Waft Open Issues & Concerns

Date: 2026-01-10

Status: Active Development

(v0.3.1-alpha)

Table of Contents

1. Critical Issues

2. High Priority Issues

3. Medium Priority Issues

4. Low Priority Issues

5. Known Limitations

6. Technical Debt

7. Future Considerations

Critical Issues

None (Post Phase 1 Refactoring) [x]

All critical error handling issues have

been resolved in Phase 1.

High Priority Issues

1. God Objects in Core Modules

Severity: High

Impact: Maintainability, Testability

Files:

- src/waft/main.py (2020 lines, 54

commands)

- src/waft/core/visualizer.py (2344

lines, 10+ responsibilities)

- src/waft/core/agent/base.py (924

lines, multiple concerns)

Problem:

These files violate Single

Responsibility Principle:

main.py:

- Mixes CLI parsing, business logic,

display, gamification, epistemic

tracking

- 54 separate command functions in

one file

- Difficult to test individual commands

- Changes to one command risk

affecting others

visualizer.py:

- Git status collection

- System info collection

- Work effort tracking

- HTML generation (500+ lines of

embedded strings)

- Analytics aggregation

- Multiple rendering methods

agent/base.py:

- OODA cycle logic

- Inventory management

- Reproduction mechanics

- Genome tracking

- Decision engine integration

- Empirica integration

Recommended Fix: See

REFACTORING_PLAN.md Phase 2

Estimated Effort: 2-3 weeks

2. Foundation Module Duplication

Severity: High

Impact: Maintenance, Confusion

Files:

- src/waft/foundation.py (1088 lines) -

PRODUCTION

- src/waft/foundation_v2.py (1059

lines) - EXPERIMENTAL

Problem:

Nearly identical files with minor

enhancements in v2:

- Both actively imported

- Bug fixes to one don't propagate to

the other

- Unclear which should be used

- Will diverge over time

Current Usage:

```
# foundation.py used by:
- src/waft/verify_foundation.py
- src/waft/generate_artifacts.py

# foundation_v2.py used by:
- scripts/generate_foundation_demo.py (demo only)
```

Recommended Fix:

1. Decide canonical version (likely v2
with enhancements)
2. Migrate all code to canonical
version
3. Deprecate and remove duplicate
4. Document migration path

See:

docs/FOUNDATION_STATUS.md for

detailed analysis

Estimated Effort: 1-2 days

3. Circular Dependency Risk

Severity: High

Impact: Architecture, Modularity

Problem:

42+ files use parent directory imports

(from ..):

- main.py imports from core.

- core. imports from api.

- api. imports from core.*

Potential Circular Chain:

```
cli/ -> core/ -> api/ -> core/ (CIRCULAR)
```

Recommended Fix:

1. Map all imports to detect cycles

2. Use dependency injection to break

cycles

3. Create clear layering:

```
cli/ -> core/ -> models/

api/ -> core/ -> models/

(Never: core -> cli or core -> api)
```

Estimated Effort: 2-3 days

4. Incomplete Karma System

Severity: High

Impact: Feature Completeness

File: src/waft/karma.py (239 lines)

Problem:

5 unimplemented methods with

TODO comments:

```
def calculate_karma(self, life_log: Dict) -> float:
    # TODO: Implement Karma calculation
    pass

def access_akasha(self, soul_id: str) -> Dict:
    # TODO: Implement Akasha access
    pass

# Similar TODOs at lines 151, 181, 204, 219
```

Impact:

Karma/reincarnation system is

essentially a stub - non-functional

feature exposed in API.

Options:

1. Complete implementation (3-4

days)

2. Remove and document as "future

feature"

3. Keep as experimental/stub with

clear warnings

Recommended: Option 2 (remove or

hide until ready)

Estimated Effort: 0.5 days (removal)

OR 3-4 days (completion)

5. Missing Abstraction Layers

Severity: High

Impact: Testability, Flexibility

Problem:

8+ manager classes without shared

interface:

`MemoryManager`

`SubstrateManager`

`EmpiricaManager`

`GamificationManager`

`GitHubManager`

`TavernKeeper`

`Narrator`

`DecisionMatrixCalculator`

Issues:

- Can't mock in tests

- API contract unclear

- Can't swap implementations

- Hard to understand hierarchy

Recommended Fix:

Create Manager ABC:

```
from abc import ABC, abstractmethod



class Manager(ABC):

    @abstractmethod

    def initialize(self) -> bool:

        pass



    @abstractmethod

    def validate(self) -> tuple[bool, Optional[str]]:

        pass
```

Estimated Effort: 2 days

Medium Priority Issues

6. Embedded HTML in Visualizer

Severity: Medium

Impact: Maintainability, Separation of

Concerns

File: src/waft/core/visualizer.py

Problem:

500+ lines of HTML/CSS/JS

embedded as Python strings:

```
html = f"""
<!DOCTYPE html>
<html>
<head>
    <style>
        /* 200 lines of CSS */
    </style>
</head>
<body>
    <!-- 300 lines of HTML -->
    <script>
```

Issues:

- No syntax highlighting

- Hard to edit

- No template reuse

- Violates separation of concerns

Recommended Fix:

Extract to Jinja2 templates:

```
templates/
  dashboard.html.j2
  components/
    metrics.html.j2
    status.html.j2
```

Estimated Effort: 1-2 days

7. Template Organization

Severity: Medium

Impact: Code Organization

File: src/waft/templates/init.py (434

lines)

Problem:

Templates embedded in init.py:

- Justfile template (60 lines)

- GitHub CI template (50 lines)

- agents.py template (70 lines)

Issues:

- Can't edit templates without editing

Python

- No syntax highlighting for

embedded languages

- Makes init.py unwieldy

Recommended Fix:

Move to separate files:

```
templates/
  __init__.py (20 lines - just TemplateWriter)
  project/
    Justfile.j2
    pyproject.toml.j2
  github/
    ci.yml.j2
  code/
    agents.py.j2
```

Estimated Effort: 1 day

8. Inconsistent Directory Structure

Severity: Medium

Impact: Code Navigation

Problem:

30+ top-level files in core/ directory:

```
core/
  memory.py
  substrate.py
  empirica.py
  gamification.py
  goal.py
  reflect.py
  proceed.py
  resume.py
  continue_work.py
  decision_matrix.py
  workflow.py
  ... (20+ more files)
```

Issue: Related functionality scattered,

unclear organization

Recommended Fix:

Group by domain:

```
core/
  game/               # Gamification
    gamification.py
    goal.py
    achievements.py

  decision/           # Decision systems
    decision_matrix.py
    workflow.py
    proceed.py
```

Estimated Effort: 2 days

9. Generic Exception Handlers Remaining

Severity: Medium

Impact: Error Visibility

Problem:

32+ instances of except Exception:

(too broad):

```
except Exception:  # Catches too much
    return {}
```

Note: While less severe than bare except:, still hides specific errors.

Recommended Fix:

Replace with specific exception types:

```
except (json.JSONDecodeError, FileNotFoundError) as e:
    logger.error(f"Failed to load config: {e}")
    raise ConfigurationError(...) from e
```

Estimated Effort: 2 days

10. Magic Numbers/Strings Scattered

Severity: Medium

Impact: Maintainability

Problem:

Despite centralization effort, some

remain:

- Hardcoded font sizes in

foundation_v2.py (6 values)

- Color strings in various files

- Hardcoded thresholds in

gamification.py

Recommended Fix:

Complete config centralization:

```
# config/gamification.py

INTEGRITY_DEFAULT = 100.0

INSIGHT_PER_LEVEL = 100.0

XP_BASE = 50.0



# config/typography.py

FONT_SIZE_TITLE = 24

FONT_SIZE_SUBTITLE = 18
```

Estimated Effort: 0.5 days

Low Priority Issues

11. Inconsistent Naming Conventions

Severity: Low

Impact: Code Clarity

Examples:

- processtavernhook() vs

processcommandhook()

- continuework.py (workaround for

Python keyword)

- Mixed camelCase/snakecase in

some places

Recommended Fix:

Establish and document naming

standards, refactor gradually.

Estimated Effort: 1 day

12. Performance Optimization Opportunities

Severity: Low

Impact: User Experience

Areas:

- Git operations in visualizer (multiple

subprocess calls)

- No caching in expensive API

endpoints

- generate_html() could be optimized

Recommended Fix:

- Add response caching

- Batch git operations

- Profile and optimize hot paths

Estimated Effort: 1-2 days

13. Test Coverage Gaps

Severity: Low (but important for

production)

Impact: Reliability

Problem:

Limited automated test coverage:

- No unit tests for most managers

- Integration tests missing

- No CI/CD test automation

Recommended Fix:

Create test structure:

```
tests/
  unit/
    core/
      test_memory.py
      test_substrate.py
    cli/
      commands/
        test_project.py
  integration/
    test_project_creation.py
```

Estimated Effort: 3-5 days

Known Limitations

Platform Support

Windows:

- [!] Partial support

- Some terminal features don't work

(TTY operations)

- Dashboard may have rendering

issues

Recommendation: Document clearly,

test on Windows, fix critical issues

Scale Limits

Not tested beyond:

- ~1000 agents

- ~100MB JSONL logs

- ~500 dashboard events

Potential Issues:

- Dashboard performance

degradation

- Memory usage with large datasets

- JSONL append performance

Recommendation: Add benchmarks,

document limits

Feature Completeness

Incomplete Features:

1. Evolution cycle automation

(partially implemented)

2. Scint Gym (some error types

under-tested)

3. Karma system (unimplemented)

4. Multi-agent coordination (planned)

5. Distributed evaluation (planned)

Recommendation: Clearly mark

experimental features, set user

expectations

Technical Debt Summary

By Severity

Severity	Count	Top Priority
Critical	0	N/A (all fixed!)
High	5	Split god objects
Medium	5	Extract templates
Low	3	Naming consistency

By Category

Category	Issues	Effort (days)
Architecture	5	10-15
Code Organization	3	4-5
Error Handling	1	2
Testing	1	3-5
Documentation	0	0 (complete!)
Performance	1	1-2

Total Estimated Effort: 20-29 days

(4-6 weeks)

Future Considerations

1. Plugin System

Status: Not implemented

Priority: Low

Effort: 1-2 weeks

Allow third-party extensions:

- Custom agents

- Custom fitness functions

- Custom narrative grammars

2. Distributed Evaluation

Status: Planned

Priority: Medium

Effort: 2-3 weeks

Enable multi-machine fitness testing:

- Worker pool for parallel evaluation

- Result aggregation

- Fault tolerance

3. Advanced Analytics

Status: Basic implementation

Priority: Medium

Effort: 1-2 weeks

Enhanced dashboard:

- Real-time charts

- Phylogenetic visualization

- Fitness landscape maps

- Convergence analysis

4. API Versioning

Status: Not implemented

Priority: Low

Effort: 1 week

Proper API versioning:

- /v1/, /v2/ endpoints

- Deprecation warnings

- Migration guides

5. Multi-Language Support

Status: Python only

Priority: Low

Effort: Unknown

Agents in other languages:

- JavaScript/TypeScript agents

- Rust agents

- Language-agnostic protocol

Prioritization Matrix

Phase 2 Recommendations (Next 2-3 weeks)

Must Do:

1. Split main.py god object (High, 3-4

days)

2. Refactor visualizer.py (High, 2-3

days)

3. Resolve foundation.py duplication

(High, 1-2 days)

4. Create Manager interface (High, 2

days)

Should Do:

5. Extract HTML templates (Medium,

1-2 days)

6. Reorganize core/ directory

(Medium, 2 days)

Could Do:

7. Fix remaining generic exceptions

(Medium, 2 days)

8. Add unit tests (Low, 3-5 days)

Phase 3 Recommendations (Weeks 4-6)

Must Do:

1. Complete or remove Karma

system

2. Detect/fix circular dependencies

3. Extract template files

Should Do:

4. Performance optimization

5. Expanded test coverage

6. Complete documentation

Issue Tracking

Use GitHub Issues for:

- Bug reports

- Feature requests

- Architecture discussions

Labels:

- critical - Security, data loss, broken

core functionality

- high - Maintainability, testability,

architecture

- medium - Code quality, organization

- low - Polish, documentation, minor

improvements

- technical-debt - Refactoring needed

- help-wanted - Good for contributors

Conclusion

Post Phase 1 refactoring, Waft's

codebase is significantly improved

but still has work remaining:

[x] Strengths:

- No critical error handling issues

- Comprehensive logging

- Centralized configuration

- Excellent documentation

- Clear roadmap

[!] Areas for Improvement:

- God objects need decomposition

- Some code duplication remains

- Test coverage could be better

- Some features incomplete

Overall Assessment: Waft is in good

shape for continued development.

Phase 1 stabilization sets a strong

foundation for Phase 2-4

improvements.

Last Updated: 2026-01-10

Next Review: After Phase 2

completion

Maintained by: Waft Team