THE THREE FILTERS

ARCHITECTURE DECISION WORKSHEET simplicity-first.dev

Apply these three filters to every architecture decision, pull request, and design review. If a proposal fails any filter, simplify before proceeding. Date: _____ System/Decision: _____

FILTER 1 THE 2 AM TEST

Can a tired engineer who did not build this debug it at 2 AM with only the logs?

EVALUATE FOUR DIMENSIONS

- **Discoverability:** Can you find where the problem is? Are error messages specific? Do logs point to the source?
- **Traceability:** Can you follow a request from entry to exit? Can you trace cause to effect without the original author?
- **Change Safety:** Can you fix the problem without breaking something else? Are changes isolated?
- **Cognitive Load:** How many files, services, and concepts must you hold in your head to understand the failure?

DIAGNOSTIC CHECKLIST

- [ ] A new engineer can trace the primary path in under 1 hour
- [ ] Error messages identify the failing component and the input that caused it
- [ ] A failure can be debugged without contacting the person who built the system
- [ ] The request path touches fewer than 10 files from entry to response
- [ ] On-call mean time to recovery is under 15 minutes for primary path failures

FILTER 2 THE HALF-RULE

Build half of what you think you need. Ship it. Measure. Add only what the data justifies.

APPLY TO EVERY PROPOSAL

- **Services:** Can you deliver the same value with half the services? What happens if you merge two?
- **Abstractions:** Does every interface have 2+ implementations? If not, delete the interface, use the concrete type.
- **Features:** Which half of the planned features serve a measured need vs. a hypothetical one?
- **Infrastructure:** Can you ship on a single server and database first? What forces the second?

DIAGNOSTIC CHECKLIST

- [ ] Every abstraction has at least 2 concrete implementations today (not "someday")
- [ ] No message queue exists without a measured throughput requirement demanding it
- [ ] No caching layer exists without measured latency data proving the need
- [ ] The system could ship as a single deployable unit and still serve current users
- [ ] Every component was added in response to a real problem, not a predicted one

FILTER 3 PRIMARY PATH FIRST

Build for the 95% use case. Put edge cases on a budget. Contain them behind seams.

IDENTIFY AND PROTECT THE PRIMARY PATH

- **Traffic analysis:** What do 95% of requests actually do? That is the primary path. Design it first, test it first, monitor it first.
- **Edge case budget:** What % of engineering time goes to the 5% case? If more than 20%, the edge cases are consuming the primary path budget.
- **Seam placement:** Are edge cases contained behind clear boundaries? Can you modify the primary path without touching edge case logic?
- **Consequence test:** If an edge case fails, is the consequence severe (legal, safety, financial) or is it inconvenience? Only severe consequences justify primary-path treatment.

DIAGNOSTIC CHECKLIST

- [ ] The primary path can be read top-to-bottom without branching into edge case handlers
- [ ] Edge cases are behind explicit seams (strategy pattern, feature flags, separate modules)
- [ ] No more than 20% of sprint capacity goes to requirements serving less than 5% of users
- [ ] Every edge case promoted to the primary path has documented evidence of real usage or severe consequence
- [ ] A new developer can understand the primary path without understanding any edge cases

2 AM TEST SCORE

○ ○ ○ ○ ○

Fill one circle per checkbox passed (0-5)

HALF-RULE SCORE

○ ○ ○ ○ ○

Fill one circle per checkbox passed (0-5)

PRIMARY PATH SCORE

○ ○ ○ ○ ○

Fill one circle per checkbox passed (0-5)

INTERPRET YOUR SCORE

12-15	**Ship it.** Architecture passes all three filters.
8-11	**Simplify first.** Address the lowest-scoring filter before proceeding.
0-7	**Redesign.** Start with the simplest architecture that serves the primary path.

Apply before every architecture decision, design review, and pull request.
Print this. Stick it on the wall next to the whiteboard. Use it.

SIMPLICITY-FIRST.DEV

THE ECONOMICS OF SOFTWARE ARCHITECTURE