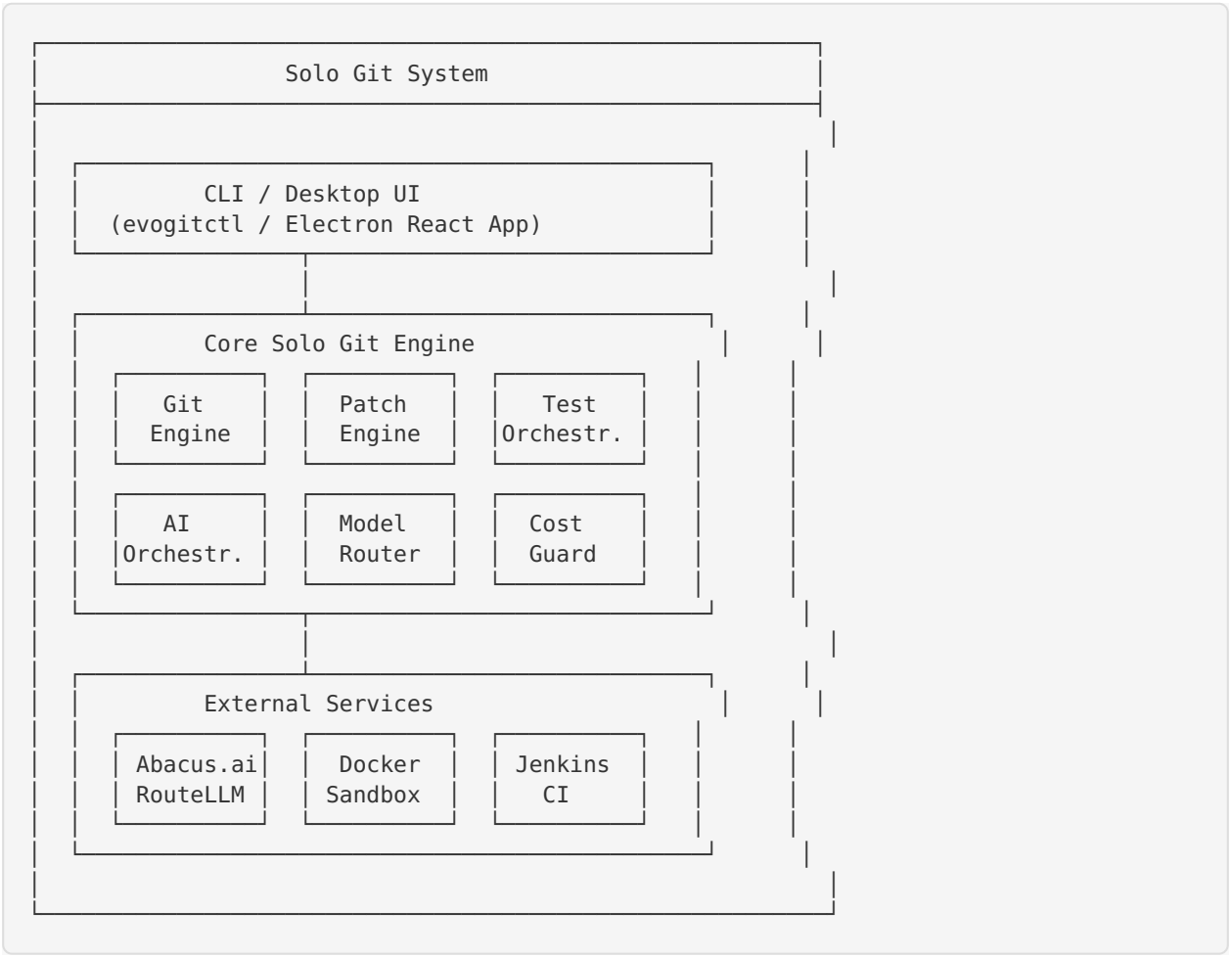
Solo Git Architecture

System Architecture Overview

High-Level Architecture



Core Components

1. Git Engine

Purpose: Manage Git operations, repositories, and workpads

Responsibilities:
- Repository initialization (zip/git)
- Workpad lifecycle (create, checkpoint, promote)
- Branch management (trunk + ephemeral pads)
- Diff generation and retrieval
- Fast-forward merge operations
- Rollback capabilities

Key Classes:
- `GitEngine` - Main Git operations orchestrator

- `Repository` - Repository abstraction
- `Workpad` - Ephemeral workspace abstraction

See: Git Engine Design (./git-engine.md)

---

2. Patch Engine

Purpose: Apply code changes and manage conflicts

Responsibilities:
- Patch application to workpads
- Conflict detection and resolution
- Patch validation
- Checkpoint creation
- Rollback on failures

Key Methods:
- `apply_patch(pad_id, diff)` - Apply unified diff
- `validate_patch(pad_id, diff)` - Check for conflicts
- `create_checkpoint(pad_id)` - Save state

---

3. Test Orchestrator

Purpose: Execute tests in isolated sandboxes

Responsibilities:
- Docker container management
- Test configuration parsing (evogit.yaml)
- Parallel test execution
- Dependency graph resolution
- Result collection and reporting
- Timeout enforcement

Key Classes:
- `TestOrchestrator` - Main test coordinator
- `SandboxManager` - Docker container lifecycle
- `TestRunner` - Individual test execution

See: Test Orchestrator Design (./test-orchestrator.md)

---

4. AI Orchestrator (Phase 2)

Purpose: Coordinate AI model interactions

Responsibilities:
- Task routing to appropriate models
- Context preparation (repo map, recent changes)
- Prompt engineering
- Response parsing

- Error handling and retries
- Cost tracking

Key Classes:
- `AIOrchestrator` - Main AI coordinator
- `ModelRouter` - Model selection logic
- `ContextBuilder` - Prepare context for models

See: Model Routing Design (./model-routing.md)

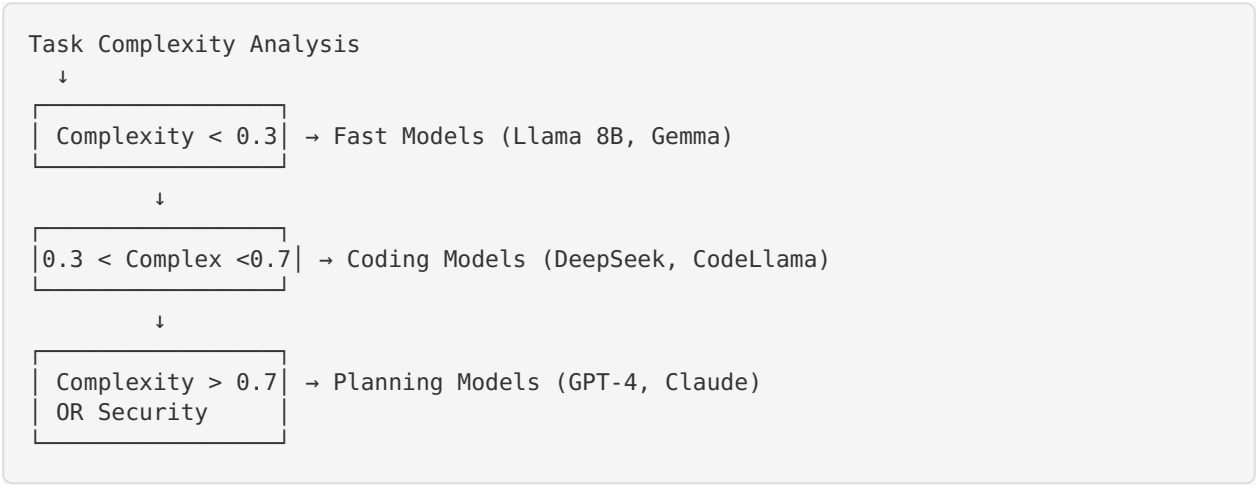
---

5. Model Router (Phase 2)

Purpose: Select optimal model for each task

Responsibilities:
- Complexity analysis
- Model tier selection (planning/coding/fast)
- Escalation policies
- Budget awareness
- Performance tracking

Selection Logic:



---

6. Cost Guard (Phase 2)

Purpose: Enforce budget limits

Responsibilities:
- Track API costs per model
- Enforce daily caps
- Send alerts at thresholds
- Prevent overspending
- Generate cost reports

Enforcement:
- Pre-flight budget check
- Token estimation

- Real-time cost tracking
- Automatic request blocking at cap

---

7. Jenkins Integration (Phase 3)

Purpose: Post-merge smoke tests and rollback

Responsibilities:
- Trigger Jenkins pipelines
- Monitor build status
- Auto-rollback on failures
- Status notifications

Pipeline Flow:

```
Promote to Trunk
  ↓
Jenkins Pipeline Triggered
  ↓
Smoke Tests Run
  ↓
┌──────────┬──────────┐
│  GREEN   │   RED    │
│          │ Rollback │
│ Deploy?  │ Reopen   │
│          │ Workpad  │
└──────────┴──────────┘
```

Data Flow

Repository Initialization

```
User: evogitctl repo init --zip app.zip
  ↓
CLI → GitEngine.init_from_zip()
  ↓
1. Extract zip to /data/repos/<repo_id>
2. Git init
3. Git add .
4. Git commit -m "Initial"
5. Store metadata
  ↓
Return: repo_id
```

Workpad Creation

```
User: evogitctl pad create "add-feature"
  ↓
CLI → GitEngine.create_workpad()
  ↓
1. Get repo trunk
2. Create branch: pads/add-feature-<timestamp>
3. Checkout new branch
4. Store workpad metadata
  ↓
Return: pad_id
```

Patch Application

```
User/AI: Apply patch to pad_x123
  ↓
CLI → PatchEngine.apply_patch()
  ↓
1. Checkout workpad branch
2. Validate patch (no conflicts)
3. Apply patch (git apply)
4. Stage changes (git add)
5. Commit (git commit)
6. Create checkpoint tag
  ↓
Return: checkpoint_id
```

Test Execution

```
User: evogitctl test run --pad pad_x123
  ↓
CLI → TestOrchestrator.run_tests()
  ↓
1. Load test config (evogit.yaml)
2. Build dependency graph
3. For each test:
   a. Create Docker container
   b. Mount workpad (read-only)
   c. Run test command
   d. Collect output
   e. Clean up container
4. Aggregate results
  ↓
Return: TestResults (green/red)
```

Workpad Promotion

```
User: evogitctl pad promote pad_x123
  ↓
CLI → GitEngine.promote_workpad()
  ↓
1. Check tests are green (if required)
2. Checkout trunk (main)
3. Fast-forward merge (--ff-only)
4. Delete workpad branch
5. Update metadata
  ↓
Return: commit_hash
  ↓
(Optional) Trigger Jenkins pipeline
```

File Structure

```
/data/
├── repos/                    # Repository storage
│   ├── repo_a1b2c3d4/       # Individual repository
│   │   ├── .git/            # Git internals
│   │   ├── src/             # Source code
│   │   └── evogit.yaml      # Test configuration
│   └── repo_e5f6g7h8/
│
├── metadata/                 # System metadata
│   ├── repos.json           # Repository metadata
│   └── workpads.json        # Workpad metadata
│
├── logs/                     # Audit logs
│   ├── api_calls.log        # AI API interactions
│   ├── git_operations.log   # Git commands
│   └── test_runs.log        # Test execution logs
│
└── cache/                    # Caches
    ├── embeddings/          # Code embeddings (RAG)
    └── models/              # Model metadata
```

Security Considerations

Sandboxing

- **Tests run in Docker**: Isolated from host system
- **Read-only mounts**: Workpads mounted read-only
- **Network isolation**: Containers have no network access
- **Resource limits**: CPU, memory, and time limits enforced

API Security

- **API keys**: Stored securely in config file
- **HTTPS only**: All API calls over TLS

- **Rate limiting**: Respect API rate limits
- **Cost caps**: Hard budget limits enforced

Git Security

- **No force push**: Never force push to trunk
- **Fast-forward only**: Promotes must be fast-forward
- **Protected trunk**: Trunk branch is protected
- **Audit trail**: All operations logged

Performance Characteristics

Repository Operations

- **Init from zip**: O(n) where n = file count
- **Init from Git**: O(repo size)
- **Workpad create**: O(1) - just branch creation
- **Patch apply**: O(m) where m = patch size
- **Promote**: O(1) - fast-forward merge

Test Execution

- **Serial**: O(sum of test times)
- **Parallel**: O(longest test time)
- **Container overhead**: ~2-3 seconds per test

AI Operations (Phase 2)

- **Planning**: 2-5 seconds (GPT-4/Claude)
- **Coding**: 5-15 seconds (DeepSeek)
- **Fast ops**: 1-3 seconds (Llama 8B)

Scalability

Current Design (Single Machine)

- **Repositories**: Hundreds
- **Concurrent workpads**: Dozens
- **Parallel tests**: 4-8 (configurable)
- **API requests**: Rate limited by provider

Future Scaling (Phase 4+)

- **Distributed test execution**: Multiple Docker hosts
- **Shared repository storage**: Network file system
- **Load balancing**: Multiple API endpoints
- **Horizontal scaling**: Multiple Solo Git instances

Related Documents

- Git Engine Design (./git-engine.md)
- Test Orchestrator Design (./test-orchestrator.md)
- Model Routing Design (./model-routing.md)
- Phase 1 Overview (../phases/phase-1-overview.md)

Last Updated: October 16, 2025