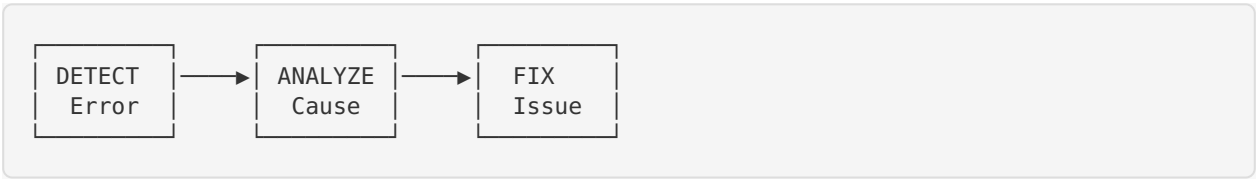
Auto-Fix Workflow Example

This guide demonstrates how the AI Orchestrator's auto-fix capabilities handle common errors automatically.

Table of Contents

Overview

The auto-fix system works in three stages:



1. **DETECT**: Identify errors from project output
2. **ANALYZE**: Categorize and understand root cause
3. **FIX**: Apply appropriate fix (automatic or AI-generated)

Example 1: Missing Dependencies

The Error

```
$ npm start
error: Cannot find module 'express'
```

Auto-Fix Process

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

Output:

```
{
  "error_type": "dependency",
  "missing_package": "express",
  "fix_strategy": "install_dependency",
  "fix_command": "npm install express",
  "confidence": 0.95
}
```

Fix Applied

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

Result:

```
✅ Fix Applied:
   Command: npm install express
   Status: Success
   Package installed: express@4.18.2
```

Before/After

Before	After
❌ `Cannot find module 'express'`	✅ Server starts on port 3000

Example 2: Syntax Errors

The Error

```
# app.py - Line 15
def calculate_total(items):
    total = 0
    for item in items:
        total += item.price
    return total  # Missing closing parenthesis on previous line actually
```

Error:

```
SyntaxError: unexpected EOF while parsing at line 16
```

Auto-Fix Process

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

Analysis:

```
{
  "error_type": "syntax",
  "file": "app.py",
  "line": 16,
  "issue": "unmatched_parenthesis",
  "confidence": 0.85
}
```

AI-Generated Fix

Claude analyzes the code and generates:

```
# BEFORE (broken)
def process_order(order
    items = order.items
    return calculate_total(items)

# AFTER (fixed)
def process_order(order):
    items = order.items
    return calculate_total(items)
```

Fix Applied

```
✅ Fix Applied:
   File: app.py
   Change: Added missing ')' and ':' on line 10
   Backup: app.py.backup.1708234567
```

Example 3: Import Errors

The Error

```
# views.py
from flask import Flask, render_template, jsonify
from models import User, Product  # Works
from utils import validate_email  # Error!
```

Error:

```
ModuleNotFoundError: No module named 'utils'
```

Analysis

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

Output:

```
{
  "error_type": "import_error",
  "missing_module": "utils",
  "possible_causes": [
    "File doesn't exist",
    "Wrong import path",
    "Missing __init__.py"
  ],
  "file_exists": false
}
```

AI Fix

The orchestrator checks the codebase and suggests:

```
# Option 1: If utils exists in different location
from app.utils import validate_email

# Option 2: If function exists in different file
from helpers import validate_email

# Option 3: Create the missing module
# Creates utils.py with:
def validate_email(email):
    import re
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return bool(re.match(pattern, email))
```

Result

```
✅ Fix Applied:
   Action: Created utils.py with validate_email function
   Reason: Function was referenced but not defined
```

Example 4: Port Conflicts

The Error

```
$ npm start
Error: listen EADDRINUSE: address already in use :::3000
```

Analysis

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

Output:

```
{
  "error_type": "port_conflict",
  "port": 3000,
  "process_using_port": "node (PID: 12345)",
  "fix_options": [
    "kill_process",
    "use_different_port"
  ]
}
```

Fix Options

Option 1: Kill existing process

```
@ai-orchestrator fix_issues("/path/to/project", strategy="kill_process")
```

```
✅ Process killed:
   PID: 12345
   Port 3000 now free
```

Option 2: Change port

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

```
✅ Port updated:
   File: .env
   Change: PORT=3000 → PORT=3001
```

Example 5: Environment Variables

The Error

```
// database.js
const mongoose = require('mongoose');
mongoose.connect(process.env.MONGODB_URI);
```

Error:

```
MongooseError: The `uri` parameter to `openUri()` must be a string, got "undefined"
```

Analysis

```
{
  "error_type": "configuration",
  "missing_env_var": "MONGODB_URI",
  "has_env_example": true,
  "env_example_path": ".env.example"
}
```

Auto-Fix

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

Actions taken:

```
1. Found .env.example with MONGODB_URI template
2. Created .env from .env.example
3. Set MONGODB_URI=mongodb://localhost:27017/myapp

✅ Fix Applied:
   File: .env created
   Variables: MONGODB_URI set to local default
   Note: Update with production values before deploying
```

Example 6: Runtime Errors

The Error

```
# api.py
@app.route('/user/<id>')
def get_user(id):
    user = User.query.get(id)
    return jsonify(user.to_dict())  # Error when user is None!
```

Error:

```
AttributeError: 'NoneType' object has no attribute 'to_dict'
```

Analysis

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

Output:

```
{
  "error_type": "runtime",
  "category": "null_reference",
  "file": "api.py",
  "line": 5,
  "issue": "accessing method on potentially None object",
  "root_cause": "User.query.get() returns None if not found"
}
```

AI-Generated Fix

Claude analyzes and generates:

```
# BEFORE
@app.route('/user/<id>')
def get_user(id):
    user = User.query.get(id)
    return jsonify(user.to_dict())

# AFTER
@app.route('/user/<id>')
def get_user(id):
    user = User.query.get(id)
    if user is None:
        return jsonify({"error": "User not found"}), 404
    return jsonify(user.to_dict())
```

Result

```
✅ Fix Applied:
   File: api.py
   Change: Added null check with 404 response
   Confidence: 0.92
   Validated: Syntax check passed
```

Fix Confidence Levels

The orchestrator assigns confidence to each fix:

Confidence	Action	Examples
0.9+	Auto-apply	Dependency install, port kill
0.7-0.9	Apply with backup	Syntax fixes, imports
0.5-0.7	Suggest only	Logic changes, refactors
< 0.5	Manual review	Architecture issues

Verification After Fix

Always verify fixes worked:

```
# Option 1: Quick verification
@ai-orchestrator run_project("/path/to/project")

# Option 2: Full verification loop
@ai-orchestrator verify_project("/path/to/project")
```

The verification loop will:

1. Run the project

2. Run tests
3. If still failing, analyze and fix again
4. Repeat until success or max cycles

Next Steps

- See 05_full_development_cycle.md (05_full_development_cycle.md) for complete project walk-through
- See 06_debugging_and_testing.md (06_debugging_and_testing.md) for debugging strategies