The Playbook

Seven moves to make in your first six months of KCS — for small and emerging teams

Program playbook · For team leads, KDEs, and KCS program management · Nick Morgan · May 2026

The premise.

The first-cohort teams built the KCS practice by crashing and burning their way to what works. They paid the tax that turned KCS from concept into discipline. They:

- Sampled cases without templates
- Scored articles without taxonomy
- Coached engineers without coach training
- Ran PAR cycles before PAR had a workbook

Newer teams should not have to pay that tax again.

This is KCS applied to KCS itself: capture the practice once, reuse it many times, adapt it to context.

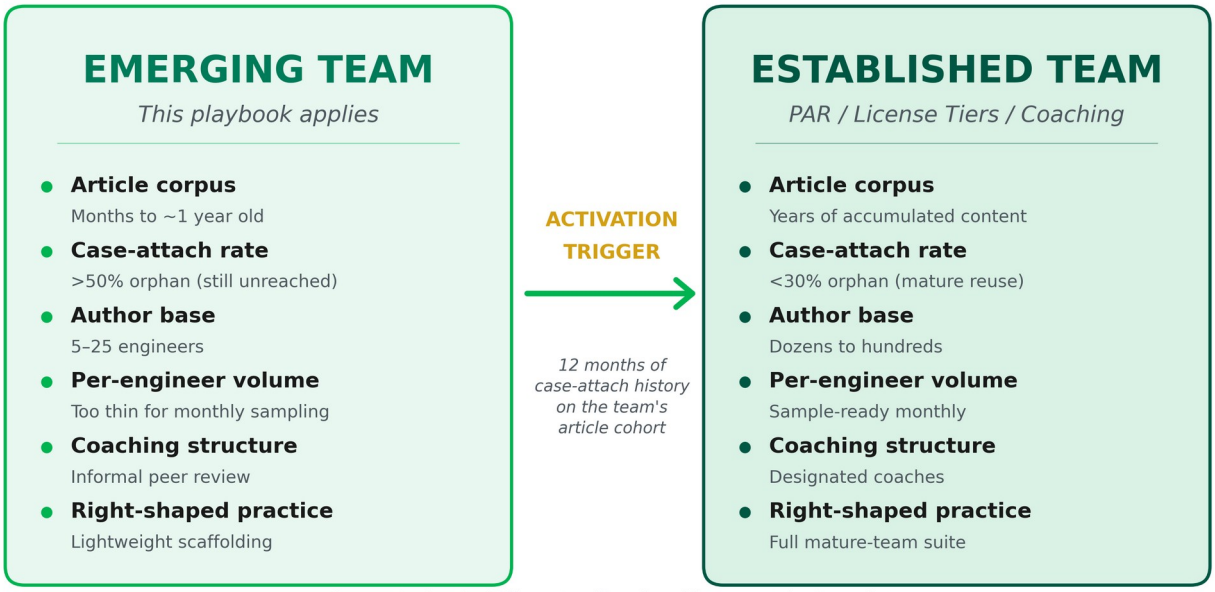
The hard work of figuring out what works at scale is done. What new teams need is a different question — not “how do we invent KCS practice,” but “how do we adopt the version that already exists in a way that fits where we actually are.”

Who this is for. Any customer technical support team in early KCS maturity: a young corpus, a small author base, no PAR history. Tomorrow it's the next emerging product line; today it's whichever team you're standing this up with.

The playbook below is the same in each case — the data calibration changes, the moves don't.

I. Where you are — a 30-second self-assessment

If your team matches the emerging-team profile below — a young corpus, a small author base, too little per-engineer volume for monthly sampling — this playbook is for you. If it matches the established-team profile instead, you've graduated, and the mature-team practice (PAR cycles, License Tiers, coaching structure) applies and is what you should be running.



Same playbook. Different calibration. Movement is the point.

> *Maturity is per-team, not per-program.*

Applying late-stage practice to an early-stage team isn't rigor — it's process-as-theater, and the team will correctly recognize it as such. Stage-matched practice respects the data and builds trust in the program.

II. The playbook — seven moves to make in your first six months

Each move below stands on its own. Run them concurrently — none of them blocks any other from starting. The numbering is for reference, not sequence.

1. Stand up a team-specific dashboard before doing anything else

What it is. A self-contained, monthly-refresh dashboard scoped to your team's product KB(s). Not a leadership reporting artifact — a working tool for the team.

Minimum tabs:

- Overview with corpus shape
- Top issues by use_count
- Emerging issues (articles climbing in the last 90–120 days)
- Components / subsystems
- Customer-journey stages

Why this is move #1. It changes the conversation from “what scores are we measuring you on” to “here is the data; what do you make of it.” That reframing is what makes everything else in this playbook land.

2. Run a monthly “what landed?” stand-up, 30 minutes, no slides

Who. Engineering manager plus 2–3 senior engineers, dashboard open live.

Two questions. What is appearing in the emerging view? What surprised us?

What it isn't. No KPIs, no rollup to leadership, no slide deck. This is the KDA-equivalent for teams that aren't yet at formal-KDA scale — it builds the muscle of looking at signal regularly without imposing the formal-KDA overhead.

Why monthly, not quarterly. For a young KB the data shifts fast enough that quarterly cadence misses most of what's happening. Monthly costs 6 hours per year and catches signal you'd otherwise miss.

3. Designate one Knowledge Domain Expert — informally

What it is. Identify the person who is already the de-facto domain authority — usually the engineer with the highest article count or the highest article-use ratio. Make the role explicit, light on accountability, mostly the named person for “where does this knowledge belong?” and “is this article shaped right?” questions.

Watch out for. Do not establish a Knowledge Worker tier below the KDE yet — that emerges from volume the team doesn't have, and trying to layer it prematurely creates titles without function.

4. Publish a one-page domain-specific content supplement

What it is. The enterprise content standard applies as the base for every team. The supplement is a single additive page on top of it — not a rewrite.

What the page covers:

- Title patterns that are findable for your specific customer audience
- When to author a hub article vs. a leaf article
- How to handle backend- or platform-specific symptoms in titles
- Any vocabulary conventions unique to your domain

Why this matters even in month one. Title-pattern problems compound over time. A young KB's top-use article is often a generic title that snared traffic by accident — exactly the kind of early-corpus problem the supplement is designed to prevent. Fix it before the team has 2,000 articles, not after.

5. Build a recognition loop, not a scoring loop

What it is. When an article climbs into the emerging-issues view, surface it in a team channel or weekly digest with the author named. Low effort, high signal.

The inverse of PAR. PAR measures whether engineers handled existing knowledge well; recognition measures whether the knowledge they wrote landed. For a team in this phase the latter is the right reinforcement mechanism, because the former is barely formed.

6. Treat the dashboard's monthly refresh as a content event

The cycle. Each month: pull fresh data, push the updated dashboard, briefly note in the team channel what climbed or what's new.

Why this works. The cadence keeps the tool feeling living rather than static, builds the habit of looking at signal, and gives the team an artifact to point to when leadership asks how things are going — without forcing leadership to read engineering-level detail.

7. Communicate the activation trigger for mature-team practice — publicly

What it is. Tell the team explicitly when you will move from this playbook to the mature-team practice (PAR cycles, License Tier progression, formal coaching).

The trigger. Data-driven, not calendar-driven: the team's article cohort reaches ≥80% adoption rate (% of articles with non-zero use_count). For most emerging teams that takes 9–18 months from the playbook starting, depending on team velocity and product visibility.

Why publicly. Naming the trigger signals respect for the team's current stage (no premature judgment) and creates a concrete forward expectation they can plan against. It also defuses the natural anxiety that comes with “you're not in PAR yet” — the team knows when and why they'll be.

III. What to leave on the bench

Things to specifically avoid imposing on an emerging team. Each one looks reasonable in isolation, and each one would damage the program if applied prematurely.

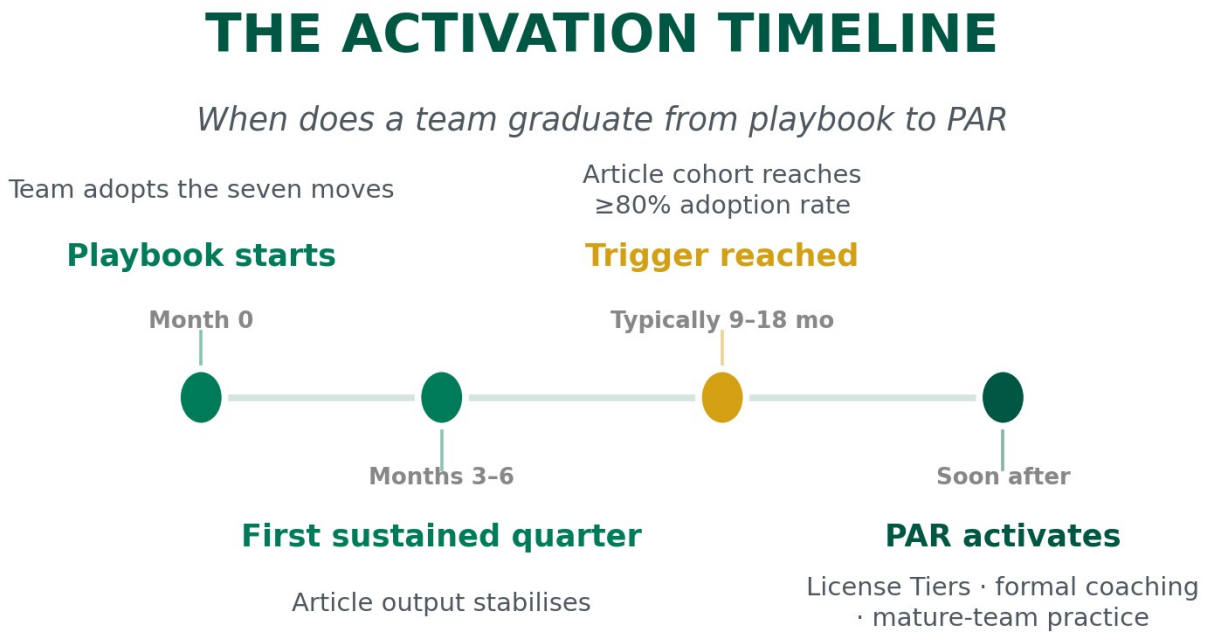
Detail on each:

- **Formal PAR cycles.** PAR assumes per-engineer monthly case sample, a mature

article corpus to reuse from, and a coach-engineer relationship graph. Emerging teams have none of these. PAR sampling on early-stage data measures noise, and the team will correctly recognize that the metrics are illegitimate. Avoid “PAR-lite” or modified PAR variants too — better to say “not yet” cleanly than to dilute the model with a watered-down version.

- **Reuse Accuracy scoring.** Same issue: the denominator (articles being reused) is barely formed for an emerging team. Scoring against it produces statistical noise dressed up as a metric.
- **License Tier progression (Candidate / Contributor / Publisher / Coach).** Designed for per-engineer volume that emerging teams don't have. Trying to assign tiers to a 5–25-person team where most are writing their first articles is overhead without payoff. License Tiers activate alongside PAR — same trigger, same timeline.
- **Formal coaching structure.** Coaching as a defined role emerges from case volume — there has to be enough handling to coach against. An informal “senior reviews newer” practice works at this scale; titled coaches don't.
- **Cross-team metric comparison against the established portfolio.** Comparing an emerging team's reuse rate (when it eventually has one) to an established team's would be apples-to-oranges — the corpus age, case volume, and audience are different. Hold off on cross-team comparison until the emerging team has graduated into the same maturity band.

IV. When you graduate — what changes



Once the activation trigger is met — when your team's article cohort reaches ≥80% adoption (use_count > 0) — the mature-team practice activates. Concretely:

Element	Emerging-team posture (this playbook)	Mature-team posture (graduation)
Primary feedback mechanism	Dashboard + monthly stand-up + recognition loop	PAR cycles, monthly per-engineer sample, formal coaching
KDE / KW structure	One informal KDE, no formal KW tier	Designated KDE plus KW tier with role-defined responsibilities
Coaching	Informal peer review	Designated coaches with assigned engineers; PAR-

		integrated coaching
Metrics applied	Use_count, emerging-article rate, author recognition	Reuse Rate, Knowledge Loss, Solve Loop Index, Reuse Accuracy
License Tier framework	Not applied	Engineers progress Candidate → Contributor → Publisher → Coach
Cross-team rollup	Excluded from cross-team comparisons	Included in portfolio-wide rollups (PAR Leadership, Coaching View)
Dashboard role	Team's primary working tool	Complement to PAR, not replacement

The transition isn't abrupt. The dashboard the team has been using through the emerging phase stays in place — it becomes the demand-side complement to PAR's handling-side view. The KDE role they've been doing informally becomes a formal program role. The monthly stand-up either continues (if it's working) or folds into the PAR review rhythm.

> *Graduation isn't an end-state. It's an entry into a different phase of work.*

The mature-team practice is itself evolving — Phase 1 to Phase 2 indicators are still being defined for the established teams. Graduation means joining that conversation, not finishing one.

V. Worked examples — two emerging teams in this posture

Two real applications of the playbook, anonymized. They are deliberately different in size and shape so the calibration point comes through.

WORKED EXAMPLES

Same playbook, calibrated to the data each team actually has

EXAMPLE A

Single-product team, young technical domain

SHAPE

~500 articles · ~440 total uses · ~14 authors. Median article birth in the most recent quarter. Highest orphan rate in the portfolio. Distinct domain vocabulary unlike any other team.

POSTURE

Full playbook applies. Designate one KDE; informal peer review; recognition loop on emerging articles. Domain-specific content supplement to be authored.

ACTIVATION TARGET

~18 months out — once the current article cohort has 12 months of case-attach history.

EXAMPLE B

SaaS product family with multiple sub-products

SHAPE

~850 articles · ~2,900 uses · ~90 authors. Four sub-products with one dominant member carrying ~80% of articles and use.

POSTURE

Full playbook applies. Larger author base means a Knowledge Worker tier could emerge sooner, but probably still does not earn formal definition until per-product corpora mature.

ACTIVATION TARGET

Same approximate window as Example A.

The moves are universal. The numbers are local.

More detail on each below:

Example A — single-product team in a young technical domain

Shape. Roughly 500 articles, ~440 total uses, ~14 authors. Median article birth in the most recent quarter. Highest orphan rate in the portfolio. Single-product KB, distinct domain vocabulary unlike any other product team.

Posture. Full playbook applies. Designate one KDE; informal peer review; recognition loop on emerging articles. Domain-specific content supplement to be authored.

Activation target. Once the article cohort reaches ≥80% adoption — typically 12–18 months for a team at this size.

Example B — SaaS product family with multiple sub-products

Shape. Roughly 850 articles, ~2,900 uses, ~90 authors. Four sub-products with one dominant member of the family carrying ~80% of articles and use.

Posture. Full playbook applies, with one nuance — the larger author base (90 vs. Example A's 14) means a Knowledge Worker tier could emerge sooner than in a smaller team, but probably still doesn't earn formal definition until the per-product corpora mature.

Activation target. Same approximate window as Example A.

Both teams demonstrate the principle: same playbook, calibrated to the data each team actually has. The moves are universal; the numbers are local.

VI. Resources and where to start

- **Your dashboard.** Ask your Knowledge Program Manager or your team's leadership to provide one. Standing up the dashboard, configuring the data extraction, and hosting it are program-level responsibilities — not work the team itself has to do. What the team does need to do is use it once it's there (move #1 of the playbook). If your organization doesn't yet have a Support → Dev dashboard pattern in place, that's the first conversation to have with program management before applying this playbook.

- **Your content supplement.** Your organization's enterprise content standard is the base. Your one-page supplement should cover: title conventions for your specific customer audience, when to author a hub vs. leaf article, how to handle backend or platform-specific symptoms in titles, and any vocabulary unique to your domain. The KDE writes it; KCS program management reviews and signs off.

- **Activation trigger criteria.** The trigger is your article cohort reaching ≥80% adoption (article use_count > 0). Practically, that puts PAR activation 9–18 months after the playbook starts, depending on team velocity. Track the adoption rate on your dashboard; once it crosses 80% reliably, you're ready.

- **Where to go for help.** Your Knowledge Program Manager owns the program-level scaffolding (dashboard, content standard, PAR cadence, License Tiers) and is the right escalation point when anything in this playbook needs interpretation against your team's specific data. If your organization doesn't have a designated KPM, the role most likely to be holding this responsibility is your Knowledge Manager, Support Operations lead, or the most senior KCS-trained engineer.

The principle worth carrying forward

The KCS practice was built by teams who paid the tax of figuring out what works. The job of program management now is to make sure newer teams don't pay that tax twice. The playbook above is what the established teams wish they'd had in their first eighteen months — distilled from what they learned the hard way, scaled to fit a younger team's reality. Reuse what works. Adapt to context. Defer what doesn't

fit yet. Graduate when the data warrants it.