Debugging and Testing Guide

This guide shows how to use the AI Orchestrator for debugging issues and running tests effectively.

Table of Contents

Overview

Debugging Tools

Tool	Purpose	When to Use
`run_project`	Execute and capture errors	Initial error discovery
`analyze_errors`	Deep error analysis	Understanding root cause
`fix_issues`	Apply fixes	After analysis
`verify_project`	Full fix loop	Complex multi-error situations

Testing Tools

Tool	Purpose	When to Use
`test_project`	Run test suite	Verify functionality
`verify_project`	Test + fix loop	Fix failing tests automatically

Debugging Workflow

Step 1: Reproduce the Error

```
@ai-orchestrator run_project("/path/to/project")
```

Example Output:

```
❌ Project Execution Failed

Exit Code: 1
Stdout: Starting server...
Stderr:
  TypeError: Cannot read property 'map' of undefined
    at renderUsers (/app/components/UserList.js:15:23)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)
```

Step 2: Analyze the Error

```
@ai-orchestrator analyze_errors("/path/to/project")
```

Example Output:


```
{
  "errors": [
    {
      "type": "runtime",
      "category": "null_reference",
      "file": "components/UserList.js",
      "line": 15,
      "message": "Cannot read property 'map' of undefined",
      "root_cause": "'users' array is undefined when component renders",
      "stack_trace": [...],
      "suggested_fixes": [
        {
          "description": "Add default empty array for users",
          "confidence": 0.9,
          "code": "const users = props.users || [];"
        },
        {
          "description": "Add null check before map",
          "confidence": 0.85,
          "code": "users && users.map(...)"
        }
      ]
    }
  ]
}
```


Step 3: Apply Fix

```
@ai-orchestrator fix_issues("/path/to/project")
```

Example Output:

```
✅ Fix Applied:
   File: components/UserList.js
   Line: 12
   Change: Added default value for users prop

Before:
  const { users } = props;
  return users.map(user => <UserCard key={user.id} {...user} />);

After:
  const users = props.users || [];
  return users.map(user => <UserCard key={user.id} {...user} />);
```

Step 4: Verify Fix

```
@ai-orchestrator run_project("/path/to/project")
```

Example Output:

```
✅ Project Started Successfully

Status: RUNNING
Output: Server listening on http://localhost:3000
```

Testing Workflow

Running Tests

```
@ai-orchestrator test_project("/path/to/project")
```

Example Output (pytest):

```
╔═══════════════════════════════════════════════════════════════╗
║                        TEST RESULTS                           ║
╠═══════════════════════════════════════════════════════════════╣
║ Framework: pytest                                             ║
║ Duration: 4.32s                                               ║
╠═══════════════════════════════════════════════════════════════╣
║ SUMMARY                                                       ║
║   ✅ Passed:  12                                              ║
║   ❌ Failed:   2                                              ║
║   ⏭️  Skipped:  1                                             ║
║   Total:     15                                               ║
╠═══════════════════════════════════════════════════════════════╣
║ FAILURES                                                      ║
╠═══════════════════════════════════════════════════════════════╣
║ test_user_auth.py::test_login_invalid_password                ║
║   AssertionError: Expected 401, got 500                       ║
║   Line 45: assert response.status_code == 401                 ║
╠═══════════════════════════════════════════════════════════════╣
║ test_tasks.py::test_create_task_without_title                 ║
║   KeyError: 'error'                                           ║
║   Line 23: assert 'error' in response.json                    ║
╚═══════════════════════════════════════════════════════════════╝
```

Auto-Fixing Failed Tests

```
@ai-orchestrator verify_project("/path/to/project")
```

The verification loop will:
1. Run tests
2. Analyze failures
3. Generate fixes
4. Re-run tests
5. Repeat until all pass (or max cycles)

Common Debugging Scenarios

Scenario 1: Server Won't Start

```
# Step 1: Try to run
@ai-orchestrator run_project("/path/to/project")

# Output shows: "Error: EADDRINUSE: address already in use :::3000"

# Step 2: Analyze
@ai-orchestrator analyze_errors("/path/to/project")

# Output shows: port_conflict, process using port

# Step 3: Fix (kill process)
@ai-orchestrator fix_issues("/path/to/project", strategy="kill_process")

# Or change port
@ai-orchestrator fix_issues("/path/to/project", strategy="change_port")
```

Scenario 2: Import/Module Errors

```
# Error: ModuleNotFoundError: No module named 'requests'

# Analyze identifies missing dependency
@ai-orchestrator analyze_errors("/path/to/project")

# Fix installs it
@ai-orchestrator fix_issues("/path/to/project")
# → pip install requests
```

Scenario 3: Database Connection Issues

```
# Error: psycopg2.OperationalError: connection refused

@ai-orchestrator analyze_errors("/path/to/project")

# Analysis shows:
# - Database not running, or
# - Wrong connection string, or
# - Missing environment variable

# Fix options:
# 1. Start database
# 2. Update .env with correct URL
# 3. Fall back to SQLite for development
```

Scenario 4: Type Errors in JavaScript

```
# Error: TypeError: Cannot read property 'x' of undefined

@ai-orchestrator analyze_errors("/path/to/project")

# Analysis:
# - Object is undefined at runtime
# - Missing null check
# - API response different than expected

# Fix adds defensive coding:
# const value = obj?.x ?? defaultValue;
```

Scenario 5: Test Assertion Failures

```
# Test fails: Expected 200, got 404

@ai-orchestrator analyze_errors("/path/to/project")

# Analysis shows:
# - Route not defined
# - Route defined with different path
# - Route requires authentication

# Fixes:
# 1. Add missing route
# 2. Correct test to use right path
# 3. Add auth token to test
```

Scenario 6: Infinite Loop / Timeout

```
# Project hangs, doesn't respond

@ai-orchestrator run_project("/path/to/project", timeout=30)

# Times out after 30 seconds
# Analysis may show:
# - Infinite loop in code
# - Blocking synchronous operation
# - Waiting for unavailable resource

# AI analyzes the code for loop conditions
```

Test Framework Support

Supported Frameworks

Framework	Language	Auto-Detection
pytest	Python	`pytest.ini`, `conftest.py`
unittest	Python	`test_*.py` pattern
jest	JavaScript	`jest.config.js`, `package.json`
mocha	JavaScript	`mocha.opts`, `.mocharc`
vitest	JavaScript	`vitest.config.ts`
Django	Python	`manage.py test`

Framework-Specific Features

pytest

```
@ai-orchestrator test_project("/path/to/project")

# Parses output for:
# - passed/failed/skipped counts
# - fixture errors
# - collection errors
# - assertion details
```

Jest

```
@ai-orchestrator test_project("/path/to/project")

# Parses output for:
# - Test suites passed/failed
# - Individual test results
# - Snapshot failures
# - Coverage report
```

Running Specific Tests

```
# Run only specific test file
@ai-orchestrator test_project("/path/to/project",
  test_path="tests/test_auth.py")

# Run tests matching pattern
@ai-orchestrator test_project("/path/to/project",
  pattern="*login*")
```

Tips and Best Practices

1. Start with Analysis

Always analyze before fixing:

```
# Good
@ai-orchestrator analyze_errors("/path/to/project")
# ... review analysis ...
@ai-orchestrator fix_issues("/path/to/project")

# Bad (might apply wrong fix)
@ai-orchestrator fix_issues("/path/to/project")  # without analysis
```

2. Use Verification Loop for Multiple Errors

```
# If you have multiple errors:
@ai-orchestrator verify_project("/path/to/project")

# Instead of:
@ai-orchestrator fix_issues("/path/to/project")  # might only fix one
@ai-orchestrator run_project("/path/to/project")  # check
@ai-orchestrator fix_issues("/path/to/project")  # next error
# ... repeat manually
```

3. Check Backups Before Major Changes

```
# Backups are created automatically at:
# .backups/<filename>.<timestamp>

# If fix causes new problems:
cp .backups/app.py.1708234567 app/app.py
```

4. Review AI-Generated Fixes

For low-confidence fixes:

```
@ai-orchestrator analyze_errors("/path/to/project")

# If confidence < 0.7, review the suggestion before applying
# The fix will be suggested but not auto-applied
```

5. Combine with Code Review

After fixing issues, review the changes:

```
# After fixing
@ai-orchestrator fix_issues("/path/to/project")

# Review what was changed
@ai-orchestrator orchestrate_task("Review recent changes in /path/to/project for cor-
rectness")
```

6. Use Timeouts for Potentially Hanging Code

```
@ai-orchestrator run_project("/path/to/project", timeout=60)

# Prevents infinite loops from blocking
```

7. Test After Every Significant Change

```
# Make changes
@ai-orchestrator orchestrate_task("Add validation to user input")

# Always test
@ai-orchestrator test_project("/path/to/project")
```

Debug Output Interpretation

Error Categories

```
╔═══════════════════════════════════════════════════════════╗
║                    ERROR CATEGORIES                       ║
╠═══════════════════════════════════════════════════════════╣
║ SYNTAX          → Code structure errors                   ║
║ RUNTIME         → Errors during execution                 ║
║ DEPENDENCY      → Missing packages/modules                ║
║ IMPORT          → Cannot import module/symbol             ║
║ TYPE            → Type mismatch errors                    ║
║ CONFIGURATION   → Config/env issues                       ║
║ NETWORK         → Connection/API errors                   ║
║ DATABASE        → DB connection/query errors              ║
║ PERMISSION      → File/resource access denied             ║
║ PORT            → Port in use conflicts                   ║
║ MEMORY          → Out of memory errors                    ║
║ TIMEOUT         → Operation took too long                 ║
║ LOGIC           → Incorrect behavior (hardest to auto-fix)║
╚═══════════════════════════════════════════════════════════╝
```

Fix Confidence Guide

```
Confidence ≥ 0.9:  Auto-applied (safe, deterministic fixes)
Confidence ≥ 0.7:  Applied with backup
Confidence ≥ 0.5:  Suggested, requires approval
Confidence < 0.5:  Suggestion only, manual review needed
```

Related Guides

- 04_auto_fix_workflow.md (04_auto_fix_workflow.md) - Auto-fix examples
- 05_full_development_cycle.md (05_full_development_cycle.md) - Complete workflow
- DEVELOPMENT_WORKFLOW.md (../DEVELOPMENT_WORKFLOW.md) - Workflow overview