Complete Testing Guide - Incremental Build

Testing Philosophy

Test each component in isolation before integration testing. This incremental approach helps identify issues quickly and ensures each layer works before adding complexity.

Testing Levels

```
Level 1: Module Tests (Individual components)
    ↓
Level 2: Device Tests (Single device functionality)
    ↓
Level 3: API Tests (Alpaca endpoints)
    ↓
Level 4: Integration Tests (Multiple devices)
    ↓
Level 5: Client Tests (Real astronomy software)
```

Level 1: Module Testing

1.1 Test Configuration Module

File: `config.py`

Test Script: `test_config.py`

python

```python
#!/usr/bin/env python3
"""Test configuration module"""

import config

def test_config():
    print("Testing configuration...")

    # Test server config
    assert config.SERVER_PORT == 5555
    assert config.SERVER_HOST == '0.0.0.0'
    print("✓ Server config OK")

    # Test device config
    assert 'telescope' in config.DEVICES
    assert 'camera_zwo' in config.DEVICES
    assert 'camera_touptek' in config.DEVICES
    print("✓ Device config OK")

    # Test error codes
    assert config.ASCOM_ERROR_CODES['NOT_CONNECTED'] == 1031
    print("✓ Error codes OK")

    print("\n✅ Configuration module PASSED\n")

if __name__ == '__main__':
    test_config()
```

Run:

bash

```bash
cd ~/onstepx-alpaca
source venv/bin/activate
python3 test_config.py
```

Expected Output:

```
Testing configuration...
✓ Server config OK
✓ Device config OK
✓ Error codes OK

✅ Configuration module PASSED
```

1.2 Test Helper Functions

Test Script: test_helpers.py

python

```python
#!/usr/bin/env python3
"""Test helper functions"""

import alpaca_helpers as helpers

def test_coordinate_parsing():
    print("Testing coordinate parsing...")

    # Test RA parsing
    ra = helpers.parse_ra_to_hours("12:30:45")
    assert abs(ra - 12.5125) < 0.001
    print(f"  RA parse: {ra:.4f} hours")

    # Test Dec parsing
    dec = helpers.parse_dec_to_degrees("+45:30:15")
    assert abs(dec - 45.504167) < 0.001
    print(f"  Dec parse: {dec:.4f} degrees")

    # Test RA formatting
    ra_str = helpers.format_ra_hours(12.5125)
    print(f"  RA format: {ra_str}")

    # Test Dec formatting
    dec_str = helpers.format_dec_degrees(45.504167)
    print(f"  Dec format: {dec_str}")

    print("✓ Coordinate parsing OK\n")

def test_validation():
    print("Testing validation...")

    valid, msg = helpers.validate_range(50, 0, 100, "test")
    assert valid == True
    print("  Range validation OK")

    valid, msg = helpers.validate_range(150, 0, 100, "test")
    assert valid == False
    print("  Range rejection OK")

    print("✓ Validation OK\n")

def test_clamp():
    print("Testing clamp...")

    assert helpers.clamp(50, 0, 100) == 50
    assert helpers.clamp(-10, 0, 100) == 0
    assert helpers.clamp(150, 0, 100) == 100
```

```
    print("✓ Clamp OK\n")

if __name__ == '__main__':
    test_coordinate_parsing()
    test_validation()
    test_clamp()
    print("✅ Helper functions PASSED\n")
```

Run:

```
bash

python3 test_helpers.py
```

Level 2: Device Testing

2.1 Test Telescope Module (Without Hardware)

Test Script: `test_telescope_mock.py`

```python
#!/usr/bin/env python3
"""Test telescope module without hardware"""

from telescope import OnStepXMount
import alpaca_helpers as helpers

def test_telescope_initialization():
    print("Testing telescope initialization...")

    mount = OnStepXMount(port=None, baudrate=9600)

    assert mount.is_connected == False
    assert mount.tracking_rate == 0  # Sidereal
    assert mount.site_latitude == 0.0
    print("✓ Initialization OK\n")

def test_coordinate_conversion():
    print("Testing internal coordinate methods...")

    mount = OnStepXMount()

    # These methods don't require connection
    result = mount.destination_side_of_pier(12.0, 45.0)
    print(f"  Pier side calculation: {result}")

    print("✓ Coordinate conversion OK\n")

if __name__ == '__main__':
    test_telescope_initialization()
    test_coordinate_conversion()
    print("✅ Telescope module PASSED (mock)\n")
```

2.2 Test Telescope with Hardware

Test Script: `test_telescope_hardware.py`

python

```python
#!/usr/bin/env python3
"""Test telescope with actual OnStepX hardware"""

from telescope import OnStepXMount
import time

def test_connection():
    print("Testing telescope connection...")

    mount = OnStepXMount()
    success = mount.connect()

    if not success:
        print("✗ Connection failed - check USB connection")
        return False

    print(f"✓ Connected on {mount.port}")
    print(f"  Site Latitude: {mount.site_latitude}")
    print(f"  Site Longitude: {mount.site_longitude}")

    return mount

def test_position_reading(mount):
    print("\nTesting position reading...")

    ra = mount.get_ra()
    dec = mount.get_dec()
    alt = mount.get_altitude()
    az = mount.get_azimuth()
    lst = mount.get_sidereal_time()

    print(f"  RA: {ra:.4f} hours")
    print(f"  Dec: {dec:.4f} degrees")
    print(f"  Alt: {alt:.4f} degrees")
    print(f"  Az: {az:.4f} degrees")
    print(f"  LST: {lst:.4f} hours")

    print("✓ Position reading OK\n")

def test_tracking(mount):
    print("Testing tracking...")

    is_tracking = mount.get_tracking()
    print(f"  Current tracking: {is_tracking}")

    # Toggle tracking
    mount.set_tracking(not is_tracking)
```

```
    time.sleep(1)
    new_tracking = mount.get_tracking()

    # Restore original state
    mount.set_tracking(is_tracking)

    print("✓ Tracking control OK\n")

def test_park_status(mount):
    print("Testing park status...")

    at_park = mount.get_at_park()
    at_home = mount.get_at_home()

    print(f"  At park: {at_park}")
    print(f"  At home: {at_home}")

    print("✓ Park status OK\n")

if __name__ == '__main__':
    print("="*60)
    print("TELESCOPE HARDWARE TEST")
    print("="*60)
    print("⚠️  Ensure OnStepX is connected via USB")
    print("="*60 + "\n")

    mount = test_connection()
    if mount:
        test_position_reading(mount)
        test_tracking(mount)
        test_park_status(mount)

        mount.disconnect()
        print("✅ Telescope hardware PASSED\n")
    else:
        print("✗ Tests aborted - no connection\n")
```

Run:

```
bash

# Connect OnStepX mount via USB first!
python3 test_telescope_hardware.py
```

2.3 Test ZWO Camera (Without Hardware)

Test Script: test_camera_zwo_mock.py

python

```python
#!/usr/bin/env python3
"""Test ZWO camera module without hardware"""

from camera_zwo import ZWOCamera, ZWO_AVAILABLE

def test_imports():
    print("Testing ZWO imports...")

    if ZWO_AVAILABLE:
        print("✓ ZWO SDK available")
    else:
        print("✗ ZWO SDK not available")
        print("  Install: pip install zwoasi")
        return False

    return True

def test_initialization():
    print("\nTesting ZWO initialization...")

    try:
        camera = ZWOCamera(camera_id=0)
        print("✓ Camera object created")

        assert camera.is_connected == False
        assert camera.camera_state == 0  # Idle
        print("✓ Initial state OK\n")

        return True
    except Exception as e:
        print(f"✗ Initialization failed: {e}\n")
        return False

if __name__ == '__main__':
    if test_imports():
        test_initialization()
        print("✅ ZWO camera module PASSED (mock)\n")
```

2.4 Test ZWO Camera with Hardware

Test Script: test_camera_zwo_hardware.py

python

```python
#!/usr/bin/env python3
"""Test ZWO camera with actual hardware"""

from camera_zwo import ZWOCamera, ZWO_AVAILABLE
import time

def test_connection():
    print("Testing camera connection...")

    if not ZWO_AVAILABLE:
        print("✗ ZWO SDK not available")
        return None

    camera = ZWOCamera(camera_id=0)
    success = camera.connect()

    if not success:
        print("✗ Connection failed - check USB")
        return None

    print(f"✓ Connected: {camera.sensor_name}")
    print(f"  Size: {camera.camera_xsize} x {camera.camera_ysize}")
    print(f"  Pixel size: {camera.pixel_size_x} µm")
    print(f"  Sensor type: {camera.sensor_type}")
    print(f"  Max binning: {camera.max_bin_x}")

    return camera

def test_temperature(camera):
    print("\nTesting temperature...")

    camera.update_temperature()
    print(f"  Temperature: {camera.ccd_temperature:.1f}°C")

    if camera.can_set_ccd_temperature:
        print(f"  Cooling: Supported")
        print(f"  Cooler on: {camera.cooler_on}")
        print(f"  Cooler power: {camera.cooler_power}%")
    else:
        print(f"  Cooling: Not supported")

    print("✓ Temperature OK\n")

def test_short_exposure(camera):
    print("Testing short exposure (1 second)...")

    # Set binning for faster test
```

```
    camera.bin_x = 2
    camera.bin_y = 2
    camera.num_x = camera.camera_xsize // 2
    camera.num_y = camera.camera_ysize // 2

    camera.start_exposure(1.0, True)
    print("  Exposure started...")

    # Monitor progress
    while not camera.image_ready:
        print(f"  Progress: {camera.percent_completed}%")
        time.sleep(0.5)

    print("  Exposure complete!")

    # Get image
    img = camera.get_image_array()
    print(f"  Image shape: {img.shape}")
    print(f"  Image dtype: {img.dtype}")
    print(f"  Image min/max: {img.min()} / {img.max()}")

    print("✓ Exposure OK\n")

def test_binning(camera):
    print("Testing binning modes...")

    for bin_mode in [1, 2, 4]:
        if bin_mode <= camera.max_bin_x:
            camera.bin_x = bin_mode
            camera.bin_y = bin_mode
            print(f"  {bin_mode}x{bin_mode} binning: OK")

    # Reset to 1x1
    camera.bin_x = 1
    camera.bin_y = 1

    print("✓ Binning OK\n")

def test_gain_range(camera):
    print("Testing gain range...")

    print(f"  Gain range: {camera.gain_min} - {camera.gain_max}")

    # Test setting gain
    camera.gain = (camera.gain_min + camera.gain_max) // 2
    print(f"  Set gain to: {camera.gain}")

    print("✓ Gain OK\n")

if __name__ == '__main__':
```

```
    print("="*60)
    print("ZWO CAMERA HARDWARE TEST")
    print("="*60)
    print("⚠️  Ensure ZWO camera is connected via USB")
    print("="*60 + "\n")

    camera = test_connection()
    if camera:
        test_temperature(camera)
        test_binning(camera)
        test_gain_range(camera)
        test_short_exposure(camera)

        camera.disconnect()
        print("✅ ZWO camera hardware PASSED\n")
    else:
        print("✗ Tests aborted - no connection\n")
```

Level 3: API Endpoint Testing

3.1 Test Management API

Test Script: test_api_management.py

python

```python
#!/usr/bin/env python3
"""Test Alpaca management API"""

import requests
import json

BASE_URL = "http://localhost:5555"

def test_api_versions():
    print("Testing /management/apiversions...")

    r = requests.get(f"{BASE_URL}/management/apiversions")
    data = r.json()

    assert r.status_code == 200
    assert 'Value' in data
    assert 1 in data['Value']

    print("✓ API versions OK\n")

def test_description():
    print("Testing /management/v1/description...")

    r = requests.get(f"{BASE_URL}/management/v1/description")
    data = r.json()

    assert r.status_code == 200
    assert 'Value' in data
    assert 'ServerName' in data['Value']

    print(f"  Server: {data['Value']['ServerName']}")
    print("✓ Description OK\n")

def test_configured_devices():
    print("Testing /management/v1/configureddevices...")

    r = requests.get(f"{BASE_URL}/management/v1/configureddevices")
    data = r.json()

    assert r.status_code == 200
    assert 'Value' in data
    assert isinstance(data['Value'], list)

    print(f"  Devices found: {len(data['Value'])}")
    for device in data['Value']:
        print(f"    - {device['DeviceType']} (device {device['DeviceNumber']}): {device['DeviceName']}")
```

```
    print("✓ Configured devices OK\n")

if __name__ == '__main__':
    print("="*60)
    print("MANAGEMENT API TEST")
    print("="*60)
    print("⚠️  Server must be running: python3 main.py")
    print("="*60 + "\n")

    try:
        test_api_versions()
        test_description()
        test_configured_devices()
        print("✅ Management API PASSED\n")
    except requests.exceptions.ConnectionError:
        print("✗ Cannot connect to server")
        print("  Start server: python3 main.py\n")
    except Exception as e:
        print(f"✗ Test failed: {e}\n")
```

Run:

```
bash

# Terminal 1: Start server
python3 main.py

# Terminal 2: Run tests
python3 test_api_management.py
```

3.2 Test Telescope API Endpoints

Test Script: `test_api_telescope.py`

python

```python
#!/usr/bin/env python3
"""Test telescope Alpaca API endpoints"""

import requests
import time

BASE_URL = "http://localhost:5555"
DEVICE_NUM = 0

def test_connection():
    print("Testing telescope connection...")

    # Get initial state
    r = requests.get(f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/connected")
    data = r.json()
    initial_state = data['Value']
    print(f"  Initial state: {'Connected' if initial_state else 'Disconnected'}")

    # Connect
    if not initial_state:
        print("  Connecting...")
        r = requests.put(
            f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/connected",
            data={'Connected': 'true'}
        )
        data = r.json()
        assert data['ErrorNumber'] == 0
        time.sleep(2)

    # Verify connected
    r = requests.get(f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/connected")
    data = r.json()
    assert data['Value'] == True

    print("✓ Connection OK\n")
    return True

def test_position():
    print("Testing position reading...")

    # RA
    r = requests.get(f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/rightascension")
    data = r.json()
    assert 'Value' in data
    print(f"  RA: {data['Value']:.4f} hours")

    # Dec
```

```
    r = requests.get(f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/declination")
    data = r.json()
    assert 'Value' in data
    print(f"  Dec: {data['Value']:.4f} degrees")

    # Alt
    r = requests.get(f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/altitude")
    data = r.json()
    print(f"  Alt: {data['Value']:.4f} degrees")

    # Az
    r = requests.get(f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/azimuth")
    data = r.json()
    print(f"  Az: {data['Value']:.4f} degrees")

    print("✓ Position OK\n")

def test_tracking():
    print("Testing tracking...")

    # Get current state
    r = requests.get(f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/tracking")
    data = r.json()
    initial_tracking = data['Value']
    print(f"  Initial tracking: {initial_tracking}")

    # Toggle
    r = requests.put(
        f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/tracking",
        data={'Tracking': 'false' if initial_tracking else 'true'}
    )
    assert r.json()['ErrorNumber'] == 0

    # Restore
    r = requests.put(
        f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/tracking",
        data={'Tracking': 'true' if initial_tracking else 'false'}
    )

    print("✓ Tracking OK\n")

def test_capabilities():
    print("Testing capabilities...")

    caps = [
        'canfindhome',
        'canpark',
        'canpulseguide',
        'canslew',
        'cansync'
```

```
    ]

    for cap in caps:
        r = requests.get(f"{BASE_URL}/api/v1/telescope/{DEVICE_NUM}/{cap}")
        data = r.json()
        print(f"  {cap}: {data['Value']}")

    print("✓ Capabilities OK\n")

if __name__ == '__main__':
    print("="*60)
    print("TELESCOPE API TEST")
    print("="*60)
    print("⚠️  Server running + OnStepX connected")
    print("="*60 + "\n")

    try:
        if test_connection():
            test_position()
            test_tracking()
            test_capabilities()
            print("✅ Telescope API PASSED\n")
    except Exception as e:
        print(f"✗ Test failed: {e}\n")
```

3.3 Test Camera API Endpoints

Test Script: `test_api_camera.py`

python

```python
#!/usr/bin/env python3
"""Test camera Alpaca API endpoints"""

import requests
import time
import base64
import numpy as np

BASE_URL = "http://localhost:5555"
DEVICE_NUM = 0  # ZWO camera

def test_connection():
    print("Testing camera connection...")

    # Connect
    r = requests.put(
        f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/connected",
        data={'Connected': 'true'}
    )
    assert r.json()['ErrorNumber'] == 0
    time.sleep(2)

    # Verify
    r = requests.get(f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/connected")
    assert r.json()['Value'] == True

    print("✓ Connection OK\n")

def test_camera_info():
    print("Testing camera info...")

    # Name
    r = requests.get(f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/name")
    print(f"  Name: {r.json()['Value']}")

    # Size
    r = requests.get(f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/cameraxsize")
    xsize = r.json()['Value']

    r = requests.get(f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/cameraysize")
    ysize = r.json()['Value']

    print(f"  Size: {xsize} x {ysize}")

    print("✓ Camera info OK\n")

def test_exposure():
```

```
    print("Testing exposure (2 seconds)...")

    # Set binning for faster test
    requests.put(f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/binx", data={'BinX': 2})
    requests.put(f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/biny", data={'BinY': 2})

    # Start exposure
    r = requests.put(
        f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/startexposure",
        data={'Duration': 2.0, 'Light': 'true'}
    )
    assert r.json()['ErrorNumber'] == 0
    print("  Exposure started")

    # Monitor progress
    while True:
        r = requests.get(f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/imageready")
        if r.json()['Value']:
            break

        r = requests.get(f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/percentcompleted")
        print(f"  Progress: {r.json()['Value']}%")
        time.sleep(0.5)

    print("  Image ready!")

    # Download image (Base64)
    r = requests.get(f"{BASE_URL}/api/v1/camera/{DEVICE_NUM}/imagearrayvariant")
    data = r.json()['Value']

    # Decode
    img_bytes = base64.b64decode(data['Data'])
    img = np.frombuffer(img_bytes, dtype=np.uint16).reshape((data['Height'], data['Width']))

    print(f"  Downloaded: {img.shape}, min={img.min()}, max={img.max()}")
    print("✓ Exposure OK\n")

if __name__ == '__main__':
    print("="*60)
    print("CAMERA API TEST")
    print("="*60)
    print("⚠️  Server running + ZWO camera connected")
    print("="*60 + "\n")

    try:
        test_connection()
        test_camera_info()
        test_exposure()
        print("✅ Camera API PASSED\n")
    except Exception as e:
```

```
    except Exception as e:
        print(f"✗ Test failed: {e}\n")
```

Level 4: Integration Testing

4.1 Full System Test

Test Script: test_integration.py

python

```python
#!/usr/bin/env python3
"""Full system integration test"""

import requests
import time

BASE_URL = "http://localhost:5555"

def test_all_devices():
    print("Testing all configured devices...")

    r = requests.get(f"{BASE_URL}/management/v1/configureddevices")
    devices = r.json()['Value']

    for device in devices:
        print(f"\n Testing {device['DeviceType']} {device['DeviceNumber']}...")

        device_type = device['DeviceType'].lower()
        device_num = device['DeviceNumber']

        # Connect
        r = requests.put(
            f"{BASE_URL}/api/v1/{device_type}/{device_num}/connected",
            data={'Connected': 'true'}
        )

        if r.json()['ErrorNumber'] == 0:
            print(f"  ✓ Connected")
        else:
            print(f"  ✗ Connection failed: {r.json()['ErrorMessage']}")

def test_coordinated_operation():
    """Test telescope + camera working together"""
    print("\nTesting coordinated operation...")

    # This would test scenarios like:
    # 1. Slew telescope to target
    # 2. Wait for slew complete
    # 3. Take image
    # 4. Download image

    print("  (Placeholder for coordinated tests)")

if __name__ == '__main__':
    test_all_devices()
    test_coordinated_operation()
```

Level 5: Client Application Testing

5.1 Test with N.I.N.A.

Steps:

1. Open N.I.N.A.
2. Go to Equipment → Telescope
3. Click "Choose" → Select "ASCOM Telescope"
4. In ASCOM Chooser: "Alpaca" → "Discover"
5. Select your Raspberry Pi
6. Choose Telescope device 0
7. Click "Connect"

Verify:

- ✓ Coordinates display correctly
- ✓ Tracking toggle works
- ✓ Slew commands work
- ✓ Park/Unpark works

5.2 Test with PHD2

Steps:

1. Open PHD2
2. Connect Equipment
3. Select "ASCOM Camera" for camera
4. Select "ASCOM Telescope" for mount
5. Choose Alpaca devices

Verify:

- ✓ Camera takes exposures
- ✓ Star detection works
- ✓ Guide pulses sent to mount

Common Issues & Solutions

Issue: "Module not found"

Solution:

```bash
cd ~/onstepx-alpaca
source venv/bin/activate
pip install -r requirements.txt
```

Issue: "Permission denied" on serial port

Solution:

```bash
sudo usermod -a -G dialout $USER
sudo usermod -a -G plugdev $USER
# Log out and back in
```

Issue: Camera not detected

Solution:

```bash
# Check USB
lsusb | grep -i zwo

# Check library
ls -l /usr/local/lib/libASICamera2.so

# Test directly
python3 -c "import zwoasi as asi; asi.init('/usr/local/lib/libASICamera2.so'); print(asi.get_num_cameras())"
```

Issue: Slow image download

Solution: Use `imagearrayvariant` instead of `imagearray`:

```python
# Slow (JSON)
r = requests.get(f"{BASE_URL}/api/v1/camera/0/imagearray")

# Fast (Base64)
r = requests.get(f"{BASE_URL}/api/v1/camera/0/imagearrayvariant")
```

Performance Testing

Test Image Download Speed

Script: `test_performance.py`

python

```python
#!/usr/bin/env python3
import requests
import time

BASE_URL = "http://localhost:5555"

# Take test exposure
requests.put(f"{BASE_URL}/api/v1/camera/0/startexposure",
             data={'Duration': 1.0, 'Light': 'true'})

# Wait for ready
while True:
    r = requests.get(f"{BASE_URL}/api/v1/camera/0/imageready")
    if r.json()['Value']:
        break
    time.sleep(0.1)

# Time download
start = time.time()
r = requests.get(f"{BASE_URL}/api/v1/camera/0/imagearrayvariant")
elapsed = time.time() - start

data = r.json()['Value']
size_mb = len(data['Data']) / 1024 / 1024

print(f"Download time: {elapsed:.2f} seconds")
print(f"Data size: {size_mb:.2f} MB")
print(f"Speed: {size_mb/elapsed:.2f} MB/s")
```

Automated Test Suite

Create: `run_all_tests.sh`

bash

```
#!/bin/bash

echo "================================================"
echo "OnStepX Alpaca Bridge - Complete Test Suite"
echo "================================================"

cd ~/onstepx-alpaca
source venv/bin/activate

echo ""
echo "=== Level 1: Module Tests ==="
python3 test_config.py
python3 test_helpers.py

echo ""
echo "=== Level 2: Device Tests (Mock) ==="
python3 test_telescope_mock.py
python3 test_camera_zwo_mock.py

echo ""
echo "=== Level 3: API Tests ==="
echo "Starting server in background..."
python3 main.py &
SERVER_PID=$!
sleep 5

python3 test_api_management.py
python3 test_api_telescope.py
python3 test_api_camera.py

echo ""
echo "Stopping server..."
kill $SERVER_PID

echo ""
echo "================================================"
echo "Test Suite Complete"
echo "================================================"
```

Make executable and run:

bash

```
chmod +x run_all_tests.sh
./run_all_tests.sh
```

Summary

✅ **Test incrementally** - Don't skip levels ✅ **Mock first** - Test without hardware when possible ✅ **One feature at a time** - Isolate failures ✅ **Automate** - Create test scripts for regression testing ✅ **Document issues** - Keep notes on problems and solutions