

Technical Writing Grammar Checker Tools: Selection and Workflow Design



Goal: pick checkers built for technical and localization workflows, not generic prose



Evaluate across three layers: tool capability, team workflow, governance/security



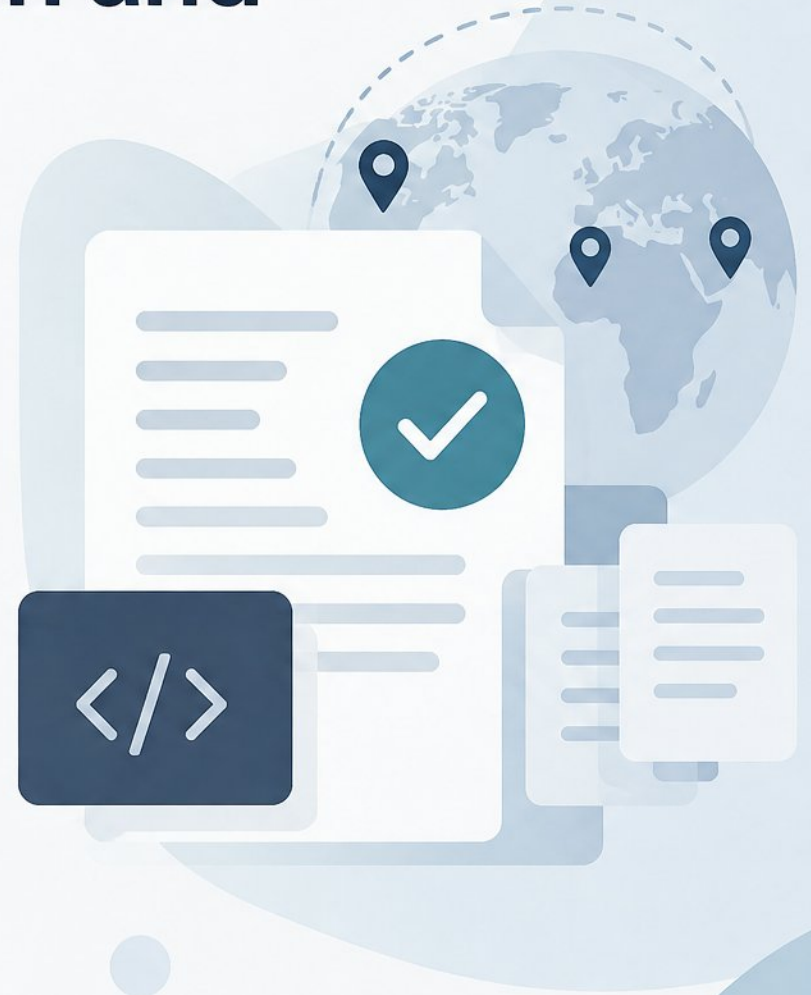
Takeaway: a reusable selection scorecard plus a workflow blueprint



Scope: grammar, style, terminology, readability, localization QA — not full editorial strategy



Running example: API docs with code samples and five target locales



Why Technical Writing Breaks Generic Grammar Checkers



Technical content: code snippets, CLI commands, API names, units, tables, controlled language



Generic checkers optimize for prose; they fail on complex sentences and domain vocabulary



Valid terms like FFT, QoS, and Cpk get flagged as errors



Rules fire inside JSDoc comments and string literals used to emit browser JS



Noisy false positives train writers to ignore the tool and miss genuine errors

API Overview

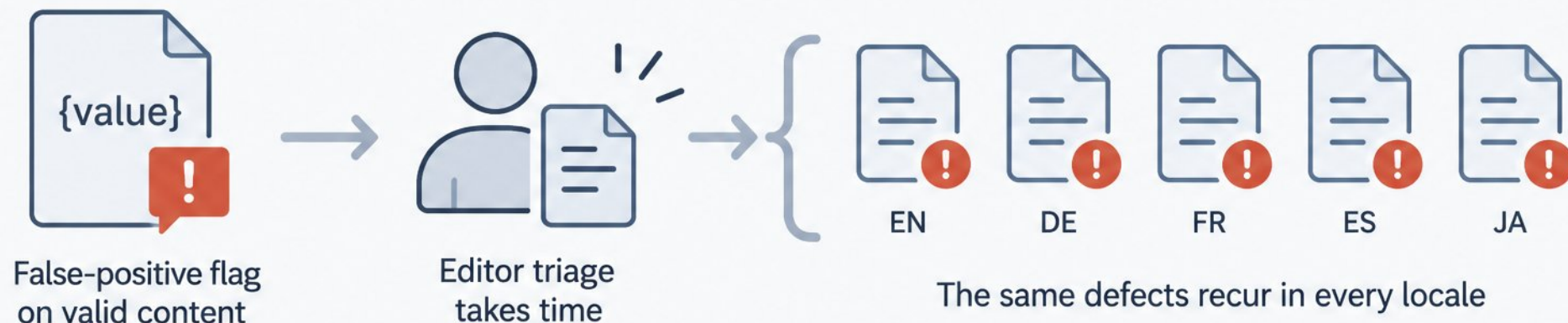
The system uses QoS to prioritize traffic and optimize performance.

```
/**
 * Process data using FFT.
 * @param input
 * @returns result
 */
function computeFFT(input) {
  return FFT(input);
}
```

The Cpk value must meet the requirement.

```
device.setConfig("mode", "auto");
```

Cost of Unfit Tools: Review Rework and Downstream Localization Damage



False flags outweigh missed errors; one spectacular flag makes writers disable the tool.



Rework compounds: editors triage noise, then the same defects recur in every locale.



Checkers miss broken placeholders, missing ICU plurals, altered markup, invalid locale tags.



Translation cost tracks source quality; fixing clarity and terms upstream is cheapest.



A fit-for-purpose layer cuts rework and improves reader comprehension.

Tool Landscape: Five Categories and Where Each One Fits

Different tools, different strengths. Know what each category is best at, and where it has limits.



General-purpose writing assistants

- General-purpose writing assistants: strong prose checking, weak on structured docs



Docs-as-code prose linters

- Docs-as-code prose linters: syntax-aware, configurable, run locally and in CI/CD (Vale, textlint)



Developer and IDE checkers

- Developer and IDE checkers: lint comments, commit messages, and Markdown without touching code



Enterprise governance platforms

- Enterprise governance platforms: centralized rules, term bases, and quality scoring



Localization QA tools

- Localization QA tools: placeholders, tags, and segment checks; mature pattern is a hybrid stack

Evaluation Criteria Part 1: Linguistic Accuracy and Technical Content Handling



Define metrics before the demo, not after



Precision = correct flags ÷ total flags



Recall = correct flags ÷ expected flags



Track false flags per page for noise level



Set thresholds per error class: verb chains, split compounds, agreement



Prioritize precision over recall; false flags erode trust



Check code blocks, inline spans, markup, variables, acronyms, units



Test pluralized identifiers, passthrough constructs, do-not-flag vocabulary

Per-Error-Class Scoring

Error Class	Precision	Recall	False Flags per Page
Verb chains			
Split compounds			
Agreement			



Evaluation Criteria Part 2:

Integration, Governance, Localization, and TCO



Integration fit:

editor plugins, CI/CD exit codes, APIs, CCMS and Git pipelines



Governance:

data residency, retention, opt-out of model training, SSO, audit logs, DLP



Localization readiness:

locale variants, term base and TM alignment, placeholder and tag validation



TCO covers licensing, admin, rule authoring, false-positive triage, change management



Worked example:

four candidate tools scored—see where each loses points

Evaluation Criteria	Tool 1	Tool 2	Tool 3	Tool 4
Integration fit				
Governance				
Localization readiness				
Total cost of ownership				
Worked example				



A structured evaluation helps you compare tools and identify trade-offs.

Customization: Rules, Vocabularies, and Terminology Governance



Build a project rule set: product names, acronyms, units, deprecated terms, forbidden phrases



Mechanisms differ: accept/reject lists, case-aware substitution, custom dictionaries, terminology files



Model terminology per concept: approved, preferred, discouraged, forbidden forms across variants



Connect checkers to the shared term base so docs and localization match



Manage the rule lifecycle: propose, review, test, publish, deprecate, communicate



Shared Term Base

Concept: User Interface

Form Type	English (en-US)	German (de-DE)
 Approved	User interface	Benutzeroberfläche
 Preferred	UI	Benutzeroberfläche
 Discouraged	User Interface	Benutzer-Interface
 Forbidden	GUI	GUI



Documentation

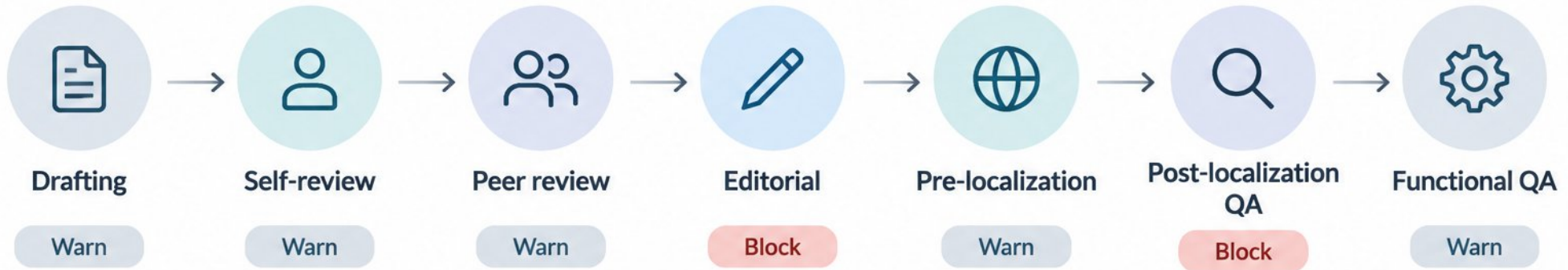


Localization

Workflow Design: Checkpoints Across the Content Lifecycle

FROM CREATION TO GLOBAL READINESS

PEOPLE
PROCESS
BETTER CONTENT



Map checkpoints: drafting, self-review, peer review, editorial, pre-localization, post-localization QA, functional QA



Assign each tool one distinct job per checkpoint; avoid duplicated or contradictory feedback



Automate low-risk checks in CI/CD; reserve human judgment for ambiguity, tone, and context



Set severity levels: block structural defects, warn on style, never block on cosmetics



Build learning loops: surface recurring rule violations as training needs

CLEARER
CONTENT
BRIGHTER
POSSIBILITIES

Automating Checks in Docs-as-Code Pipelines

Higher quality docs, earlier feedback, smoother releases.



- Reference pattern: one lint job checks out the repo, installs styles, runs Vale with review comments and `fail_on_error`.



- Practical setting: published style action disables noisy checks like sentence-case headings until the vocabulary list is complete.



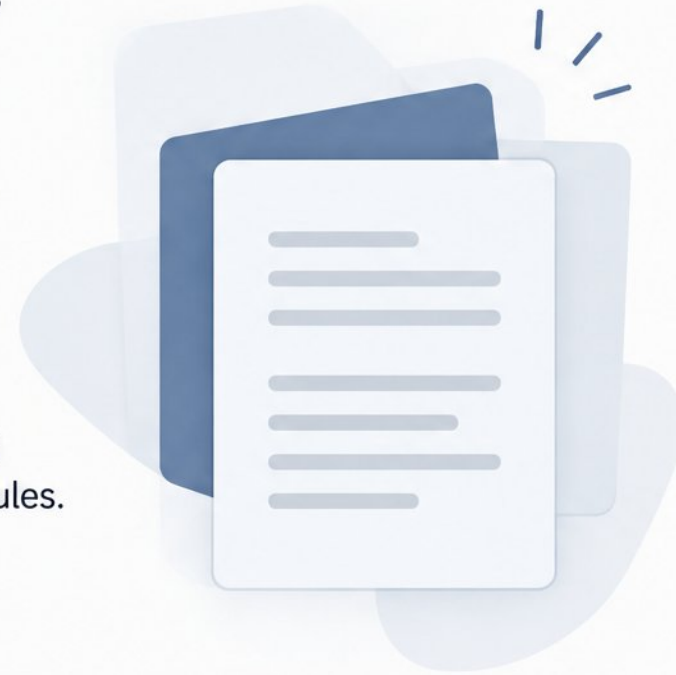
- Run checks in three places: inside the editor, at file handoff, and in CI for software strings.



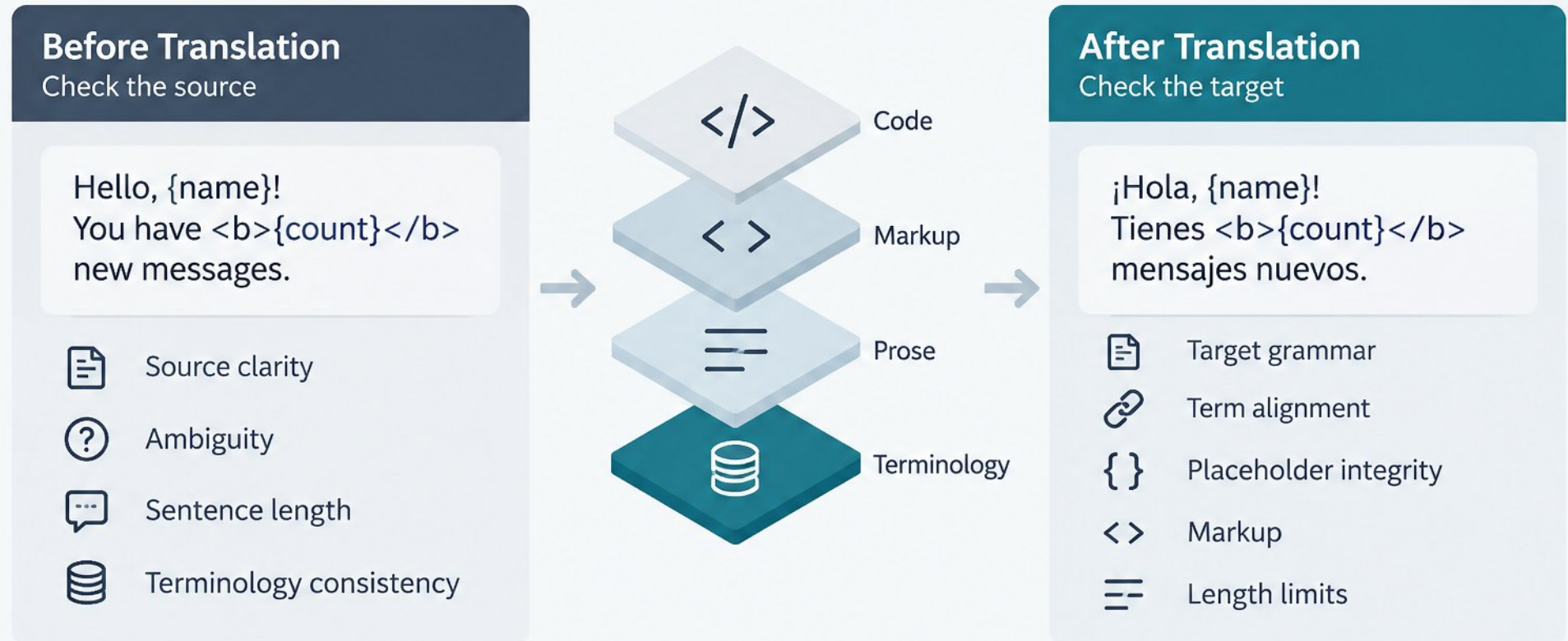
- Pair prose linting with structural validation: broken xrefs, unlisted pages, malformed code blocks, broken links, Markdown format rules.



- Gate carefully: scope failing builds to error-level issues only; blocking PRs on style noise makes teams disable the checker.



Localization Checks: Before and After Translation



Automate clear pass/fail rules: placeholders, ICU plurals, locale tags, markup, alt text



Don't transfer source assumptions—English rules don't map to other locales or plural systems



Coordinate config across docs, localization, and vendors; one named owner holds the release gate

Governance, Security, and Data Handling in Vendor Evaluation



Ask where content is processed, stored, retained, and excluded from model training—in writing



Compare deployment options: cloud with opt-out, private cloud, on-premises, fully offline



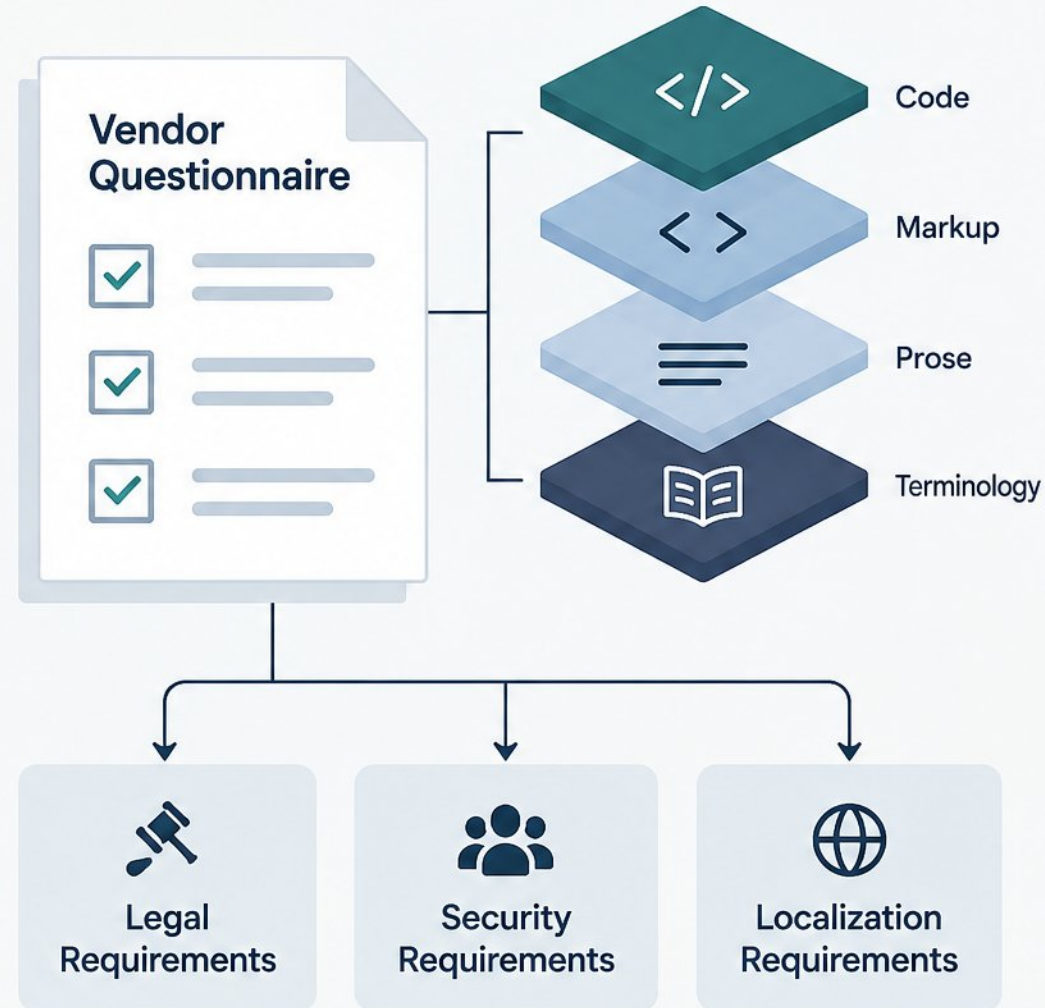
Verify SSO, role-based permissions, audit trails, and PII sanitization before data leaves



Request SOC 2 Type 2, ISO/IEC 27001, GDPR posture, and a data processing addendum



Map vendor answers to your legal, security, and localization requirements



Metrics, Pilots, and Continuous Improvement



Define outcomes: detection rate, false-positive rate per page, cycle time, rework



Pilot with representative technical content and a human-corrected reference



Score blind against the reference edits, per error type and genre



Never collapse results into one number: track precision and recall separately

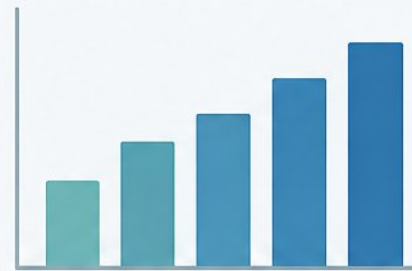


User impressions can override metrics after one spectacular false flag



Track over time as products and locales change to justify keep, replace, or invest

Detection rate by error type



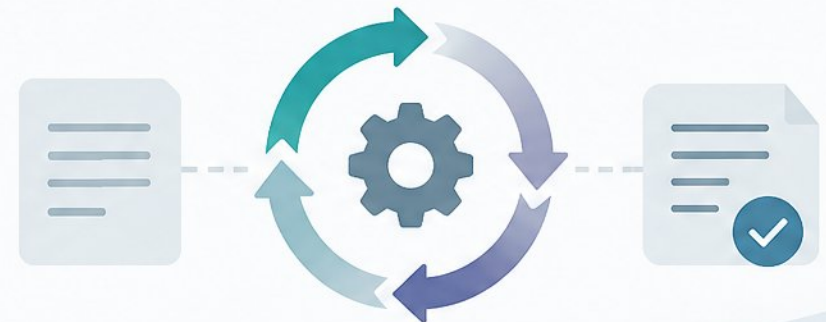
False positives per page by error type



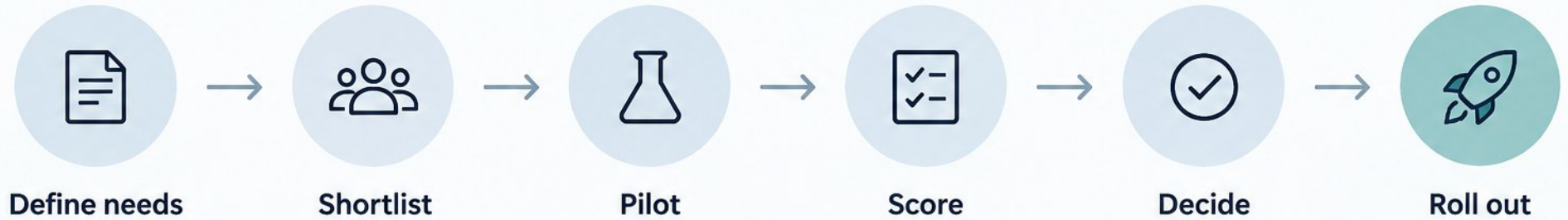
Review cycle time



Localization defect rate



Selection Framework and Workflow Blueprint



Six-step selection

- ✓ Six-step selection: define needs, shortlist, pilot, score, decide, roll out

Reusable scorecard

- ✓ Linguistic quality
- ✓ Technical handling
- ✓ Integration
- ✓ Localization
- ✓ Governance
- ✓ TCO

Workflow blueprint



- ✓ Workflow blueprint: assign each tool a distinct check at every lifecycle stage

Quick-reference checklist

- ✓ Writers
- ✓ Editors
- ✓ Content ops
- ✓ Localization leads
- ✓ Quick-reference checklist for writers, editors, content ops, and localization leads

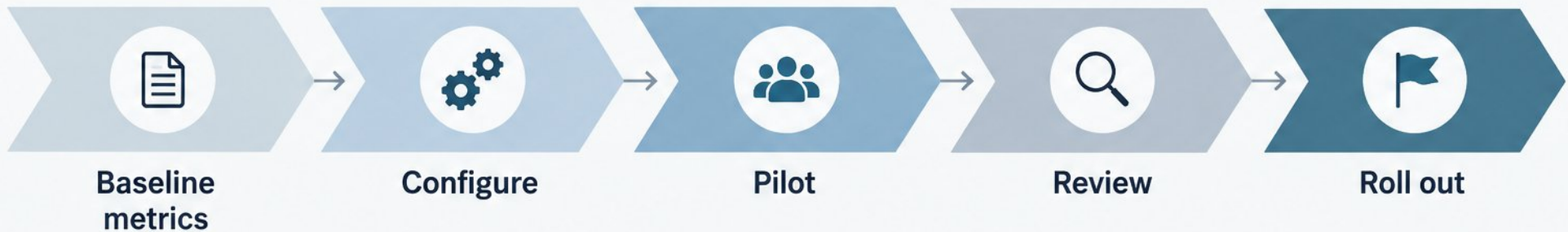


Scale the framework

- ✓ Scale the framework: solo writers need less governance than multi-locale programs

Next Steps: Pilot Plan, Owners, and 90-Day Roadmap

- Pilot scope: one doc set, one style package, four weeks
- Run one candidate checker plus the incumbent side by side
- Assign owners for rules, terminology, config, vendor, and gate
- Sequence: baseline metrics, configure, pilot, review, roll out
- Change management: training, office hours, exception log, rule proposals
- Leave with three commitments: scorecard criteria, pilot owner, go/no-go date



Owners



rules



terminology



config



vendor



gate



Leave with three commitments: scorecard criteria, pilot owner, go/no-go date

PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/4d2b5309/public