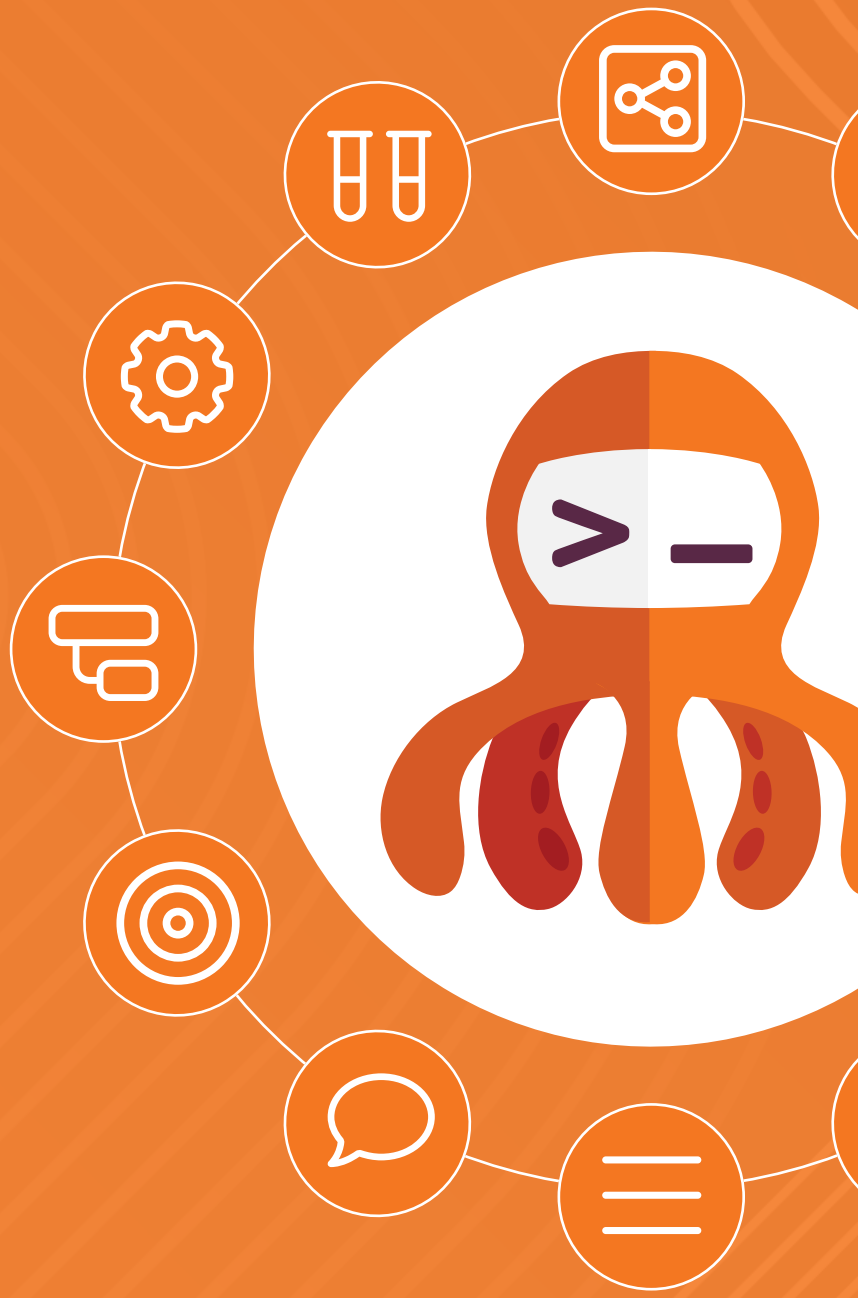
THE GREAT FILTER: A MANIFESTO FOR HUMAN-LED, AI-POWERED SOFTWARE DEVELOPMENT

An Article for R&D Leaders on Transforming AI from a Tactical Gadget into a Strategic Asset.



By TIKAL

Introduction

AI coding tools promise a revolution in engineering velocity. Yet for most R&D leaders, the reality is a mix of inconsistent results, ad-hoc prompting, and a troubling lack of governance.

Developers are either using AI as a glorified autocomplete or getting lost in its non-deterministic nature, creating code that lacks context and architectural cohesion.

The problem isn't the technology; it's the absence of a scalable process. Treating AI as a simple productivity shortcut is a strategy destined for failure. To unlock its full potential, AI cannot just be a tool for developers; it must be integrated into a structured, governed system for development.

Based on our work with dozens of engineering teams and insights from our AI Task Force, we developed the **Twelve-Factor Agentic SDLC**. This methodology provides a clear framework to transform AI from a tactical gadget into a strategic asset that accelerates delivery while enhancing quality and control.

Why Most Teams Struggle to Adopt AI Coding Agents

The promise of AI-assisted development is real: faster feature delivery, reduced boilerplate, and scalable testing. But the day-to-day reality often looks different.

Developers bounce between vague prompts and off-target results. Code lacks context. Teams can’t agree when to collaborate with AI versus when to delegate. And quality control? Inconsistent at best.

> The problem isn’t the tech. It’s the process — or lack of one. Most teams treat AI agents like an advanced search engine or productivity shortcut. But scaling them requires a deeper shift in how development is structured.

The Twelve Factors & Their Practical Application

The Twelve-Factor Agentic SDLC provides a structured framework for integrating AI agents into modern software development. Each factor is paired with practical, team-level actions taken directly from real-world implementations.

1 | Strategic Mindset

Developer as Orchestrator, AI as Intern

Treat AI as a fast, knowledgeable junior partner that requires clear direction, mentorship, and rigorous review.
Adopt the fundamental mental model: you're the experienced engineer orchestrating an intelligent but junior partner. The agent is fast and knowledgeable but lacks experience, taste, and deep context.

- **Assess Your Position:** Use the AI Collaboration Matrix to understand your maturity level. Are you primarily "Vibe Coders" experimenting with prompts, or "AI Newbies" just getting started?
- **Define Your Goal:** Set a clear improvement objective (e.g., "Move from vibe-based prompting to consistent async delegation workflows this quarter").
- **Review Common Pitfalls:** Check for team-level issues like "No feedback loop" or "No context provisioning."
- **Assess Competency:** Evaluate prompt engineering skills across your team (Novice, Practitioner, Expert, Master).

2 | Context Scaffolding

Treat Context as a Dependency

Manage all context—code, documentation, and even the AI models themselves—with the same rigor as a critical software library.
Manage agent context with the same rigor as code dependencies. Assemble project data, team knowledge, and external documentation before prompting.

- **Assemble Local Context:** Use your IDE's integration to surface project-specific code, config, and structure.
- **Assemble Team Context:** Connect to your team's internal knowledge systems and guidelines.
- **Perform Smart Search:** Consult trusted documentation sources or developer-focused knowledge retrieval tools to augment agent input. (Note: tool names evolve quickly — focus on capabilities, not brands.)

3 | Mission Definition

Start with a Formal Brief

Initiate every complex task with an explicit Mission Brief that defines goals, constraints, and success criteria before writing the first prompt.

Every complex task begins with a standardized Mission Brief containing clear goals, constraints, and success criteria.

- **Create a Mission Brief:** Fill out a short, structured brief before prompting.
- **Define Clear Goals:** One clear sentence describing what's being built.
- **Specify Constraints:** List must-have tools, patterns, or limitations.
- **Define Success Criteria:** Make "done" unambiguous for both developer and agent.

4 | Structured Planning

Decompose and Triage Tasks

Use AI to generate a detailed execution plan, which the human developer reviews, approves, and triages into synchronous (interactive) and asynchronous (delegated) sub-tasks.

Use the agent to help generate a plan, then decide what to handle interactively and what to delegate.

- **Generate the Plan:** Ask your IDE agent to create a step-by-step execution plan.
- **Review and Approve:** Refine the plan with human judgment.
- **Triage the Plan:** Label each task as [SYNC] for hands-on collaboration, or [ASYNC] for autonomous completion.

5 | Dual Execution Loops

Pair Program or Delegate Toil

Master two distinct workflows: real-time synchronous collaboration for complex problems and asynchronous delegation to autonomous agents for well-defined tasks.

Use the right mode for the task — pair programming for complexity, async execution for scale.

- **Execute [SYNC] Tasks:** Use your IDE's interactive loop (Plan → Act → Reflect → Correct).
- **Execute [ASYNC] Tasks:** Delegate to a background agent using the mission brief.

6 | The Great Filter

Apply Irreplaceable Human Judgment

The developer is the ultimate arbiter of quality, filtering all AI output for correctness, architectural cohesion, security, and domain-specific taste before it enters the codebase.

AI doesn't commit code — people do. The developer is the ultimate quality gate.

- **Commit Synchronous Work:** Finalize interactive output only after manual validation.
- **Review Asynchronous Work:** Conduct full PR reviews with context and accountability.
- **Use AI-Assistive Reviewers:** Let them highlight issues, but keep decision-making human.
- **Perform the Final Merge:** Always a developer's responsibility.

7 | Adaptive Quality Gates

Review Appropriately for Each Workflow

Implement continuous micro-reviews for synchronous work and formal, automated macro-reviews (e.g., Pull Requests, CI pipelines) for asynchronous work.

Quality control depends on the workflow mode — adapt accordingly.

- **Micro-Reviews:** For [SYNC] work, continuously check small code chunks in real time.
- **Macro-Reviews:** For [ASYNC], require CI pipelines, tests, and structured PR descriptions.
- **Use Evaluation Suites:** Validate functional correctness and maintainability — not just coverage

8 | Risk-Based Testing

Use AI to enable risk-based testing

Focus testing where it matters most—on business and security risks—and let the agent scale the effort. This approach makes a mature testing strategy practical by shifting the developer's role from writing repetitive test code to strategically identifying potential failure points.

The AI then handles the tactical work of generating the specific and targeted tests needed to validate against those risks, ensuring the final product is robust where it counts.

- **Define Critical Risks:** In every Mission Brief, developers explicitly list the business, security, or performance risks that must be addressed.
- **Command Targeted Tests:** Instead of generic requests, developers instruct the AI to generate tests for the specific risks identified (e.g., "assert unauthorized access fails," "verify performance under load").
- **Verify Test Effectiveness:** The developer remains responsible for running the AI-generated tests and confirming they provide meaningful validation, acting as the final quality filter.

9 | Structured Traces

Document the 'Why,' Not Just the 'What'

Generate and preserve structured execution traces—the prompts, tool calls, and reasoning steps—as the primary artifact for debugging and process improvement.

This principle is put into practice through a central, version-controlled library: the team-ai-directives repository. This repository acts as the single source of truth for consistent agent behavior and is the collective AI-related intelligence of the engineering team. It houses reusable, versioned modules that are consumed by the central platform, including:

- **Link Traces to Issues:** Store prompt history, decisions, and outputs with task records.
- **Ensure Clear PR Descriptions:** Make agent-generated PRs explainable and auditable.
- **Promote Transparency:** Structured traces support onboarding, debugging, and retrospectives.

10 | Strategic Tooling

Manage a Federated, Governed Stack

Treat your team's prompts, behavioral instructions, and system templates as a version-controlled asset, managed with the same discipline as application code.

Think in capabilities — not brand names. Use modular tools fit for sync or async work, with governance in place.

- **Govern Access via Gateway:** Route agent traffic through a central service for model control, security, and budget management.
- **Map Tools to Roles:**
  - Context search & RAG: For documentation lookups
  - Interactive IDE agents: For exploratory coding
  - Autonomous agents: For [ASYNC] delivery

11 | Directives as Code

Version and Share AI Behavior

Treat your team's prompts, behavioral instructions, and system templates as a version-controlled asset, managed with the same discipline as application code.

Prompts and templates are infrastructure. Treat them like code.

- **Pull the Latest Directives:** Always start with the most current shared formats.
- **Use Templates Consistently:** For common flows like component creation or writing PRs.
- **Contribute Learnings Back:** If it works — share it via PR into the team directives repo.

12 | Team Capability

Systematize Learning and Improvement

Build organizational muscle memory by formalizing the sharing of best practices and using a versioned suite of evaluations (Evals) to objectively measure performance.

Agentic workflows aren't plug-and-play — they require ongoing coaching, reflection, and iteration.

- **Share Learnings Across the Team:** Use async channels, guilds, or forums.
- **Run AI-Focused Retrospectives:** Discuss workflow friction, not just delivery metrics.
- **Maintain Shared Memory:** Use progress.md, activeContext.md, or equivalent to keep knowledge visible.
- **Benchmark with Evaluations:** Use golden test cases or historical prompts to measure progress over time.

From Principles to Practice: The Four-Stage Workflow

This four-stage workflow translates the twelve factors into a practical, day-to-day process, moving from ad-hoc prompting to structured, high-velocity collaboration. This is how you leverage AI as a capable "intern" with you as the "orchestrator."

The developer is the ultimate arbiter of quality. You own the final git merge. Accountability is not transferable.

Stage 1: Mission Prep: Briefing & Context

- **Assess the Task:** Quickly determine if the task is best for real-time collaboration [SYNC] or if it's a well-defined chunk of work suitable for delegation [ASYNC].
- **Create a Mission Brief:** In a scratchpad or issue, formally define the task. Don't start prompting without it. Include:
  - **Goal:** A clear, one-sentence objective (e.g., "Implement the user authentication endpoint").
  - **Constraints:** Non-negotiable requirements (e.g., "Must use FastAPI," "No breaking changes to the User model").
  - **Success Criteria:** How you will know the task is "done" (e.g., "All existing tests pass, and new tests for the endpoint are added and passing").
- **Assemble Context:** Give your AI intern the right documents to succeed.
  - **Local Context:** Use your IDE's features (@codebase, @files) to make the AI aware of your current project.
  - **Team Context:** Pull in your team's best practices using your organizational knowledge system (@team).
  - **External Context:** Use a smart search tool (e.g., Sourcebot, @web) to find external patterns or documentation before asking your primary AI to write code.

Prompt Template:

```
I need to create a new Mission Brief in
our issue tracker for the objective:
"{YOUR_OBJECTIVE}".
- Propose a one-sentence **Goal**,
measurable **Success Criteria** and
define the key **Constraints**.
- Using @web, research best practices
for this task.

Help me choose the best components from
our team's library for the Context
Packet.

- Using `@team`, search `/
context_modules/personas/v1/` and
suggest the most appropriate persona.

- Using `@team`, search `/
context_modules/rules/v1/` for relevant
style guides and security rules.

- Using `@team`, search `/
context_modules/exemplars/v1/` for a
high-quality example related to my
objective.
```

Stage 2: The Core Loop: Plan, Triage, and Execute

- **Generate the Plan:** Using your Mission Brief, ask your Agentic IDE: "Generate a detailed, step-by-step plan to achieve this mission."

- **Review & Triage the Plan:** Critically review the AI's plan. Modify it as needed, then for each step, add a tag: [SYNC] for interactive work or [ASYNC] for delegation.

- **Execute the Plan:**
  - **For [SYNC] Tasks (Developer + AI Pair):** Use your Agentic IDE in a tight, interactive Plan-Action-Reflect-Correct (P-A-R-C) loop.

  - **For [ASYNC] Tasks (Delegate to AI):** Delegate the task and its context to a specialized autonomous agent (e.g., All-Hands, Jules). The expected output is a completed feature or a pull request, orchestrated by a central server.

Prompt Template:

```
Based on my Mission Brief in issue
{ISSUE-123}:
- Generate a detailed, step-by-step
plan.

- Use `@codebase` to identify all
affected files that should be part
of the plan.

- For each step, suggest if it is
better suited for [SYNC] or [ASYNC]
execution and define the expected
outcome.
```

Stage 3: Integration & Quality Assurance

- **Act as The Great Filter:** Remember, you are the final arbiter of quality. Apply your experience, taste, and domain knowledge to all AI-generated code.

- **Perform Micro-Reviews (for [SYNC] work):** Continuously validate each small piece of code as it's created during your interactive session.

- **Perform Macro-Reviews (for [ASYNC] work):** When an agent submits a pull request:
  a. Ensure it passes the full CI pipeline first (including tests, linters, and checks against the team's evaluation suite).
  b. Use an AI-assistive tool (e.g., PR-Agent) to get an initial automated analysis.
  c. Perform the final human review and merge.

- **Prompt for Risk-Based Tests:** Based on the "Key Risks" you identify in your Mission Brief, command the AI to generate targeted tests. This makes a deep testing strategy practical by automating the creation of complex validation code (e.g., "Write a test asserting a 403 for unauthorized access").

Prompt Template:

```
Generate targeted tests for `@file:
{path/to/your/code.py}`.
- The primary goal is to validate
against this risk I've identified:
'{DEVELOPER_IDENTIFIED_RISK}'.

- As a secondary goal, ensure all
**Success Criteria** from our
Mission Brief in issue {ISSUE-123}
are met.

- My IDE is configured with our
team's standards, so all tests must
adhere to the patterns and style
guides defined there.
```

Stage 4: Team Learning & Continuous Improvement

- **Link the Trace:** After completing a task, link to the relevant structured trace (the IDE chat history or the agent's PR) in the project management issue to preserve the "why" for future review and debugging.

- **Use the Right Tool for the Role:**
  - Smart Search: Sourcebot
  - Pair Programmer: Cursor/GitHub Copilot with Cline/RooCode
  - Autonomous Intern: All-Hands, Jules

- **Contribute Back:** If you develop a highly effective prompt, template, or workflow, submit it to the team's shared team-ai-directives repository. This is how the entire team gets smarter.

- **Update the Team's Memory:** After your change is merged, update any shared context files (like progress.md or activeContext.md) so the team's collective knowledge—and the AI's future context—is always current.

Prompt Template:

```
The work for issue @{ISSUE-123} is complete, and
this chat history represents a successful workflow.
Let's formalize this success for the team.
1. Analyze and Extract: Review our entire
interaction and extract the single most valuable
new rule or code exemplar that made this workflow
effective.
2. Prepare a Pull Request: Draft a complete Pull
Request to our team-ai-directives repository that
adds this new asset. The draft must include:
 A clear PR Title.
 A PR Description explaining what the new asset is
and why it's valuable to the team,
referencing @{ISSUE-123} for context.
 The complete File Content for the new rule or
exemplar, including a title, description, and code
examples.
3. Update the Issue Tracker: Independently, draft a
concise comment to post directly to @{ISSUE-123}.
This comment should be a Trace Summary of our work
on the original task, including:
 The problem that was solved.
 The key decisions made.
 The final accepted code solution.

Present the full Pull Request draft and the Issue
Tracker Comment in separate, clearly marked code
blocks for my final review before you proceed.
```

What This Means for R&D Leaders

The Twelve-Factor Agentic SDLC gives your team a clear framework to move from experimentation to scalable implementation. It doesn't replace developers — it empowers them to operate with more clarity, velocity, and control.

In a community survey we conducted with over 400 engineers in Israel, teams that adopted these practices reported 40–60% faster feature delivery — while maintaining or improving quality standards. More importantly, they gained a shared language and workflow model that scales across teams and tech stacks.

Ready to Take the Next Step?

Implementing Agentic SDLC at scale requires more than tools — it takes structure, alignment, and the right partners.

Talk to us about building your Agentic SDLC foundation. Schedule a consultation with Tikal's hands-on team.

Let's tech >

About TIKAL

Tikal is your hands-on tech consultancy partner for scaling your engineering organization and delivering impactful technology solutions. With over 25 years of experience, we help tech companies scale up their engineering capabilities with hands-on expertise in AI/ML, Backend, Data, DevOps, Fullstack, and Web, integrated into your engineering teams to bring value from day one.

As the founders of the Israeli Tech Radar, we drive industry impact by sharing insights and guiding engineering leaders and teams across the tech industry.