configurations and observation procedure stable between the help-only baseline and inline guidance treatment. Record recruitment, implementation, localization and ongoing maintenance costs as well as correctness, assistance and

individual failures. Reject the treatment if it loses values, blocks an input method or lowers correct completion, even when it saves time. Use repeated tasks for learning, interviews for presentation and involvement, and task observations for control and recovery; the concept-specific methods explain their limits. No sample or criterion is an approved target.

Concern	Method and rejection signal
Usability and interaction capability	Observe correctness, effort and satisfaction across the complete journey; reject a treatment that breaks any required stage.
Accessibility and disability access	Test declared assistive-technology configurations and inspect barriers individually; reject a blocked path even if aggregate completion improves.
Inclusivity	Sample relevant circumstances and compare observations; document omitted populations rather than generalizing from them.
Recognizability and learnability	Test unprompted action selection and a later unprompted repeat separately; record wrong selections and assistance.
Operability and error protection	Exercise controls with intended input methods and submit invalid data; reject inaccessible controls or loss of valid entries.
Assistance and self-descriptiveness	Ask users to explain the error, then find and use help if needed; record mistaken interpretations and facilitator rescue separately.
Presentation and historical aesthetics	Compare screens under the same conditions and collect explained preferences; neither preference nor the comparison establishes an edition mapping.
Engagement	Observe and discuss continuation in a realistic session; forced completion is not evidence of involvement.

The candidate dispositions retain proposed mappings and exclusions; a split label alone does not define a separate measurement. These methods add recruitment, analysis and maintenance costs and do not prove population prevalence or standards conformance.

Lifecycle reading groups

Explicit per-concept assessments; no parent inheritance. Assignments can overlap.

Runtime

- Accessibility — assessed; proposal: A barrier occurs when someone uses the interface with a particular technology; prototype evaluation can reveal it during development.
- Accountability — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Appropriateness recognizability — assessed; proposal: Recognition occurs when the person encounters the action; labels and placement can be evaluated in a prototype during development.
- Authenticity — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Availability — assessed; proposal: Running users need service readiness; pre-release evaluation tests the declared up-state predicate.
- Capacity — assessed; proposal: Running services encounter workload limits; development explores those limits before release.
- Confidentiality — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Disability access — assessed; proposal: The access path is encountered in operation and can be inspected in a representative prototype before release.
- Fault tolerance — assessed; proposal: Runtime service may continue during faults; development checks the fault model and remaining capacity.

- Faultlessness — assessed; proposal: Normal runtime operation can be observed; development evaluates representative normal traces before release.
- Inclusivity — assessed; proposal: Participation depends on real contexts of use; research with a varied sample can identify exclusions during design.
- Integrity — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Interaction capability — assessed; proposal: Discovery, control and recovery occur in use; an end-to-end prototype can expose a broken transition before release.
- Interface aesthetics and presentation — assessed; proposal: Presentation is perceived when the interface is used; mockups can be compared before implementation, subject to their fidelity.
- Learnability — assessed; proposal: Learning and recall occur across use; repeated prototype tasks can test the proposed design during development.
- Non-repudiation — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Operability (interaction) — assessed; proposal: Control is exercised at runtime; a representative interface can be exercised with declared input methods before release.
- Performance efficiency — assessed; proposal: Running services consume resources to deliver work; development compares the cost of candidate designs.
- Performance — assessed; proposal: Running services experience workload and timing; development evaluates response under representative load.
- Recoverability — assessed; proposal: Runtime recovery restores service and data; development rehearses restoration and validates recovery procedures.
- Reliability — assessed; proposal: Running services face interruptions over time; development evaluates failure histories and representative continuous operation.
- Resource utilization — assessed; proposal: Running services consume finite resources; development measures consumption against proposed budgets.
- Robustness — assessed; proposal: Running services encounter abnormal input; development explores boundaries and verifies controlled rejection.
- Security — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Self-descriptiveness — assessed; proposal: The interface communicates its state during use; message comprehension can be checked with a prototype in development.
- Time behaviour — assessed; proposal: Running services exhibit response distributions; development measures them under controlled conditions.
- Usability — assessed; proposal: The completed task and reported effort are experienced during use; a prototype can expose failures before release.
- User assistance — assessed; proposal: Help is consumed during the task; its discoverability and usefulness can be evaluated before release.
- User engagement — assessed; proposal: Involvement is experienced over use; a realistic prototype session can provide tentative design feedback, not a population estimate.
- User error protection — assessed; proposal: Errors and recovery occur during use; controlled invalid submissions can test the design during development.
- User interface aesthetics — assessed; proposal: Perception arises in use; representative mockups permit tentative comparison during development. This local assessment does not assign an ISO lifecycle.

Development time

- Accessibility — assessed; proposal: A barrier occurs when someone uses the interface with a particular technology; prototype evaluation can reveal it during development.
- Accountability — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Appropriateness recognizability — assessed; proposal: Recognition occurs when the person encounters the action; labels and placement can be evaluated in a prototype during development.
- Authenticity — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.

- Availability — assessed; proposal: Running users need service readiness; pre-release evaluation tests the declared up-state predicate.
- Capacity — assessed; proposal: Running services encounter workload limits; development explores those limits before release.
- Confidentiality — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Disability access — assessed; proposal: The access path is encountered in operation and can be inspected in a representative prototype before release.
- Fault tolerance — assessed; proposal: Runtime service may continue during faults; development checks the fault model and remaining capacity.
- Faultlessness — assessed; proposal: Normal runtime operation can be observed; development evaluates representative normal traces before release.
- Inclusivity — assessed; proposal: Participation depends on real contexts of use; research with a varied sample can identify exclusions during design.
- Integrity — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Interaction capability — assessed; proposal: Discovery, control and recovery occur in use; an end-to-end prototype can expose a broken transition before release.
- Interface aesthetics and presentation — assessed; proposal: Presentation is perceived when the interface is used; mockups can be compared before implementation, subject to their fidelity.
- Learnability — assessed; proposal: Learning and recall occur across use; repeated prototype tasks can test the proposed design during development.
- Non-repudiation — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Operability (interaction) — assessed; proposal: Control is exercised at runtime; a representative interface can be exercised with declared input methods before release.
- Performance efficiency — assessed; proposal: Running services consume resources to deliver work; development compares the cost of candidate designs.
- Performance — assessed; proposal: Running services experience workload and timing; development evaluates response under representative load.
- Recoverability — assessed; proposal: Runtime recovery restores service and data; development rehearses restoration and validates recovery procedures.
- Reliability — assessed; proposal: Running services face interruptions over time; development evaluates failure histories and representative continuous operation.
- Resource utilization — assessed; proposal: Running services consume finite resources; development measures consumption against proposed budgets.
- Robustness — assessed; proposal: Running services encounter abnormal input; development explores boundaries and verifies controlled rejection.
- Security — assessed; proposal: The proposed concern applies to operation and to design and pre-release evaluation; its scope remains subject to owner review.
- Self-descriptiveness — assessed; proposal: The interface communicates its state during use; message comprehension can be checked with a prototype in development.
- Time behaviour — assessed; proposal: Running services exhibit response distributions; development measures them under controlled conditions.
- Usability — assessed; proposal: The completed task and reported effort are experienced during use; a prototype can expose failures before release.
- User assistance — assessed; proposal: Help is consumed during the task; its discoverability and usefulness can be evaluated before release.
- User engagement — assessed; proposal: Involvement is experienced over use; a realistic prototype session can provide tentative design feedback, not a population estimate.
- User error protection — assessed; proposal: Errors and recovery occur during use; controlled invalid submissions can test the design during development.
- User interface aesthetics — assessed; proposal: Perception arises in use; representative mockups permit tentative comparison during development. This local assessment does not assign an ISO lifecycle.

Unassessed

- Maturity — unassessed; proposal: No lifecycle assignment is adopted before the historical meaning and scope are corroborated.
- Resistance — unassessed; proposal: The precise scope of resistance requires corroboration before assigning lifecycle relevance.

Assessed without assignment

None.

Presentation export

Download PDF. Ordinary builds are not export-verified; run the documented export command and check the export manifest for generation time, input digest and file hashes.

References and online details

These online destinations require a connection. Portal pages may change after this PDF was generated; the footer identifies this export. URLs remain readable when printed.

1. five-concept comparison: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/performance-and-capacity/#compare-the-questions
2. Performance guidance: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/
3. G02 guidance: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/reliability-guide/
4. recorded editions and access limits: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#sources-and-scope
5. Performance: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/performance/
6. G01 scenario: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#g01-scenario
7. Capacity: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/capacity/
8. canonical journey: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#request-journey-and-measurement-boundaries
9. Performance efficiency: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/performance-efficiency/
10. Time behaviour: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/time-behaviour/
11. Fast and Responsive: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#aliases
12. Resource utilization: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/resource-utilization/
13. Measures and channel capacity: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#resources-and-channel-capacity
14. canonical results: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#g01-worked-results
15. population and nearest-rank calculation: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#percentiles-and-populations

16. G01-C: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#g01-c-capacity-variant
17. G01-E comparison: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#g01-e-efficiency-variant
18. alternative comparison: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#caching-alternatives-and-verification
19. family comparison: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reliability-and-operations/
20. canonical measurement limits: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/reliability-guide/#measures-and-boundaries
21. Availability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/availability/
22. Reliability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/reliability/
23. Fault tolerance: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/fault-tolerance/
24. Recoverability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/recoverability/
25. Robustness: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/robustness/
26. Faultlessness: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/faultlessness/
27. Maturity: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/maturity/
28. executive decisions: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/executive/#availability-and-reliability
29. Source limits and mappings: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/reliability-guide/#sources-and-scope
30. ISO/IEC 25010:2023 catalogue: https://www.iso.org/standard/78176.html
31. 2011 catalogue: https://www.iso.org/standard/35733.html
32. Q42 model: https://quality.arc42.org/articles/arc42-quality-model
33. SEI report CMU/SEI-2003-TR-004: https://www.sei.cmu.edu/documents/704/2003_005_001_14213.pdf
34. Q42 dimensions: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#q42-dimensions
35. checklist and rubric: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#design-checklist
36. canonical lifecycle explanation: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/performance-guide/#lifecycle
37. Lifecycle reading: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/lifecycle/#how-to-read-the-generated-groups
38. synthetic G03 scenario: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/security-guide/#g03-scenario
39. technique comparison: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/interaction-guide/#technique-choices
40. concept-specific methods: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/reference/interaction-guide/#concept-specific-observations
41. candidate dispositions: https://zealous-bay-01d7e9503.5.azurestaticapps.net/ontologies/architecture/#interaction-candidate-boundaries
42. Accessibility: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/accessibility/#lifecycle-assessment
43. Accountability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/accountability/#lifecycle-assessment
44. Appropriateness recognizability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/appropriateness-recognizability/#lifecycle-

assessment
45. Authenticity: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/authenticity/#lifecycle-assessment
46. Availability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/availability/#lifecycle-assessment
47. Capacity: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/capacity/#lifecycle-assessment
48. Confidentiality: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/confidentiality/#lifecycle-assessment
49. Disability access: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/disability-access/#lifecycle-assessment
50. Fault tolerance: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/fault-tolerance/#lifecycle-assessment
51. Faultlessness: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/faultlessness/#lifecycle-assessment
52. Inclusivity: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/inclusivity/#lifecycle-assessment
53. Integrity: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/integrity/#lifecycle-assessment
54. Interaction capability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/interaction-capability/#lifecycle-assessment
55. Interface aesthetics and presentation: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/interface-aesthetics-presentation/#lifecycle-assessment
56. Learnability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/learnability/#lifecycle-assessment
57. Non-repudiation: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/non-repudiation/#lifecycle-assessment
58. Operability (interaction): https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/operability-interaction/#lifecycle-assessment
59. Performance efficiency: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/performance-efficiency/#lifecycle-assessment
60. Performance: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/performance/#lifecycle-assessment
61. Recoverability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/recoverability/#lifecycle-assessment
62. Reliability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/reliability/#lifecycle-assessment
63. Resource utilization: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/resource-utilization/#lifecycle-assessment
64. Robustness: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/robustness/#lifecycle-assessment
65. Security: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/security/#lifecycle-assessment
66. Self-descriptiveness: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/self-descriptiveness/#lifecycle-assessment
67. Time behaviour: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/time-behaviour/#lifecycle-assessment
68. Usability: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/usability/#lifecycle-assessment
69. User assistance: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/user-assistance/#lifecycle-assessment
70. User engagement: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/user-engagement/#lifecycle-assessment

71. User error protection: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/user-error-protection/#lifecycle-assessment
72. User interface aesthetics: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/user-interface-aesthetics/#lifecycle-assessment
73. Maturity: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/maturity/#lifecycle-assessment
74. Resistance: https://zealous-bay-01d7e9503.5.azurestaticapps.net/architecture/qualities/resistance/#lifecycle-assessment