Waft Codebase Refactoring Plan

Date: 2026-01-10

Status: In Progress

Estimated Total Effort: 6-11 weeks

Executive Summary

The Waft project audit identified 47 issues across code quality, architecture, design patterns, and organization:

- 5 Critical issues requiring immediate attention
- 12 High priority issues impacting maintainability
- 18 Medium priority issues creating technical debt
- 8 Low priority issues for polish

This plan breaks the refactoring into 4

phases: Stabilization -> Architecture

-> Organization -> Completion.

Issue Severity Matrix

```
Severity  |  Count  |  Impact  |  Examples

**Critical**  |  5  |  Security, Stability  |  Bare except clause

**High**  |  12  |  Maintainability  |  Code duplication, Poor s

**Medium**  |  18  |  Code quality  |  Magic strings, Dead code

**Low**  |  8  |  Polish  |  Inconsistent naming
```

Phase 1: STABILIZATION (Week 1-2)

Goal: Fix critical error handling and

remove code smells that hide bugs.

1.1 Fix Bare Except Clauses (CRITICAL)

Effort: 1 day | Files: 6 | Lines: 11

File	Line	Current	Fix
`main.py`	106	`except:` silences TavernKeeper	`except`
`visualizer.py`	145, 151, 176, 221, 241, 298	Bare excepts	
`report.py`	TBD	Unknown exception silenced	Specific
`resume.py`	TBD	Bare except in critical path	`except`
`goal.py`	TBD	Goal processing silently fails	`except`
`ui/dashboard.py`	TBD	2 bare excepts	Specific except

Actions:

```
# BEFORE:
try:
    tavern_keeper = TavernKeeper()
except:
    tavern_keeper = None


# AFTER:
try:
    tavern_keeper = TavernKeeper()
except (ImportError, AttributeError, FileNotFoundError) as e:
    logger.warning(f"TavernKeeper initialization failed: {e}")
    tavern_keeper = None
```

1.2 Replace Generic Exception Handlers (HIGH)

Effort: 2 days | Files: 8 | Count: 32

Target files:

- visualizer.py - Replace except

Exception: in git methods

- continuework.py - Specific

exceptions for workflow

- sessionanalytics.py - Use

sqlite3.Error instead of Exception

Template:

```
# BEFORE:
except Exception as e:
    return {}


# AFTER:
except (json.JSONDecodeError, FileNotFoundError) as e:
    logger.error(f"Failed to load config: {e}", exc_info=True)
    raise ConfigurationError(f"Configuration load failed: {e}") f
```

1.3 Add Logging Infrastructure (MEDIUM)

Effort: 1 day

Create centralized logging:

```
# src/waft/logging.py

import logging

from pathlib import Path



def get_logger(name: str) -> logging.Logger:

    logger = logging.getLogger(f"waft.{name}")

    if not logger.handlers:

        handler = logging.StreamHandler()

        formatter = logging.Formatter(

            '%(asctime)s - %(name)s - %(levelname)s - %(message)s

        )
```

Update all files to use:

```
from waft.logging import get_logger

logger = get_logger(__name__)
```

1.4 Extract Configuration Constants (MEDIUM)

Effort: 1 day | Files: 5+

Create:

```
src/waft/config/
  __init__.py
  theme.py        # Emojis, colors
  abilities.py    # Command -> Ability mapping
  thresholds.py   # Gamification thresholds
  defaults.py     # Default values
```

Example - theme.py:

```
"""Visual theme constants for CLI output."""


class Emoji:
    WAFT = "~"
    DICE = ""
    INTEGRITY = ""
    INSIGHT = "*"
    SUCCESS = "*"
    SPARKLES = "*"


class Color:
    SUCCESS = "green"
    ERROR = "red"
    WARNING = "yellow"
    INFO = "cyan"
    TITLE = "bold cyan"
```

Example - abilities.py:

```
"""Command -> D&D Ability mapping."""

from dataclasses import dataclass

from typing import Literal


AbilityType = Literal["STR", "DEX", "CON", "INT", "WIS", "CHA"]


@dataclass(frozen=True)

class CommandAbility:

    command: str

    ability: AbilityType

    dc: int = 10


COMMAND_ABILITIES = [

    CommandAbility("new", "CHA"),

    CommandAbility("verify", "CON"),
```

1.5 Remove Dead Code (MEDIUM)

Effort: 0.5 day | Lines saved: 893

Files to remove:

-

src/waft/core/decisionmatrixv1backup.py

(620 lines) - unused backup

- src/waft/testloop.py (273 lines) -

incomplete test harness

Actions:

```
git rm src/waft/core/decision_matrix_v1_backup.py

git rm src/waft/test_loop.py

git commit -m "refactor: Remove dead code (backup files, unused t
```

Phase 2: ARCHITECTURE (Week 3-5)

Goal: Break down god objects,

improve modularity.

2.1 Split main.py into Command Modules (CRITICAL)

Effort: 3-4 days | Current: 2019 lines |

Target: <200 lines per module

New structure:

```
src/waft/cli/
  __init__.py
  app.py                    # Typer app setup + routing (150 lines
  utils.py                  # Shared utilities (_process_tavern_h


  commands/
    __init__.py
    project.py              # new, init, verify, info (300 lines)
    dependencies.py         # sync, add (150 lines)
    empirica.py             # session, finding, unknown, check, a
    gamification.py         # dashboard, stats, character, chroni
    decision.py             # Decision engine commands (250 lines
    analytics.py            # Analytics commands (200 lines)
    goals.py                # Goal commands (150 lines)
    git_ops.py              # Git operations (200 lines)
```

Migration plan:

1. Create directory structure

2. Extract commands by category

(one category per day)

3. Update imports in each module

4. Update pyproject.toml entry point

5. Test each command after

migration

6. Remove old main.py

Example - commands/project.py:

```
"""Project management commands."""

import typer

from pathlib import Path

from rich.console import Console

from waft.core.substrate import SubstrateManager

from waft.core.memory import MemoryManager

from waft.logging import get_logger



logger = get_logger(__name__)

console = Console()

app = typer.Typer(help="Project management commands")



@app.command()

def new(

    name: str,
```

2.2 Refactor visualizer.py God Object (CRITICAL)

Effort: 2-3 days | Current: 2338 lines |

Target: <300 lines per class

New structure:

```
src/waft/core/visualization/
  __init__.py
  visualizer.py             # Main coordinator (200 lines)
  collectors/
    __init__.py
    git_collector.py        # Git status collection (150 lines)
    system_collector.py     # System info collection (100 lines)
    work_collector.py       # Work effort tracking (150 lines)
    analytics_collector.py  # Analytics aggregation (200 lines)
  renderers/
```

Implementation:

```
# visualizer.py (new)
from .collectors import (
    GitCollector, SystemCollector, WorkCollector, AnalyticsCollec
)
from .renderers import HtmlRenderer


class Visualizer:
    def __init__(self, project_path: Path):
        self.project_path = project_path
        self.git = GitCollector(project_path)
        self.system = SystemCollector()
        self.work = WorkCollector(project_path)
        self.analytics = AnalyticsCollector(project_path)
        self.renderer = HtmlRenderer()
```

2.3 Decompose agent/base.py (HIGH)

Effort: 2 days | Current: 924 lines |

Target: <200 lines per class

New structure:

```
src/waft/core/agent/

  base.py               # Core OODA cycle (200 lines)

  genome.py             # Genome tracking (150 lines)

  inventory.py          # Item management (200 lines)

  reproduction.py       # Conjugate, spawn (200 lines)

  traits.py             # Archetypes, anatomy (150 lines)
```

Example - base.py (refactored):

```
from .genome import AgentGenome
from .inventory import AgentInventory
from .reproduction import AgentReproduction


class BaseAgent:
    """Base agent with OODA cycle (Observe, Decide, Act, Reflect)


    def __init__(self, name: str, archetype: str = "explorer"):
        self.name = name
        self.archetype = archetype


        # Composition instead of inheritance
        self.genome = AgentGenome(self)
        self.inventory = AgentInventory(self)
        self.reproduction = AgentReproduction(self)
```

2.4 Resolve foundation.py Duplication (HIGH)

Effort: 1 day | Impact: Remove 1088

duplicate lines

Analysis needed:

```
# Check which version is actually used

grep -r "from.*foundation import\|from.*foundation_v2 import" src/
```

Decision tree:

1. If only foundationv2.py is used ->

Delete foundation.py

2. If both are used:

- Audit differences

- Migrate all imports to

foundationv2.py

- Add deprecation warning to

foundation.py

- Delete foundation.py in next release

3. If only foundation.py is used ->

Consider if v2 features are needed

Migration guide (if needed):

```
# Foundation.py -> Foundation_v2.py Migration

## Breaking Changes

- `DocumentConfig` now requires `font_config: FontConfig`

- `generate_specimen_d_audit()` renamed to `generate_clinical_rep

## New Features

- Font family selection (FontFamily enum)

- Cover page support

- Metadata rail

- Rule blocks

## Migration Steps

1. Update imports: `from waft.foundation_v2 import ...`

2. Add font config: `font_config=FontConfig(family=FontFamily.HEL
```

Phase 3: ORGANIZATION (Week 6-8)

Goal: Improve module organization,

create clear boundaries.

3.1 Reorganize core/ Directory (MEDIUM)

Effort: 2 days | Current: 30+ files |

Target: <10 subdirectories

Current issues:

- 30+ Python files in core/ (too flat)

- Related functionality scattered

- Unclear organization rationale

New structure:

```
src/waft/core/
  game/                    # Gamification system
    __init__.py
    gamification.py        # Rewards, XP
    goal.py                # Goals
    achievements.py        # (future)

  decision/                # Decision systems
    __init__.py
    decision_matrix.py     # WSM calculator
    workflow.py            # Workflow management
    proceed.py             # PROCEED/HALT gates

  observation/             # Observation & memory
    __init__.py
    resume.py              # Resume work
    reflect.py             # Reflection
```

Migration:

```
# Create new directories

mkdir -p src/waft/core/{game,decision,observation,project,integra



# Move files

git mv src/waft/core/gamification.py src/waft/core/game/

git mv src/waft/core/goal.py src/waft/core/game/



# Update __init__.py files for backward compatibility

# Add imports in src/waft/core/__init__.py
```

3.2 Extract Templates to Separate Files (MEDIUM)

Effort: 1 day | Current: 434 lines in

init.py

New structure:

```
src/waft/templates/
  __init__.py              # 30 lines (just TemplateWriter class
  writer.py                # Template writing logic
  project/
    Justfile.j2            # 60 lines
    pyproject.toml.j2      # 40 lines
    .gitignore.j2          # 20 lines
    README.md.j2           # 50 lines
  github/
    ci.yml.j2              # 50 lines
```

Use Jinja2:

```
# templates/writer.py
from jinja2 import Environment, PackageLoader, select_autoescape
from pathlib import Path


class TemplateWriter:
    def __init__(self):
        self.env = Environment(
            loader=PackageLoader('waft.templates', 'project'),
            autoescape=select_autoescape()
        )


    def render(self, template_name: str, **context) -> str:
        template = self.env.get_template(template_name)
        return template.render(**context)
```

3.3 Create Abstraction Layers (HIGH)

Effort: 2 days

Define Manager interface:

```
# src/waft/core/interfaces.py

from abc import ABC, abstractmethod

from pathlib import Path

from typing import Optional, Tuple



class Manager(ABC):

    """Base interface for all managers."""



    def __init__(self, project_path: Path):

        self.project_path = project_path

        self._initialized = False
```

Update existing managers:

```
# src/waft/core/memory.py
from .interfaces import Manager


class MemoryManager(Manager):
    def initialize(self) -> bool:
        """Initialize the _pyrite structure."""
        try:
            self.pyrite_path.mkdir(parents=True, exist_ok=True)
            # ... create subdirectories ...
            return True
        except OSError as e:
            logger.error(f"Failed to initialize memory: {e}")
            return False


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

3.4 Document Dependencies (HIGH)

Effort: 1 day

Create dependency graph:

```
# Use pydeps or manual analysis

pip install pydeps

pydeps src/waft --max-bacon=2 --show-deps --cluster
```

Document in

docs/ARCHITECTURE.md:

```
# Waft Architecture


## Layer Structure



+-----------------------------------+

| CLI Layer (cli/) | User interaction

+-----------------------------------+

| API Layer (api/) | HTTP endpoints

+-----------------------------------+

| Core Layer (core/) | Business logic

+-----------------------------------+

| Models Layer (models/) | Data

structures
```

```
## Dependency Rules

- CLI can depend on Core, Models

- API can depend on Core, Models

- Core can depend on Models

- Models can depend on nothing (pure data)

- **Never:** Core -> CLI, Core -> API
```

Phase 4: COMPLETION (Week 9-10)

Goal: Complete unfinished features,

improve tests, add documentation.

4.1 Complete Karma System OR Remove (HIGH)

Effort: 3-4 days (complete) OR 0.5

days (remove)

Current state: 5 unimplemented

methods (239 lines)

Option A: Complete Implementation

```
# karma.py

def calculate_karma(self, life_log: Dict[str, Any]) -> float:

    """Calculate karma from life log."""

    actions = life_log.get('actions', [])

    positive_actions = sum(1 for a in actions if a.get('impact', 

    negative_actions = sum(1 for a in actions if a.get('impact', 



    karma = (positive_actions - negative_actions) / max(len(actio

    return max(-1.0, min(1.0, karma))  # Clamp to [-1, 1]



def access_akasha(self, soul_id: str) -> Dict[str, Any]:

    """Access the Akashic records (soul persistence)."""

    akasha_path = self.project_path / "_pyrite" / "akasha" / f"{s

    if not akasha_path.exists():

        return {}
```

Option B: Remove (Recommended)

```
git rm src/waft/karma.py

# Update imports

# Add to docs/FUTURE_FEATURES.md
```

4.2 Add Unit Tests (HIGH)

Effort: 2-3 days

Create test structure:

```
tests/
  unit/
    core/
      test_memory.py
      test_substrate.py
      test_decision_matrix.py
      game/
        test_gamification.py
        test_goal.py
    cli/
      commands/
        test_project.py
        test_dependencies.py
  integration/
    test_project_creation.py
```

Example tests:

```
# tests/unit/core/test_memory.py

import pytest

from pathlib import Path

from waft.core.memory import MemoryManager



def test_initialize_creates_structure(tmp_path):

    manager = MemoryManager(tmp_path)

    assert manager.initialize()

    assert (tmp_path / "_pyrite").exists()

    assert (tmp_path / "_pyrite" / "active").exists()



def test_validate_detects_missing_structure(tmp_path):

    manager = MemoryManager(tmp_path)

    is_valid, error = manager.validate()

    assert not is_valid
```

4.3 Add Documentation (MEDIUM)

Effort: 2 days

Create:

- docs/ARCHITECTURE.md - System

architecture

- docs/CONTRIBUTING.md -

Contribution guide

- docs/API.md - API documentation

- docs/CLI.md - CLI command

reference

Update:

- README.md - Reflect new

structure

- Docstrings in all modules

4.4 Performance Optimization (LOW)

Effort: 1-2 days

Targets:

- Add caching to git operations in

visualizer

- Add response caching to expensive

API endpoints

- Profile and optimize generate_html()

in visualizer

Execution Strategy

Execution Principles

1. Test After Each Change - Verify

functionality preserved

2. Commit Frequently - Small, atomic

commits

3. Document Decisions - Add

comments explaining why

4. Pause and Evaluate - Review

progress weekly

5. Stay DRY - Don't repeat yourself

6. Keep It Simple - Avoid

Weekly Checkpoints

- End of Week 1: Review stabilization changes
- End of Week 3: Review architecture changes
- End of Week 6: Review organization changes
- End of Week 8: Review completion status

Success Metrics

- [ ] All bare except: clauses replaced

- [ ] No file > 500 lines

- [ ] No duplicate code between

foundation.py versions

- [ ] All managers implement Manager

interface

- [ ] Test coverage > 60%

- [ ] Documentation complete

- [ ] No circular dependencies

Risk Management

Risk	Likelihood	Impact	Mitigation
Breaking changes	High	High	Thorough testing, gradual
Merge conflicts	Medium	Medium	Work in feature branch
Scope creep	Medium	Medium	Stick to plan, defer new f
Test failures	High	Medium	Fix immediately, don't pro
Performance regression	Low	Medium	Profile before/aft

Rollback Plan

If issues arise:

1. Each phase in separate branch:

refactor/phase-1, refactor/phase-2,

etc.

2. Tag before each merge:

v0.3.1-pre-refactor, v0.3.2-phase1,

etc.

3. Keep old code commented for 1

release cycle

4. Document breaking changes in

CHANGELOG.md

Next Steps

1. Review this plan with stakeholders

2. Create GitHub issues for each

major task

3. Set up project board with phases

4. Begin Phase 1: Stabilization

5. Commit and push this plan to

repository