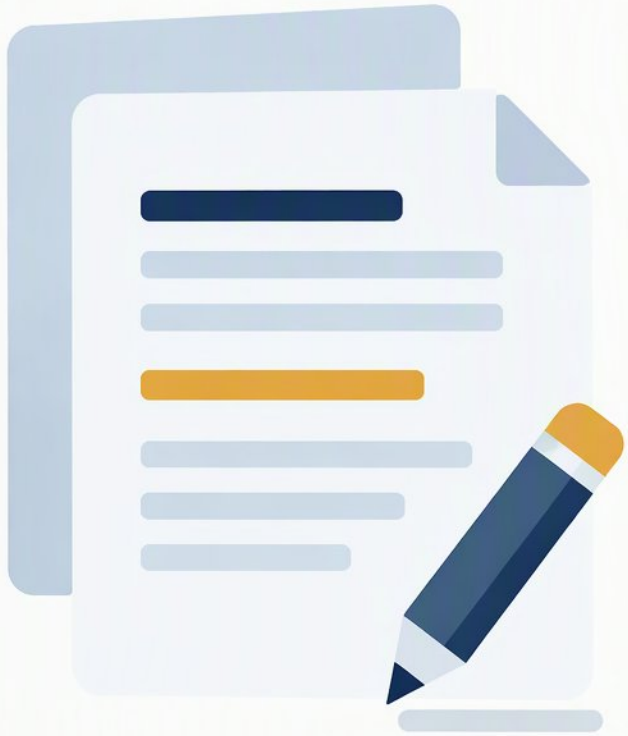
Professional Technical Writing Style Examples: Patterns, Strengths, and Pitfalls

1. Diagnose and fix style with concrete before/after examples, not rules alone
2. Style covers word choice, sentence structure, paragraph logic, tone, and information design
3. Writers, engineers, and editors share goals of clarity, accuracy, and usability
4. Roadmap: patterns → strengths → pitfalls → annotated rewrites → checklist → practice
5. Running case: an API reference entry and an installation-guide paragraph



How to Read the Examples: A Four-Part Annotation Framework

1 Pattern

Identify the recurring writing pattern at work in this example.

2 Strength

Note what works well about the sentence choice and why.

Pattern?

What works?

"The system is easy to use."

Potential pitfall?

How to improve?

3 Pitfall

Spot the potential downside or risk in the sentence choice.

4 Revision

Suggest a clearer, stronger version of the sentence.

Key Takeaways

- Every example shows four layers: pattern, strength, pitfall, revision
- Annotate three things: the sentence choice, reader effect, editorial fix
- A pattern recurs; a one-off error is not a pattern
- Five lenses: clarity, accuracy, brevity, consistency, usability
- Triage: fix accuracy and structure first, polish terminology last

Pattern Gallery: High-Value Moves in Technical Prose

Small changes make a big difference. Use these patterns to write clearer, more helpful technical content.

1 Active, direct instructions: choose "Select Save" over passive phrasing

Before	After
The Save option should be selected by you.	Select Save.

More direct and easier to follow.

2 One idea per sentence; procedures stay under 20 words per step

Before	After
First, open the file and then choose the option you need and confirm it.	Open the file. Choose the option you need. Confirm it.

Clearer, more scannable steps.

3 Front-load goals before actions: "To save the settings, click OK"

Before	After
Click OK to save the settings.	To save the settings, click OK.

States the purpose up front.

4 Use consistent terminology and parallel structure in lists and procedures

Before	After
Open the file, select an option, and then click OK.	Open the file. Select an option. Click OK.

Consistent terms and parallel structure are easier to read.

5 Replace vague qualifiers with precise, concrete nouns and numbers

Before	After
Use a few settings to improve the results significantly.	Use three settings to improve the results.

More precise and easier to understand.

Strengths in Context:
What Good Style Actually Buys You



1. Fewer support calls: 59 with quality docs versus 641 with condensed docs (GE)
2. Pilot projects: predicted 50–100 tickets became 2; a 55-per-quarter stream vanished
3. Clearer procedures and warnings support safe, correct task completion
4. Precise API and error text builds developer trust and speeds translation reuse
5. Vale linting in CI cut editing time and surfaced style issues before review (Datadog)

Pitfall Clinic, Part 1: Hidden Verbs and Noun Stacks

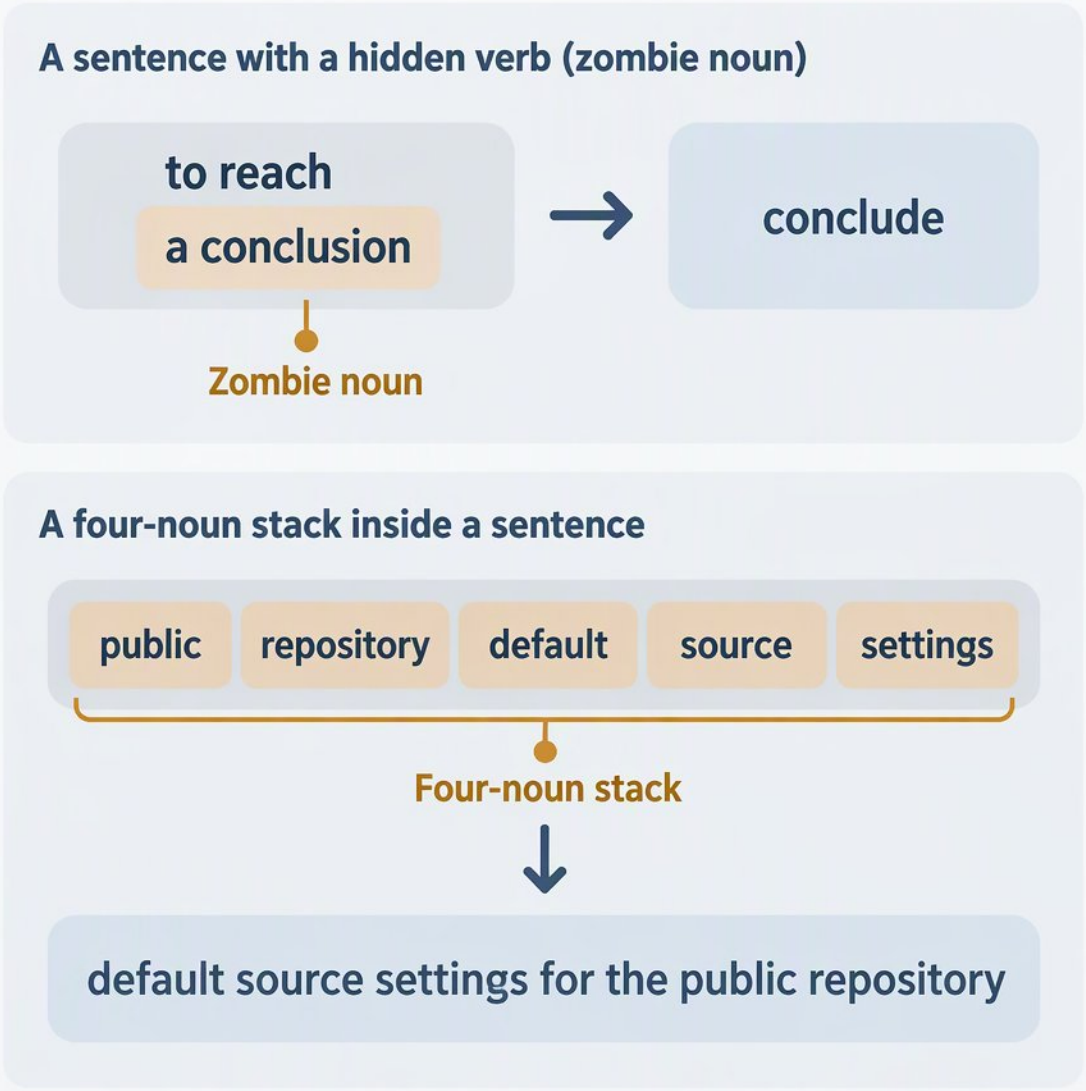
1. **Zombie nouns**
   “to reach a conclusion” → “conclude”
   measurement of temperature is done” → temperature is measured”

2. **Noun strings**
   “public repository default source settings” → default source settings for the public repository”

3. **Why they hurt**
   abstract subjects, extra prepositions, forced passive voice, untranslatable ambiguity

4. **Diagnostic habit**
   look for -tion/-ment/-ance plus a ‘to be’ verb; three or more nouns in a row

5. **Legitimate uses**
   abstract concepts like “information”; parallelism inside lists



Pitfall Clinic, Part 2: Hedging, Passive Evasion, and Buried Answers

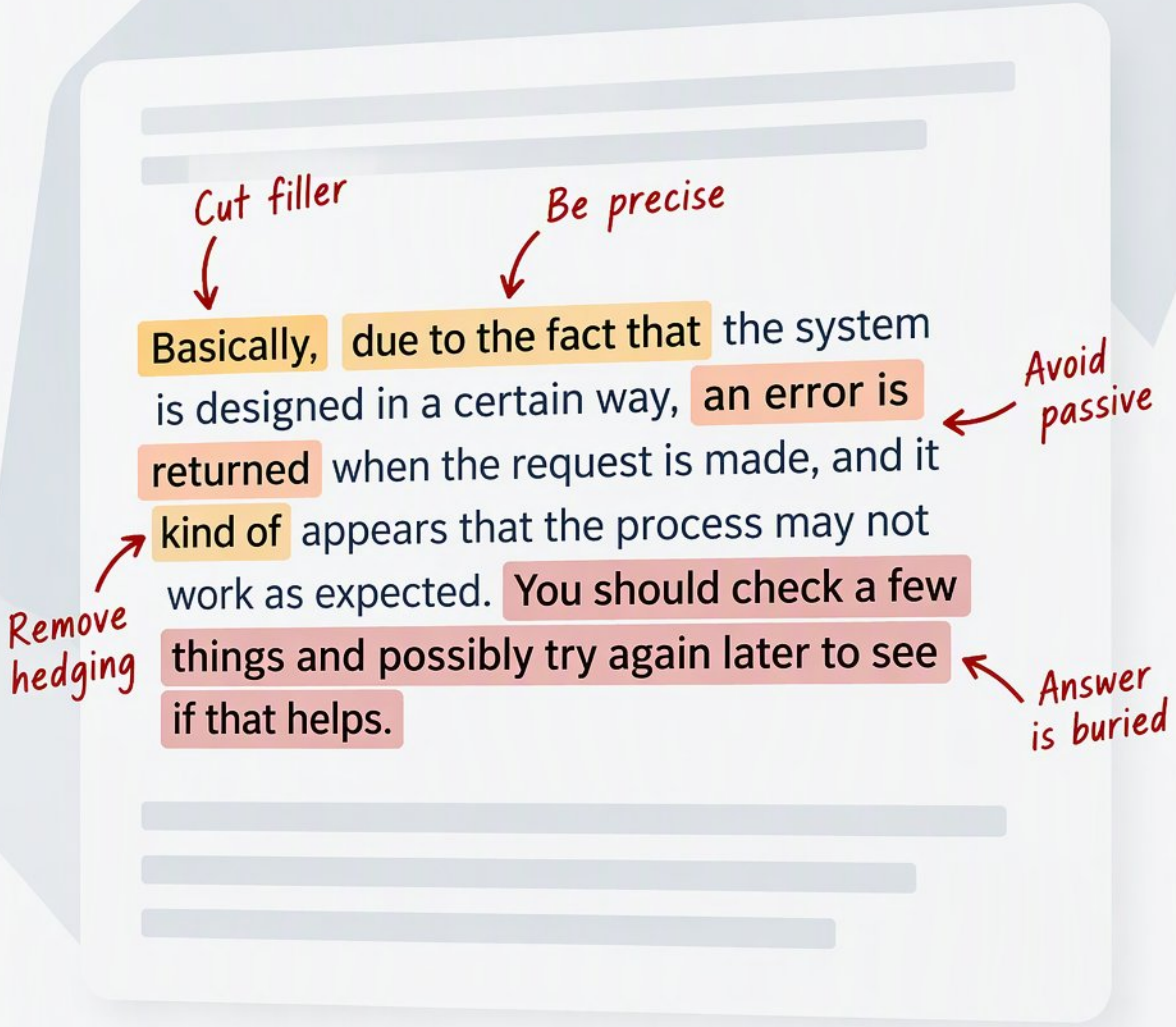
1. **Cut filler and hedging:**
   "basically", "kind of",
   "due to the fact that" →
   one precise word

2. **Passive hides the actor:**
   "An error is returned" →
   "The API returns an error"

3. **Legitimate passive:**
   actor unknown, unimportant,
   or system behavior with
   no human actor

4. **Don't bury the answer**
   under background — readers
   are hunting for the step they need

5. **Root cause:**
   drafting from the writer's
   mental model, not the reader's task



Write for the reader's task.
Clear, direct, and easy to find.

Annotated Rewrite: A Procedure Paragraph

Original: Dense and Hard to Follow

The configuration file is to be updated once the system has been stopped, as described in the administrator guide, in order to ensure that the changes are applied correctly. A backup should be made before the update is performed. After the file has been modified, the system is then started again.

- Prerequisites buried mid-paragraph
- Passive, nominalized language
- Instruction mixed with explanation

Revised: Clear, Scannable Steps

Prerequisites

- Make a backup of the configuration file.
  Stop the system.

Steps

- Update the configuration file.
- Start the system again.

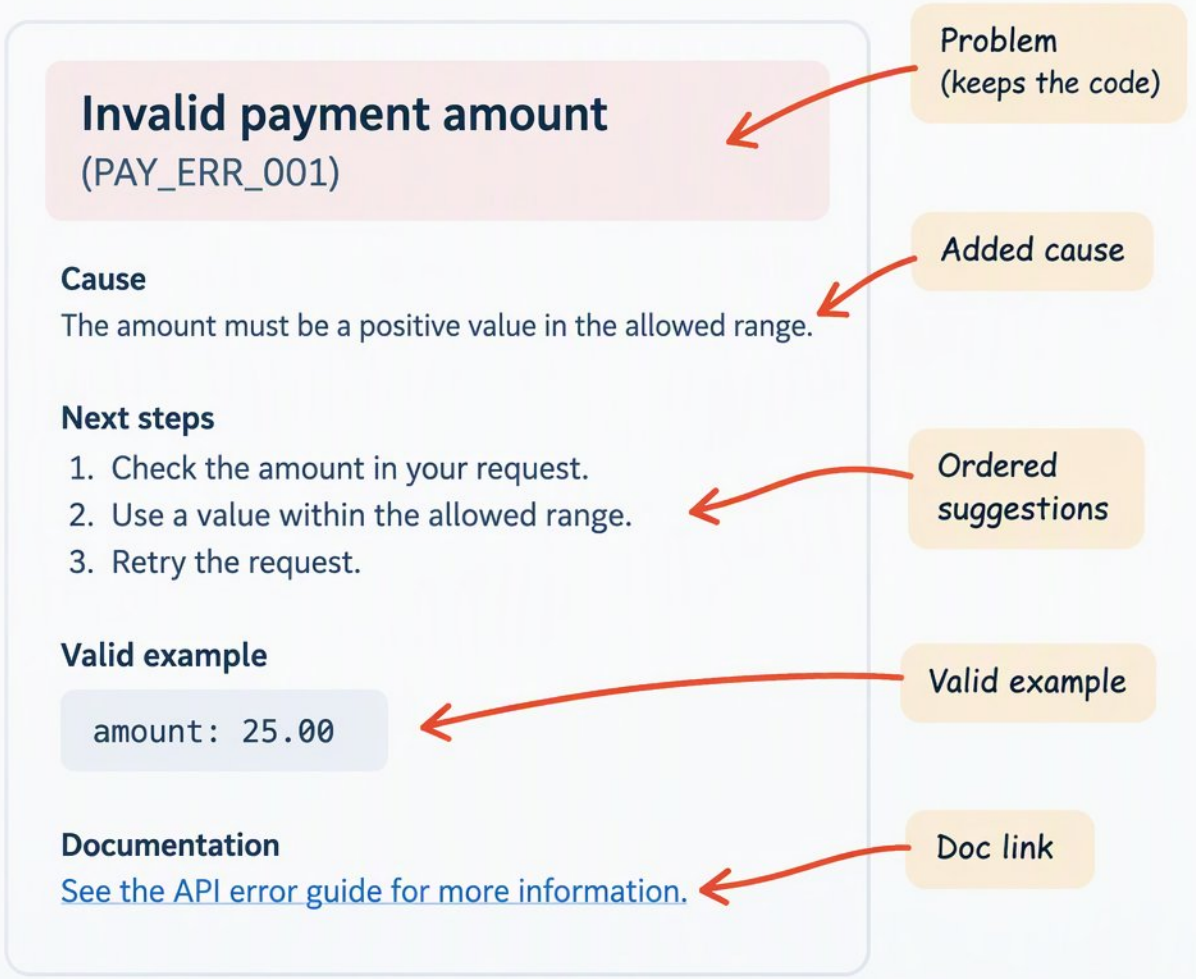
Expected result

- The changes are applied correctly.

1. Passive, nominalized original buries prerequisites mid-paragraph
2. Diagnosis: hidden actor, no expected result, instruction mixed with explanation
3. Revision: imperative steps, one action each, prerequisites first, result appended
4. Readers scan procedures in short glances, so shorter is safer
5. Compare original vs. revised phrasing and reader effect side by side

Annotated Rewrite: An API Error Description

1. Original: “Invalid payment amount” (PAY_ERR_001) — accurate but useless at failure
2. Diagnosis: no cause, no next step, no valid-value example
3. Revision pattern: problem + details + ordered suggestions + doc link + valid examples
4. Standard: AIP-193 / RFC 9457 fields — detail brief but actionable
5. **Discuss:** which added element most reduces a support ticket?



Discuss: which added element most reduces a **support ticket?**

Annotated Rewrites: Concept Explanation and Safety Warning

Concept explanation

Original
"As we saw earlier, the main point is that this approach can help in several ways. First, it improves clarity... Second, it supports consistency..."

Delays the claim with narrative transitions and buries the key point.

Revised
"This approach improves clarity and supports consistency. Here's how..."

Front-loads the claim, then layers detail progressively.

Safety warning

Original
"As we saw earlier, you must always wear the required protective equipment, and you must follow all safety procedures to avoid serious injury."

Uses narrative transitions and stacked "must," which can feel wordy and less direct.

Revised
"Wear the required protective equipment to avoid serious injury."

One imperative, actor named, consequence stated, no stacked "must."

Key takeaways

- Concept rewrite: front-load the claim, then layer detail progressively
- Cut narrative transitions like "as we saw earlier"—they block topic reuse
- Warning rewrite: one imperative, actor named, consequence stated, no stacked "must"
- Comparison: original vs. revised choices, with reader effect for each
- Discuss: when does directness become terseness for a non-expert reader?

Style Choices by Document Type



Procedures and tutorials

- Procedures and tutorials: imperative mood, sequencing, prerequisites, expected results



Reference and API docs

- Reference and API docs: consistency, parameter precision, predictable structure



Concept topics

- Concept topics: front-loaded claims with progressive detail



Release notes

- Release notes: scannable structure and impact-first phrasing; troubleshooting: symptom → cause → fix



DITA information typing separates task, concept, and reference so content stays focused and reusable

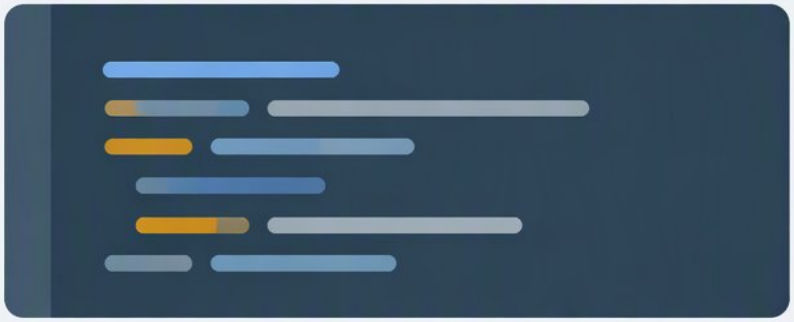


The same pattern can be a strength in one doc type and a pitfall in another

Engineer-Written Documentation: Strengths to Keep, Habits to Fix

Keep

- Keep: precision, system knowledge, real constraints, concrete code-adjacent examples



Fix

- Fix: writing for reviewers, not for readers who lack context
- Fix: assuming shared mental models and skipping prerequisites
- Fix: inconsistent naming across code identifiers, UI labels, and prose

API key ⟷ **API token**

Lightweight practices: reader-task prompts, a terminology check, a pre-review style pass

Quick win: a small Vale rule set flags terminology drift in CI (e.g., "API key" vs "API token").

Automating the Style Pass: Linting, Terminology, and CI

CONSISTENT PROSE. HIGHER QUALITY. LESS MANUAL REVIEW.



1. Vale lints prose like code: YAML rules, style folders, one .vale.ini
2. Starter stack: Microsoft or Google style, plus write-good and alex
3. Start with terminology and a reject list, before passive-voice rules
4. Linting catches terminology drift, hedging, broken links, missing alt text
5. Run in CI as a merge gate; humans still own structure and accuracy

Editor’s Checklist and Triage Workflow



Triage Workflow



Reusable checklist in five lenses: clarity, accuracy, brevity, consistency, usability

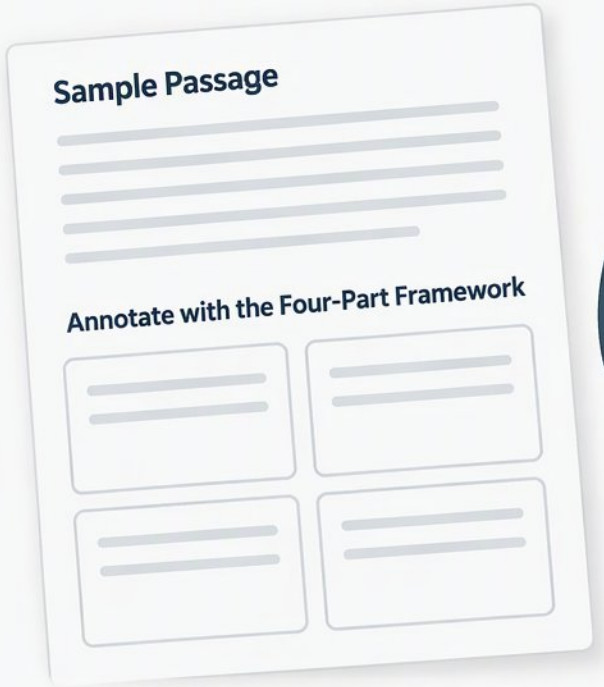
Explain changes with reader effect, not rule citation

Ask about reader task before rewriting an author's sentence

Scoring rubric separates technical accuracy review from style review

Workshop: Revise a Sample and Build Your Three-Item Plan

1. **Hands-on exercise:** revise the API error entry and installation paragraph using the checklist
2. **Peer review:** swap drafts, annotate with the four-part framework, score with the shared rubric
3. **Debrief:** which single change produced the largest reader impact?
4. **Take-away plan:** one pattern to adopt, one pitfall to cut, one linting rule to enable this month
5. **Measure before and after:** support tickets, review comments, or lint warnings





Your Three-Item Plan

Adopt	Cut	Enable

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/7f32faf7/public