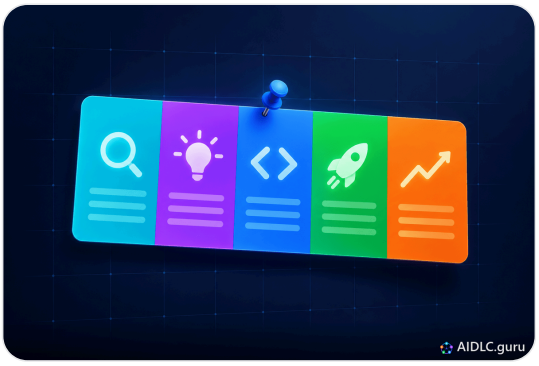
Quick Reference

AIDLC Cheatsheet

Quick reference guide for all five AIDLC phases. Keep this handy as you implement AI-Native Software Delivery.

One page for the wall beside your desk: the key activities and example prompts for each phase, the gate that ends it, and who drafts versus who decides. Below those are the five parts of a clear problem, a prompt skeleton, six stop signs, the ten operating rules, review by risk tier and the numbers worth watching. The PDF holds all of it with the diagrams.



A Analyze — Phase 01

AI-assisted requirements gathering and research synthesis

KEY ACTIVITIES	EXAMPLE PROMPTS
✓ Requirements extraction	"Summarize these requirements..."
✓ Research synthesis	"What are the risks of..."
✓ Competitive analysis	"Compare approaches for..."
✓ Feasibility assessment	"What questions should I ask..."

I Ideate — Phase 02

AI-driven architecture design and documentation generation

KEY ACTIVITIES	EXAMPLE PROMPTS
✓ Architecture diagrams	"Design an architecture for..."
✓ Design pattern selection	"Generate a Mermaid diagram..."
✓ API contract design	"What patterns fit this..."
✓ Technical specifications	"Write a technical spec for..."

D Develop — Phase 03

AI coding agents and intelligent code generation

KEY ACTIVITIES	EXAMPLE PROMPTS
✓ Delegate scoped tasks to agents	"Write failing tests for these criteria..."
✓ Tests from the criteria, before code	"Implement until these tests pass..."
✓ Review agent diffs	"Review this code for..."
✓ Refactoring and bug fixes	"Why is this failing..."

L Launch — Phase 04

Release assurance and deployment automation

KEY ACTIVITIES	EXAMPLE PROMPTS
✓ Full-suite and mutation re-checks	"Find what this suite misses..."
✓ CI/CD configuration	"Create a CI pipeline..."
✓ Deployment scripts	"Write deployment script..."
✓ Release validation	"What should I test before..."

Curate

Phase 05

AI-powered monitoring and continuous improvement

KEY ACTIVITIES

- Log analysis
- Root cause analysis
- Performance optimization
- Documentation updates

EXAMPLE PROMPTS

"Analyze these logs..."
"What caused this error..."
"How can I optimize..."
"Update docs to reflect..."

At a Glance

Phase	Focus	AI Role	Output
Analyze	Requirements	Research assistant	Specs, analysis docs
Ideate	Design	Architect partner	Diagrams, specs
Develop	Code	Coding agent	Working, tested code
Launch	Deploy	QA engineer	Verified release, pipelines
Curate	Maintain	SRE assistant	Fixes, updates



Gates at a Glance

Each phase ends at a gate. Work does not move forward until the gate passes, and a failed gate sends it back, not onward.

Phase	Gate	Passes when	If it fails
Analyze	Brief signed	The product owner signs a requirements brief with testable acceptance criteria	Stay in Analyze and close the open questions
Ideate	Design approved	A person approves the design, the contract and the spec	Revise the design, or return to Analyze if the brief was wrong
Develop	Tests pass	Tests written from the criteria pass, and a person has reviewed and merged the change	Fix the code, never the test that proves the criterion
Launch	Release verified	The suite passes on the integrated release and the rollback has been rehearsed	The failing check goes back to Develop
Curate	Lesson captured	What production taught is written into a runbook, a test or a rule for the agents	The same incident will happen again



Who Does What

An agent can draft nearly everything. It is never accountable for anything.

Phase	The agent drafts	A person decides	Who
Analyze	Requirements, conflicts, gaps and risks from the source material	What the problem is and what done means	Product owner
Ideate	Design options with their trade-offs, the contract and the spec	Which design to build	Tech lead
Develop	The plan, the tests and the code	Whether the change is merged	The reviewing engineer
Launch	Release notes, the release plan and the release-only checks	Whether the release ships	Release owner
Curate	Incident summaries, runbook updates and proposed rules	What changes, and what goes back to Analyze	Service owner



Before You Prompt

Five parts of a clear problem. If one is missing, the agent will guess it.

Who

The specific person or role with the problem.

Needs

The outcome they need, not the feature.

Why

The reason it matters. It settles later trade-offs.

Must never

The failure that makes any solution unacceptable.

Done when

Acceptance criteria in Given/When/Then.



Problem Definition →

A Prompt Skeleton That Works

Five parts, in this order, for any task you hand to an agent. Fill in every bracket before you send it.

Part	What goes in it
Context	The files, the spec and the rules to read first. Point at real paths.
Task	One outcome, small enough to review in a single sitting.
Constraints	What must not change: existing tests, public interfaces, files out of scope.
Done when	The acceptance criteria, by ID, and the command that proves them.
Show your work	Ask for the plan and the assumptions first, and wait for approval.

```
CONTEXT
  Read [spec file] and [the module it changes].
  Follow the repo instruction file.

TASK
  Implement [one criterion or one slice].

CONSTRAINTS
  Do not edit existing tests.
  Do not touch [paths out of scope].

DONE WHEN
  [AC-1], [AC-2] pass under [test command].

SHOW YOUR WORK
  First reply with your plan, your assumptions
  and your open questions. Wait for my approval
  before you write any code.
```



Prompt Library →

Stop Signs

Six moments to stop the agent and bring in a person. Carrying on past any of them is how a small task becomes an incident.

A test was edited

The agent changed or deleted a test to get to green. Fix the code, never the test that proves the criterion.

The scope grew

The diff touches files the plan never named. Stop and approve a new plan, or split the work.

Third try, same fix

The agent is looping on one failure. The problem is in the context or the plan, not in the next attempt.

Secrets or personal data

A credential or a customer record is about to reach a prompt, a log or a commit. Check the data tier first.

No criterion covers it

The agent is deciding behavior the brief never described. That decision belongs to the product owner.

Nobody can explain the diff

If the reviewer cannot say what the change does and why, it is not ready to merge, whatever the checks say.



Failure Patterns →

Ten Rules for AI in the SDLC

The operating model in ten sentences.

1. **AI in the SDLC is an operating model change, not a tooling purchase.**
2. **Authoring is cheap. Verification is the constraint.**
3. **Individual output is not organizational throughput.**
4. **Governance before scale.**
5. **Baseline before adoption.**
6. **Measure outcomes, not activity.**
7. **Humans own merges.**
8. **The bar for AI-authored code is the bar for human code, applied with more suspicion.**
9. **Agents are services.**
10. **The compounding asset is your context, not the model.**

Full playbook →

Do's

- ✓ Provide context in every prompt
- ✓ Review all AI-generated code
- ✓ Iterate on prompts for better results
- ✓ Document AI-assisted decisions
- ✓ Start with one phase, expand gradually

Don'ts

- ✗ Blindly accept AI output
- ✗ Skip security reviews
- ✗ Expect AI to understand without context
- ✗ Replace human judgment entirely
- ✗ Try to implement all phases at once

What Good Looks Like

Signs your team is getting AIDLC right - qualities to aim for, not benchmark promises.

Less time on boilerplate

More of your day goes to design and review, less to repetitive typing.

Tests written by default

Coverage stops being the thing that gets cut when deadlines loom.

Docs that stay current

Documentation tracks the code instead of rotting behind it.

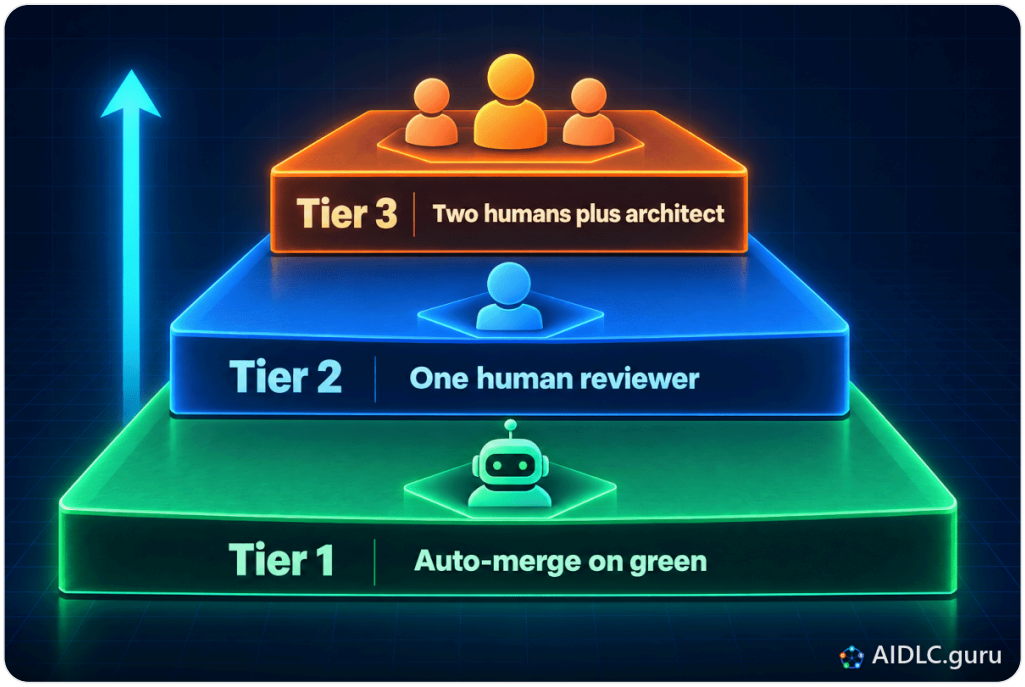
Humans own the review

Every agent diff is read and understood before it ships.

Review by Risk Tier

Review effort follows the risk of the change, not the number of pull requests.

Tier	What it covers	Review	Agent-authored?
1. Low risk	Docs, copy, tests, flag-off scaffolding and safe-class dependency bumps	Can merge on green checks, with a person sampling the merges on a regular cadence	Yes
2. Standard	Ordinary feature work behind a flag	One human reviewer, after AI pre-review has cleared the mechanical comments	Yes, and labeled as such
3. High risk	Authentication, money and eligibility logic, personal data paths, migrations and infrastructure	Two humans plus an architect. No AI auto-approval	No



AI Code Review at Scale →

Numbers to Watch

Measure outcomes, not activity, and report every number against the baseline you captured before adoption.

✓ Report these

Cycle time, ticket to production

PR review turnaround

Change failure rate

Bug escape rate

Deployment frequency

✕ Not proof of success

Adoption: how many people have a license or used it this week

AI-authored lines of code

Suggestions accepted

Tasks completed by agents



Measuring Impact →

Adoption Order

Crawl, walk, run. Take one step, build trust, then take the next.

1. **Engineers.** Coding agents in daily work, inside the review process you already have.
2. **Product.** AI to research, define the details and explain the problem in every story.
3. **QA.** Tests generated from the acceptance criteria. Manual regression ends.
4. **Deployment.** Button-press deploy, then auto-build with one click, then deploy on push.
5. **Production support.** Extra credit: an agent as the first responder.



Crawl, Walk, Run →