024/07 @juhan

Test Plan Template: Space Launch System (SLS)

NASA Document Number: 00004TEST

Test Plan Title: Guidance, Navigation, and Control Systems (**GN&C**) **Test Plan for Space Launch System (SLS)**

System Under Test: Space Launch System (SLS) Guidance, Navigation, and Control Systems (GN&C)

Test Objectives:

- Verify that the SLS rocket meets all safety and performance requirements
- Validate the rocket's ability to carry out a successful launch and deployment of the payload
- Identify and fix any defects or issues in the rocket's systems and subsystems

Test Scope:

- The SLS rocket's entire system, including:
    - Core stage
    - Booster stages
    - Upper stage
    - Payload fairing
    - Guidance, navigation, and control systems
    - Propulsion systems
    - Electrical power systems
    - Communication systems
    - Thermal protection systems
    - Structural systems

Test Approach:

- A combination of simulation-based testing, ground testing, and flight testing will be used to validate the SLS rocket's performance
- Testing will be conducted at various levels, including:
    - Component level
    - Subsystem level
    - System level
    - Integrated system level

Test Suite: SLS Rocket Launch Sequence

Test Case 1: Pre-Launch Sequence

- Verify that all systems are nominal and ready for launch
- Check that the rocket's thrust level is at 0%
- Confirm that the altitude is at 0 meters

Test Case 2: Launch Sequence Initiation

- Simulate the launch sequence initiation command
- Verify that the rocket's thrust level increases to 100%
- Check that the altitude starts increasing

Test Case 3: Max-Q

- Simulate the maximum dynamic pressure (Max-Q) phase
- Verify that the rocket's structural integrity is maintained
- Check that the altitude continues to increase

Test Case 4: Main Engine Cutoff (MECO)

- Simulate the MECO event
- Verify that the main engines shut down
- Check that the altitude reaches the expected value

Test Case 5: Stage Separation

- Simulate the stage separation event
- Verify that the booster stage separates from the core stage
- Check that the core stage continues to ascend

Test Case 6: Fairing Jettison

- Simulate the fairing jettison event
- Verify that the fairing is jettisoned
- Check that the payload is exposed

Test Case 7: Orbit Insertion

- Simulate the orbit insertion maneuver
- Verify that the rocket reaches the desired orbit
- Check that the payload is deployed

Test Case 8: Post-Launch Sequence

- Simulate the post-launch sequence
- Verify that all systems are nominal and ready for payload deployment
- Check that the rocket's thrust level is at 0%

Test Suite: SLS Rocket Launch Sequence Test Automation

Test Objectives:

- Verify that the SLS Rocket Launch Sequence executes correctly
- Identify any defects or issues in the launch sequence

Test Scope:

- The SLS Rocket Launch Sequence, including:
    - Pre-launch checks
    - Liftoff and ascent
    - Max-Q and Mach 1
    - Main engine cutoff and stage separation
    - Fairing deployment and payload release
    - Orbit and deployment of secondary payloads

Test Approach:

- Use a combination of simulation-based testing and mock-based testing to verify the launch sequence
- Implement a modular and scalable test architecture to ensure easy maintenance and updates

Test Cases:

Pre-Launch Checks

- Test Case 1: System Checks
    - Verify that all systems are online and functioning correctly
    - Verify that all subsystems are initialized and ready for launch
- Test Case 2: Weather Checks
    - Verify that weather conditions are within acceptable limits
    - Verify that wind, temperature, and humidity are within acceptable ranges

Liftoff and Ascent

- Test Case 1: Liftoff
    - Verify that the rocket lifts off the launchpad correctly
    - Verify that the guidance system initializes correctly
- Test Case 2: Ascent

  - Verify that the rocket ascends correctly
  - Verify that the propulsion system generates the correct thrust

Max-Q and Mach 1

- Test Case 1: Max-Q
  - Verify that the rocket reaches maximum dynamic pressure (Max-Q) correctly
  - Verify that the structural system withstands the maximum stress
- Test Case 2: Mach 1
  - Verify that the rocket reaches Mach 1 correctly
  - Verify that the propulsion system adjusts correctly for supersonic flight

Main Engine Cutoff and Stage Separation

- Test Case 1: Main Engine Cutoff
  - Verify that the main engine cuts off correctly
  - Verify that the propulsion system transitions to the next stage
- Test Case 2: Stage Separation
  - Verify that the stages separate correctly
  - Verify that the next stage initializes correctly

Fairing Deployment and Payload Release

- Test Case 1: Fairing Deployment
  - Verify that the fairing deploys correctly
  - Verify that the payload is exposed correctly
- Test Case 2: Payload Release
  - Verify that the payload releases correctly
  - Verify that the payload separates correctly from the rocket

Orbit and Deployment of Secondary Payloads

- Test Case 1: Orbit
    - Verify that the rocket reaches orbit correctly
    - Verify that the payload is in the correct orbit
- Test Case 2: Secondary Payload Deployment
    - Verify that the secondary payloads deploy correctly
    - Verify that the secondary payloads separate correctly from the rocket

Test Automation Framework:

- Use a combination of Node.js and Jest as the test automation framework
- Implement a modular and scalable test architecture to ensure easy maintenance and updates

```
// Import required libraries
const { test, expect } = require('@jest/globals');

// Pre-Launch Checks
test('System checks', () => {
  const system = new System();
  system.initialize();
  expect(system.online).toBe(true);
  expect(system.subsystems_initialized).toBe(true);
});

test('Weather checks', () => {
  const weather = new Weather();
  weather.checkConditions();
  expect(weather.withinLimits).toBe(true);
});

// Liftoff and Ascent
test('Liftoff', () => {
  const rocket = new Rocket();
  rocket.liftoff();
  expect(rocket.guidanceSystem_initialized).toBe(true);
});

test('Ascent', () => {
  const rocket = new Rocket();
  rocket.ascent();
  expect(rocket.propulsionSystem_thrust).toBeCloseTo(expectedThrust, 0.01);
});

// Max-Q and Mach 1
test('Max-Q', () => {
  const rocket = new Rocket();
  rocket.maxQ();
  expect(rocket.structuralSystem_stress).toBeLessThan(maxStress);
});

test('Mach 1', () => {
  const rocket = new Rocket();
  rocket.mach1();
  expect(rocket.propulsionSystem_supersonic).toBe(true);
});

// Main Engine Cutoff and Stage Separation
test('Main engine cutoff', () => {
  const rocket = new Rocket();
  rocket.mainEngineCutoff();
  expect(rocket.propulsionSystem_nextStage).toBe(true);
});

test('Stage separation', () => {
  const rocket = new Rocket();
  rocket.stageSeparation();
  expect(rocket.nextStage_initialized).toBe(true);
});

// Fairing Deployment and Payload Release
test('Fairing deployment',
```

Test Plan: Guidance, Navigation, and Control Systems

Module: SLS Rocket Guidance, Navigation, and Control Systems

Objective: To verify that the Guidance, Navigation, and Control Systems of the SLS rocket meet the requirements and specifications outlined in the SLS System Requirements Document (SRD) and the Guidance, Navigation, and Control Systems Requirements Document (GNC RD).

Test Scope:

- Guidance System:
    - Navigation algorithms
    - State estimation
    - Trajectory planning
    - Autopilot system
- Navigation System:
    - Inertial Measurement Unit (IMU)
    - Global Positioning System (GPS)
    - Star tracker
    - Navigation computer
- Control System:
    - Thrust vector control
    - Attitude control
    - Navigation and control software

Test Approach:

- Unit testing: Individual components and subsystems will be tested in isolation to verify their functionality and performance.
- Integration testing: Components and subsystems will be integrated and tested together to verify their interactions and overall system performance.
- System testing: The entire Guidance, Navigation, and Control Systems module will be tested in a simulated environment to verify its performance and functionality.
- Environmental testing: The Guidance, Navigation, and Control Systems module will be tested in a variety of environmental conditions (e.g. temperature, vibration, radiation) to verify its performance and functionality.

Test Cases:

Guidance System:

1. **Test Case: Navigation Algorithm**
    - Preconditions: Navigation algorithm is initialized with valid data
    - Steps:
        1. Input valid navigation data
        2. Verify navigation algorithm output is correct
    - Expected Result: Navigation algorithm output is correct
2. **Test Case: State Estimation**
    - Preconditions: State estimation algorithm is initialized with valid data
    - Steps:
        1. Input valid state estimation data
        2. Verify state estimation algorithm output is correct
    - Expected Result: State estimation algorithm output is correct
3. **Test Case: Trajectory Planning**
    - Preconditions: Trajectory planning algorithm is initialized with valid data
    - Steps:
        1. Input valid trajectory planning data
        2. Verify trajectory planning algorithm output is correct
    - Expected Result: Trajectory planning algorithm output is correct

Navigation System:

1. **Test Case: IMU Performance**
    - Preconditions: IMU is calibrated and initialized
    - Steps:
        1. Input valid IMU data
        2. Verify IMU output is correct
    - Expected Result: IMU output is correct
2. **Test Case: GPS Performance**
    - Preconditions: GPS system is initialized and receiving valid signals
    - Steps:
        1. Input valid GPS data
        2. Verify GPS output is correct
    - Expected Result: GPS output is correct
3. **Test Case: Star Tracker Performance**
    - Preconditions: Star tracker is calibrated and initialized
    - Steps:

        1. Input valid star tracker data
        2. Verify star tracker output is correct
    - Expected Result: Star tracker output is correct

Control System:

1. **Test Case: Thrust Vector Control**
    - Preconditions: Thrust vector control system is initialized and calibrated
    - Steps:
        1. Input valid thrust vector control commands
        2. Verify thrust vector control system output is correct
    - Expected Result: Thrust vector control system output is correct
2. **Test Case: Attitude Control**
    - Preconditions: Attitude control system is initialized and calibrated
    - Steps:
        1. Input valid attitude control commands
        2. Verify attitude control system output is correct
    - Expected Result: Attitude control system output is correct

Test Scripts:

- Python scripts will be used to automate the testing process
- Scripts will be written to simulate various test scenarios and input data
- Scripts will be used to verify the output of the Guidance, Navigation, and Control Systems module

Test Environment:

- Testing will be conducted in a simulated environment using NASA's Simulation and Modeling Environment (SME)
- The SME will be used to simulate various test scenarios and input data
- The SME will be used to verify the output of the Guidance, Navigation, and Control Systems module

Test Schedule:

- Unit testing: 2 weeks
- Integration testing: 4 weeks
- System testing: 6 weeks
- Environmental testing: 4 weeks

Test Deliverables:

- Test plan document
- Test scripts
- Test data
- Test results report

Test Risks:

- Inadequate testing time
- Insufficient resources
- Complexity of the Guidance, Navigation, and Control Systems module

```
// Import required libraries
const test = require('selenium-webdriver/testing');
const assert = require('assert');

// Test suite for Guidance, Navigation, and Control Systems
test.describe('Guidance, Navigation, and Control Systems', function() {
  // Test case: Navigation Algorithm
  test.it('Navigation Algorithm', function() {
    // Input valid navigation data
    const navigationData = {
      // Mock navigation data
    };
    // Verify navigation algorithm output is correct
    const output = navigationAlgorithm(navigationData);
    assert.strictEqual(output, expectedOutput);
  });

  // Test case: State Estimation
  test.it('State Estimation', function() {
    // Input valid state estimation data
    const stateEstimationData = {
      // Mock state estimation data
    };
    // Verify state estimation algorithm output is correct
    const output = stateEstimationAlgorithm(stateEstimationData);
    assert.strictEqual(output, expectedOutput);
  });

  // Test case: Trajectory Planning
  test.it('Trajectory Planning', function() {
    // Input valid trajectory planning data
    const trajectoryPlanningData = {
      // Mock trajectory planning data
    };
    // Verify trajectory planning algorithm output is correct
    const output = trajectoryPlanningAlgorithm(trajectoryPlanningData);
    assert.strictEqual(output, expectedOutput);
  });

  // Test case: IMU Performance
  test.it('IMU Performance', function() {
    // Input valid IMU data
    const imuData = {
      // Mock IMU data
    };
    // Verify IMU output is correct
    const output = imu(imuData);
    assert.strictEqual(output, expectedOutput);
  });

  // Test case: GPS Performance
  test.it('GPS Performance', function() {
    // Input valid GPS data
    const gpsData = {
      // Mock GPS data
    };
    // Verify GPS output is correct
    const output = gps(gpsData);
    assert.strictEqual(output, expectedOutput);
  });

  // Test case: Star Tracker Performance
  test.it('Star Tracker Performance', function() {
    // Input valid star tracker data
    const starTrackerData = {
      // Mock star tracker data
    };
    // Verify star tracker output is correct
    const output = starTracker(starTrackerData);
    assert.strictEqual(output, expectedOutput);
  });

  // Test case: Thrust Vector Control
  test.it('Thrust Vector Control', function() {
    // Input valid thrust vector control commands
    const thrustVectorControlCommands = {
      // Mock thrust vector control commands
    };
    // Verify thrust vector control system output is correct
    const output = thrustVectorControl(thrustVectorControlCommands);
    assert.strictEqual(output, expectedOutput);
  });

  // Test case: Attitude Control
  test.it('Attitude Control', function() {
    // Input valid attitude control commands
    const attitudeControlCommands = {
      // Mock attitude control commands
    };
    // Verify attitude control system output is correct
    const output = attitudeControl(attitudeControlCommands);
    assert.strictEqual(output, expectedOutput);
  });
});
```

CONCLUSION

This test suite covers the most important test cases for the "Guidance, Navigation, and Control Systems" module of the SLS rocket, including navigation algorithm, state estimation, trajectory planning, IMU performance, GPS performance, star tracker performance, thrust vector control, and attitude control. Note that you will need to replace the mock data and expected output with actual values specific to your system.

Propulsion Systems Test Plan

Test Plan ID: PS-TP-001

Test Plan Name: Propulsion Systems Functional and Performance Testing

Test Plan Objective:

- Verify that the Propulsion Systems meet all functional and performance requirements
- Identify any defects or issues in the Propulsion Systems

Test Scope:

- The Propulsion Systems, including:
    - Main Engines
    - Thrusters
    - Fuel Systems
    - Oxidizer Systems
    - Ignition Systems
    - Thrust Vector Control Systems

Test Approach:

- Use a combination of simulation-based testing and hardware-in-the-loop testing to verify the Propulsion Systems
- Implement a modular and scalable test architecture to ensure easy maintenance and updates

Test Environment:

- Test Stand: [Insert test stand details]
- Test Equipment: [Insert test equipment details]
- Software Tools: [Insert software tools details]

Test Schedule:

- Test Start Date: [Insert start date]
- Test End Date: [Insert end date]
- Test Duration: [Insert duration]

Test Deliverables:

- Test Reports: [Insert report details]
- Test Data: [Insert data details]
- Defect Reports: [Insert defect report details]

Test Cases:

Main Engines

- Test Case 1: Engine Startup
    - Verify that the main engine starts up correctly
    - Verify that the engine reaches the correct operating conditions
- Test Case 2: Engine Performance
    - Verify that the main engine generates the correct thrust

  - Verify that the engine operates within the correct temperature and pressure ranges
- Test Case 3: Engine Shutdown
  - Verify that the main engine shuts down correctly
  - Verify that the engine cools down correctly

Thrusters

- Test Case 1: Thruster Startup
  - Verify that the thruster starts up correctly
  - Verify that the thruster reaches the correct operating conditions
- Test Case 2: Thruster Performance
  - Verify that the thruster generates the correct thrust
  - Verify that the thruster operates within the correct temperature and pressure ranges
- Test Case 3: Thruster Shutdown
  - Verify that the thruster shuts down correctly
  - Verify that the thruster cools down correctly

Fuel Systems

- Test Case 1: Fuel Flow
  - Verify that the fuel flows correctly through the system
  - Verify that the fuel pressure and temperature are within the correct ranges
- Test Case 2: Fuel Level
  - Verify that the fuel level is correctly monitored and reported
  - Verify that the fuel level is within the correct range

Oxidizer Systems

- Test Case 1: Oxidizer Flow
  - Verify that the oxidizer flows correctly through the system
  - Verify that the oxidizer pressure and temperature are within the correct ranges

- Test Case 2: Oxidizer Level
    - Verify that the oxidizer level is correctly monitored and reported
    - Verify that the oxidizer level is within the correct range

Ignition Systems

- Test Case 1: Ignition
    - Verify that the ignition system ignites the fuel and oxidizer correctly
    - Verify that the ignition system operates within the correct temperature and pressure ranges
- Test Case 2: Ignition Timing
    - Verify that the ignition system ignites at the correct time
    - Verify that the ignition system operates within the correct timing range

Thrust Vector Control Systems

- Test Case 1: Thrust Vectoring
    - Verify that the thrust vector control system correctly adjusts the thrust vector
    - Verify that the thrust vector control system operates within the correct range
- Test Case 2: Thrust Vector Stability
    - Verify that the thrust vector control system maintains the correct thrust vector stability
    - Verify that the thrust vector control system operates within the correct stability range

Test Risks and Assumptions:

- Risk of delays in the development of SLS, which could impact the schedule of EM-1 and EM-2 missions.
- Assumption that accelerating the production of SLS and Orion will support up to two launches a year.
- Risk of not meeting the 2024 deadline for returning humans to the lunar surface.
- Assumption that the development of the Block 1B version of SLS with its more powerful Exploration Upper Stage will be necessary to achieve the 2024 deadline.
- Risk of not having a second mobile launch platform designed specifically for the SLS Block 1B.
- Assumption that skipping the "green run" test of the SLS core stage could save several months in the EM-1 schedule, but may not eliminate as much risk as running the full test.
- Risk of not meeting the projected LOC (Loss of Crew) requirement of 1:75 for Orion/SLS, which is similar to the actual LOC figure for STS.
- Assumption that the 45-day study to look at ways to speed up work on the SLS will identify opportunities to accelerate the development process.

This test plan provides a comprehensive outline of the testing activities, test cases, and test environment required to verify the Propulsion Systems. It also identifies the test risks and assumptions, resources, and schedule milestones.

TEST AUTOMATION SUITE

- Use a combination of Python and Pytest as the test automation framework
- Implement a modular and scalable test architecture to ensure easy maintenance and updates

PIP

```
import pytest

# Main Engine Test
@pytest.mark.main_engine
def test_main_engine_startup():
    engine = MainEngine()
    engine.startup()
    assert engine.operating_conditions == "normal"

@pytest.mark.main_engine
def test_main_engine_performance():
    engine = MainEngine()
    engine.run()
    assert engine.thrust == 100000  # expected thrust value
    assert engine.temperature == 500  # expected temperature value
    assert engine.pressure == 100  # expected pressure value

@pytest.mark.main_engine
def test_main_engine_shutdown():
    engine = MainEngine()
    engine.shutdown()
    assert engine.operating_conditions == "shutdown"

# Thruster Test
@pytest.mark.thruster
def test_thruster_startup():
    thruster = Thruster()
    thruster.startup()
    assert thruster.operating_conditions == "normal"

@pytest.mark.thruster
def test_thruster_performance():
    thruster = Thruster()
    thruster.run()
    assert thruster.thrust == 1000  # expected thrust value
    assert thruster.temperature == 200  # expected temperature value
    assert thruster.pressure == 50  # expected pressure value

@pytest.mark.thruster
def test_thruster_shutdown():
    thruster = Thruster()
    thr
```

NODEJS

```
// Import required libraries
const { test, expect } = require('@jest/globals');
```

```
// Simplified Pulse Detonation Engine Performance Test
test('Pulse Detonation Engine Performance', () => {
  const engine = new PulseDetonationEngine();
  engine.startup();
  expect(engine.flowRate).toBeCloseTo(100, 0.01); // expected flow rate
value
  expect(engine.specificThrust).toBeCloseTo(500, 0.01); // expected
specific thrust value
  expect(engine.specificImpulse).toBeCloseTo(200, 0.01); // expected
specific impulse value
});

// Quasi-One-Dimensional Reactive Code for Design and Analysis of
Gasdynamic-Based Propulsion Systems Test
test('Quasi-One-Dimensional Reactive Code', () => {
  const code = new QuasiOneDimensionalReactiveCode();
  code.run();
  expect(code.performanceEvaluation).toBeCloseTo(0.8, 0.01); // expected
performance evaluation value
});

// Sectored-One-Dimensional Combustor Code Test
test('Sectored-One-Dimensional Combustor Code', () => {
  const code = new SectoredOneDimensionalCombustorCode();
  code.run();
  expect(code.thermoAcousticInstability).toBeCloseTo(0.5, 0.01); //
expected thermo-acoustic instability value
});

// Main Engine Test
test('Main Engine', () => {
  const engine = new MainEngine();
  engine.startup();
  expect(engine.thrust).toBeCloseTo(100000, 0.01); // expected thrust value
  expect(engine.temperature).toBeCloseTo(500, 0.01); // expected
temperature value
  expect(engine.pressure).toBeCloseTo(100, 0.01); // expected pressure
value
});

// Thruster Test
test('Thruster', () => {
  const thruster = new Thruster();
  thruster.startup();
  expect(thruster.thrust).toBeCloseTo(1000, 0.01); // expected thrust value
  expect(thruster.temperature).toBeCloseTo(200, 0.01); // expected
temperature value
  expect(thruster.pressure).toBeCloseTo(50, 0.01); // expected pressure
value
});
```

CONCLUSION

The Propulsion Systems Test Plan for the Space Launch System (SLS) outlines a comprehensive approach to verifying the performance, safety, and reliability of the rocket's propulsion systems. Through a series of rigorous tests and evaluations, we will validate the design and functionality of the RS-25 engines, the core stage, and the booster systems, ensuring that they meet the stringent requirements for deep space exploration.

The successful execution of this test plan will provide confidence in the SLS propulsion systems' ability to power the rocket's ascent, orbit, and trans-lunar injection, ultimately enabling the safe and successful transportation of astronauts and cargo to the lunar vicinity and beyond. By identifying and mitigating potential risks and issues, we will ensure that the SLS propulsion systems are ready to support the Artemis program's ambitious goals, including returning humans to the lunar surface by 2024.

The data and insights gathered during these tests will not only inform the development of the SLS, but also contribute to the advancement of rocket propulsion technology, paving the way for future human exploration of the solar system. With a thorough and well-executed test plan, we can ensure that the SLS propulsion systems are robust, efficient, and reliable, ultimately enabling NASA to achieve its vision for a sustainable presence in space.