♛ Soviet Chess Framework

Multi-Agent System for Software Development and Operation

Version 2.0 — April 2026 | David Pineda | Sumud Project

1. The Problem

A single AI agent writing code hits three walls:

Context Loss
When a large task is split into subtasks, each subtask loses context about the whole. Result: subagents guess imports, miss dependencies, generate incompatible code.

Observed: 138 compilation errors from incorrect import guessing (Sumud, 2026-04-08)

Error Accumulation
Writing 5 files then compiling means errors multiply. 1 wrong import in file 1 cascades to 138 errors across 5 files. Fixing takes 30 minutes instead of 1.

Root cause: no intermediate verification between file generations

Role Confusion
A single agent tries to architect, implement, test, review, and deploy simultaneously. It optimizes for the last instruction, not the holistic quality.

Ref: Qian et al. (2023) — role separation reduces hallucination by 66%

2. The Solution: Three Layers

```
LAYER 3: PRODUCTION — Agents that OPERATE the product
  ♚ Rey:     Content strategy, campaigns, crisis response
  ♛ Reina:   Ethical gate, content quality review
  ♜ Torre:   Publishing to social media channels
  ♝ Alfil:   Community engagement, moderation
  ♞ Caballo: Sentiment analysis, anomaly detection
  ♟ Peon:    Data sync, scheduled maintenance

LAYER 2: DEVELOPMENT — Agents that BUILD the product
  ♚ Rey:     Architecture decisions, task decomposition
  ♛ Reina:   Code review, security gate
  ♜ Torre:   Backend implementation
  ♝ Alfil:   Frontend implementation
  ♞ Caballo: Testing (unit, integration, E2E)
  ♟ Peon:    Build, deploy, CI/CD

LAYER 1: INFRASTRUCTURE — Shared tools and communication
  MCP Server | Blackboard | Git | Database | Compiler | Test Runner
```

Key: Each layer uses the same 6 chess pieces but for different domains. Development agents build the code. Production agents run the product. Infrastructure is shared.

3. The Agents

PIECE	ROLE	CAN DO	CANNOT DO	COGNITIVE MODEL
♚ **Rey**	Architect	Read code, write decisions, decompose tasks	Write code, deploy, commit	System 2: slow, deliberate (Kahneman)
�� **Reina**	Reviewer	Read code, review diffs, run tests (read-only)	Write code, deploy	Dual-process: fast heuristics + slow analysis
♜ **Torre**	Backend	Read/write backend code, run compiler	Deploy, merge to main, modify CI	Pattern-matching for CRUD, deep for novel logic
♝ **Alfil**	Frontend	Read/write UI code, run type checker	Deploy, modify backend	Visual + structural (components, layouts)
♞ **Caballo**	Tester	Read code, write tests, run test suite	Write production code	Adversarial: finds bugs, not confirms success
♟ **Peon**	DevOps	Run deploy scripts, manage infra	Write application code	Procedural: follows protocols precisely

> **Principle of Minimal Authority** (Saltzer & Schroeder, 1975): Each agent has the minimum capabilities for its role. Torre cannot deploy. Caballo cannot write production code. This prevents entire classes of errors.

4. Communication: The Blackboard

Agents don't talk to each other directly. They read and write to a shared blackboard. This prevents context loss — the #1 problem in multi-agent systems.

```
┌───────────────────────────────────────────────┐
│             BLACKBOARD                        │
│                                               │
│  context/                                     │
│    ├── current_task.yaml   ← Rey writes       │
│    ├── stack_profile.yaml  ← Rey writes       │
│    └── import_map.json     ← Torre writes      │
│                                               │
│  signals/                                     │
│    ├── compile_result.json ← Compiler writes   │
│    ├── test_result.json    ← Test runner writes│
│    └── lint_result.json    ← Linter writes     │
│                                               │
│  artifacts/                                   │
│    └── files_in_progress/  ← Torre/Alfil write │
│                                               │
│  reviews/                                     │
│    └── pending/            ← Reina writes      │
│                                               │
│  ALL AGENTS CAN READ EVERYTHING               │
└───────────────────────────────────────────────┘
```

Import Map Protocol (prevents the 138-error problem)

Before splitting any large file, Torre writes an **import map** to the blackboard. Every submodule reads this map to get correct imports. No guessing.

```
BEFORE refactoring traveler_bot.rs (3675 lines):

1. Torre reads entire file
```

```
2. Torre extracts: imports, public functions, cross-references
3. Torre writes import_map.json to blackboard
4. For each submodule:
   a. Read import_map from blackboard
   b. Write ONE file
   c. cargo check  ← CRITICAL: verify IMMEDIATELY
   d. Fix any errors
   e. Proceed to next file
```

5. Task Lifecycle

```
┌──────────┐     ┌──────────┐     ┌──────────┐
│ CREATED  │────►│ ACTIVE   ���───►│ TESTING   │
│  (Rey)   │     │ (Torre/  │     │ (Caballo)│
└──────────┘     │  Alfil)  │     └──────────┘
                 └──────────┘           │
                                        ▼
    Each file:                   ┌──────────┐
    write → compile → fix        │ REVIEW   │
    write → compile → fix        │ (Reina)  │
    write → compile → fix        └─────┬────┘
                                 Pass? │
                            ┌────Yes───┴───No────┐
                            ▼                    ▼
                       ┌────────┐          ┌─────���────┐
                       │DEPLOY  │          │ RETURNED │
                       │ (Peon) │          │ (fix it) │
                       └────────┘          └──��───────┘
```

Contracts: Pre and Post Conditions

Every task has machine-verifiable conditions. If postconditions fail, the task is not done.

Preconditions (before starting)

- Code compiles
- All tests pass
- File exists / function exists

Postconditions (must be true when done)

- Code compiles **(mandatory)**
- All tests pass **(mandatory)**
- New code has tests
- No file exceeds 1500 LOC

6. The Core Loop: Generate-Check-Fix

> **THE #1 RULE:** Compile after EVERY file. Never write multiple files before checking. The cost of checking early is 5 seconds. The cost of 138 accumulated errors is 30+ minutes.

```
Torre's work loop:

┌─────────────────────────┐
│ 1. Read context +       │
│    blackboard            │
└───────────┬─────────────┘
            ▼
┌─────────────────────────┐
│ 2. Write/edit ONE file  │◄──────────┐
└───────────┬─────────────┘           │
            ▼                         │
┌─────────────────────────┐           │
│ 3. cargo check           │     ┌─────┴─────┐
│    (2-5 seconds)        │    │ 4. Fix    │
└────────���──┬─────────────┘     │    errors  │
            ▼                   └─────┬─────┘
       ┌────┴────┐                    │
       │ Errors? ├── Yes ─────────────┘
       └────┬────┘
            │ No
            ▼
┌─────────────────────────┐
```

```
│ 5. Signal: file ready   │
│    Proceed to next file │
└─────────────────────────┘
```

7. Quality Framework

Architectural Invariants (never violated)

ID	RULE	WHY
INV-1	No file > 1500 LOC	Large files = context loss for agents and humans
INV-2	No function > 80 LOC	Long functions = untestable, unreadable
INV-3	SQL parameterized only	Prevents SQL injection (OWASP #1)
INV-4	Auth on all admin endpoints	Prevents unauthorized access
INV-5	Every endpoint has a test	No test = no confidence = no deploy
INV-6	No secrets in code	Env vars only — secrets rotate
INV-7	No auto-publish without approval	Humans approve public-facing content

Health Score

```
Score = weighted average of 6 metrics (0-100):

Files under LOC limit     (25%)  ████████████████████████░  92
Endpoint test coverage     (25%)  ███████████████████░░░░░░  76
Compiler warnings         (15%)  ██████████████░░░░░░░░░░░  56
Dead code items           (10%)  ██████████████████░░░░░░░  72
Duplicated patterns       (15%)  ██████████░░░░░░░░░░░░░░░  40
Crate balance             (10%)  ███████��░░░░░░░░░░░░░░░░░  32

OVERALL HEALTH: 66/100 — needs improvement
TARGET: 80+
```

Test Pyramid

```
                  /    E2E     \           5-10% of tests
                 /   (slow,    \          Playwright / Cypress
                /   high value) \         Critical user journeys
               /─────────────────\
              /   Integration     \        60-70% of tests
             /  (medium speed)     \       Every API endpoint
            /───────────────────────\
           /       Unit Tests        \     20-30% of tests
          /     (fast, isolated)      \    Pure functions
         /─────────────────────────────\
```

8. Cross-Layer Interactions

Production finds a bug → Development fixes it

```
LAYER 3 (Production):
  ♞ Caballo detects: "Instagram sync failing for 2 hours"
       │
       ▼ SIGNAL: incident/sync-failure.yaml
───────────────────────────────────────────────
LAYER 2 (Development):
  ♚ Rey reads incident → creates task with contract
       │
       ▼
  ♜ Torre implements fix → cargo check → cargo test
       │
       ▼
  ♞ Caballo writes regression test
       │
       ▼
  ♛ Reina reviews → approved
       │
       ▼
  ♟ Peon deploys
       │
       ▼ SIGNAL: fix deployed
──────────��────────────────────────────────────
LAYER 3 (Production):
  ♞ Caballo verifies: sync restored ✓
```

Development enhances Production agents

```
Dev-Rey decides: "Soviet Torre needs Instagram Reels support"
     │
     ▼
Dev-Torre implements: publish_reel() in broadcaster.rs
Dev-Caballo writes: integration test for Reels endpoint
Dev-Reina reviews: security ✓, tests ✓, quality ✓
♟ Dev-Peon deploys new binary
     │
     ▼
Prod-Torre now has Reels capability
Prod-Rey updates content strategy to include Reels
```

9. How to Start

For an existing project:

> **Step 1:** Copy the Sumudfiles/ directory to your project root.
> **Step 2:** Open Claude Code and say:
> "*Read Sumudfiles/framework/adoption.md. We're using Soviet Chess. As Rey, audit this codebase and give me a health score.*"
> **Step 3:** Follow the priorities Rey identifies.
> **Step 4:** At the end of each session, generate a postmortem.

For a new project:

> **Step 1:** Copy Sumudfiles/ to your project root.
> **Step 2:** Fill in Sumudfiles/templates/stack-profile.yaml with your stack.

Step 3: Open Claude Code and say:
"Using Soviet Chess Framework, as Rey design the initial architecture for: [describe your project]. Create skeleton with proper module boundaries."

Incremental adoption (for skeptics):

WEEK	ADOPT	AGENT	EFFORT
1	Compile after every file edit	Torre	Zero — just discipline
2	Write 1 test per new endpoint	Caballo	5 min per endpoint
3	Run security checklist before commit	Reina	2 min per commit
4	Split any file > 1500 LOC	Rey	1-2 hours per file
5+	Full framework with contracts	All	Built into workflow

10. References

1. Qian, C. et al. (2023). *Communicative Agents for Software Development.* arXiv:2307.07924
2. Hong, S. et al. (2023). *MetaGPT: Meta Programming for Multi-Agent Collaborative Framework.* arXiv:2308.00352
3. Park, J.S. et al. (2023). *Generative Agents: Interactive Simulacra of Human Behavior.* arXiv:2304.03442
4. Hayes-Roth, B. (1985). *A Blackboard Architecture for Control.* Artificial Intelligence, 26(3).
5. Minsky, M. (1986). *The Society of Mind.* Simon & Schuster.
6. Brooks, R.A. (1986). *A Robust Layered Control System for a Mobile Robot.* IEEE J. Robotics.
7. Meyer, B. (1992). *Applying Design by Contract.* IEEE Computer, 25(10).
8. Smith, R.G. (1980). *The Contract Net Protocol.* IEEE Trans. Computers.
9. Beck, K. (2003). *Test-Driven Development.* Addison-Wesley.
10. Fagan, M.E. (1976). *Design and Code Inspections.* IBM Systems Journal, 15(3).
11. Kahneman, D. (2011). *Thinking, Fast and Slow.* FSG.
12. Beyer, B. et al. (2016). *Site Reliability Engineering.* O'Reilly.
13. Forsgren, N. et al. (2018). *Accelerate: The Science of DevOps.* IT Revolution.
14. Cohn, M. (2009). *Succeeding with Agile.* Addison-Wesley.
15. Kazman, R. et al. (2000). *ATAM: Architecture Evaluation.* CMU/SEI.
16. Saltzer, J.H. & Schroeder, M.D. (1975). *Protection of Information.* Proc. IEEE.
17. Claessen, K. & Hughes, J. (2000). *QuickCheck.* ICFP.
18. Letouzey, J.P. (2012). *The SQALE Method.* IEEE MTD Workshop.
19. Martin, R.C. (2003). *Agile Software Development.* Pearson.
20. Shore, J. & Warden, S. (2007). *The Art of Agile Development.* O'Reilly.

Soviet Chess Framework v2.0 — A Multi-Agent System for Software Development
Created by David Pineda with Claude Opus 4.6 — April 2026