Engineer

Engineering best practices

proven ways to work in a particular context

What it is

Engineering best practices are ways of working with enough evidence behind them to serve as the normal choice in a stated situation. Regularly restoring a backup in a separate environment is a good practice when a team needs to know whether it can recover. A written recovery plan, a successful backup job and a measured restore answer different questions. The practice earns its place because the restore shows what data can actually be opened, how long it takes and where someone still has to improvise.

A practice can be recommended by a research group, an engineering organization, a project or a standards body. The recommendation usually assumes certain conditions, such as a team able to deploy independently or someone able to review a change. A team can adopt it as advice, make it an explicit rule or show how it meets a contractual obligation. Those are distinct commitments. An outside recommendation becomes mandatory for a team only when an applicable obligation or the team’s own agreed rule makes it so.

NIST’s Secure Software Development Framework, version 1.1, organizes secure development practices around high-level outcomes that can be added to a software development lifecycle. It presents tasks and examples of ways to implement them. This distinction is useful outside security too. Producing reliable evidence that source changes were reviewed is an outcome. Requiring a particular button in a particular hosting service is one possible implementation. Where the outcome is explicit, another team can decide whether its review process supplies equivalent evidence and preserves the required protections.

The title “best practice” is also used formally. The IETF publishes a Best Current Practice series, with a process described in RFC 2026. RFC 8174 clarifies when the capitalized words described by its companion RFC carry special meaning in a

specification. In another context, a team may use “best practice” to mean established advice. Read who issued it, where it applies and what it asks before treating the phrase itself as authority.

Use the concept in a decision

Put “What are engineering best practices?” to work in a real decision. We can help relate the concept to the system you are building or buying. Tested, released and reversible, so changes reach customers on a Tuesday afternoon rather than late on a Friday night.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Automated GIS SaaS deployment with GitHub Actions — deployment, data and reporting, all off the hands

Automated GIS SaaS deployment with GitHub Actions Get in touch →

Why it matters

A sound default saves people from settling the same question on every change. A new developer sees the expected evidence before opening a review. Operators get comparable information from one release to another, and teams sharing a service can agree on which actions warrant notification. The saved time depends on the practice being clear enough to apply. “Review carefully” leaves people to invent different criteria. Asking someone to demonstrate that a customer cannot be charged more than once gives the review something observable to agree on.

Defaults also make exceptions visible. Consider a team that ordinarily releases small changes frequently. A device firmware update may require a field technician to isolate a machine, apply the update and verify the device at the site. The small-change principle can still influence the design, test sets and rollout order. The frequency of the physical release changes because the conditions of recovery are different. Stating the reason helps people keep the part that still reduces risk.

Evidence is also local. DORA describes trunk-based development in terms of integrating small changes frequently and identifies capabilities and measures teams can use to examine delivery. That is a practical lead for a team with long-lived branches, slow integration and recurrent merge work. The team still needs to observe its own integration time, rejected changes and recovery behavior. A smaller branch count alone may simply show that the team stopped counting some branches, or began combining more work behind each merge.

What it looks like in practice

Imagine a team that maintains an internal reporting tool. Pull requests must have a second reviewer and the shared test suite must pass. This rule usually catches omissions while changes are cheap to correct. When a monthly report is late, the usual test suite spends several hours replaying fixtures that cover an unrelated export path. The proposed fix changes only the report's filter and is reversible before the report is distributed.

The team first checks its actual obligations. If approval before release is required, a faster route still needs authorized approval. It then names the relevant risk. The urgent fix must include the correct records, keep the previously agreed grouping and avoid exposing rows from other business units. The author runs focused tests with known input and compares the generated report against expected rows. A separate reviewer inspects the diff and checks those results. Someone is named to decide whether this evidence is sufficient, and the previous report is kept ready in case the comparison fails. The full test suite runs afterward and will inform whether the team wants a faster permanent check.

The short exception record states the normal rule, why the case qualifies, which protections still apply and when the exception expires. It records the actual elapsed time and outcome. If the team needs the same exception again the following month, it can move the report-specific assurance into its regular workflow. Before changing the default it checks that the faster route can detect a wrong date filter and a deliberately excluded business unit. A check that fails on those seeded errors supplies stronger evidence than a timer alone.

Engineer's case on a commit gate of 22 linters with no warning tier describes making a project's agreed checks enforceable at commit time. A different accessibility case describes defects encountered despite automatic checks. The two examples expose different questions. A gate can enforce a rule consistently while the underlying test still covers only part of what users experience.

Where it goes wrong

A checklist becomes the goal. A process marked complete can still produce the failure it was supposed to prevent. Choose one sample where the practice should clearly make a difference and observe it. Check that the reviewer can reconstruct the change, that the restore opens usable data or that the release actually reaches the intended users.

A tool is mistaken for the practice. Buying a review platform does little if nobody has enough time or context to disagree with a change. Write down the outcome that the tool is supposed to support. Then inspect a real example with the people who use it and find out whether the promised information is available in time.

A local exception becomes a quiet default. A workable emergency path tends to get reused. Give exceptions a stated scope, an accountable person, a compensating measure and a review point. Repetition is useful evidence that the normal route deserves attention. Improve it openly when that evidence holds.

A team's evidence is copied without its context. A rule can work under different deployment, staffing or product constraints. Start with the problem described by the source and compare it with yours. Try a bounded change, decide in advance what improvement and harm would look like, then keep the practice when the observations support it.

Further reading

- NIST Secure Software Development Framework, version 1.1 — the framework's practices, tasks and examples of implementation.
- RFC 2026, section 5 — what Best Current Practice means within the IETF's standards process.
- RFC 8174 — when capitalized requirement words have their defined meaning in an IETF document.
- DORA on trunk-based development — small changes, frequent integration and ways to assess the practice.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What are engineering best practices?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/engineering-best-practices/