## Question 1

### Assumptions:

The implementation follows a **black-box integration testing** approach focused on verifying that the Warning status is determined correctly by the system. The testing logic was structured and implemented by the following steps:

1. **Decision Table Analysis + Boundary Values / Equivalent Partitioning**Created a decision table to define "Warning" status conditions, then selected boundary (low and high) and mid-range test data to cover both Condition A and B (based on probability\_of\_collision, miss distance, and time of closest approach).

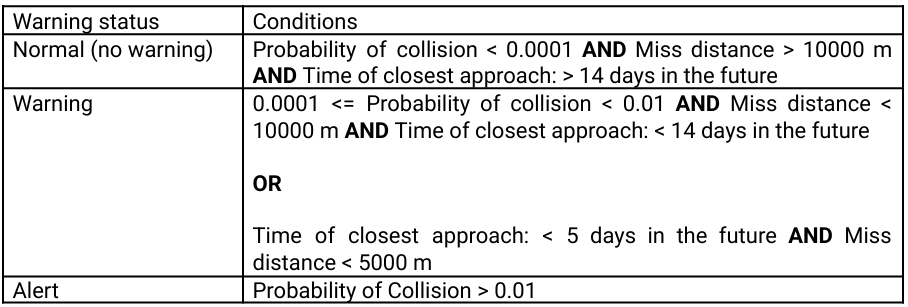
**Condition A:** 0.0001 <= probability\_of\_collision <= 0.01 AND miss\_distance <= 10000 AND TCA <= 14 days  
**Condition B:** probability\_of\_collision < 0.0001 AND miss\_distance < 5000 AND TCA < 5 days

1. **TDD Approach**Used parameterized tests in a TDD style to validate multiple scenarios through a single method with varied inputs.
2. **Scope**Test cases with values slightly outside the "Warning" boundaries were excluded, as they cover "Normal" and "Alert" statuses, which are out of scope.

One negative test was added for invalid input values.

### Doubts:

In the requirement table, I identified some ambiguities. For example:

* When **Probability of Collision = 0.01**, it's unclear whether the status should be **"Warning"** or **"Alert"**.
* Similarly, if **Time of Closest Approach = 14 days** or **Miss Distance = 10,000 meters**, the expected status is not explicitly stated.  
  

For consistency, I intentionally included these boundary values in the **"Warning"** category, but these cases should ideally be clarified with the **Product Owner** before implementing or finalizing test cases.

0.0001 <= probability\_of\_collision <= 0.01 **AND** miss\_distance <= 10000 **AND** TCA <= 14 days

**OR** probability\_of\_collision < 0.0001 **AND** miss\_distance < 5000 **AND** TCA < 5 days

### Answer:

The code is written in C#. Test uses single function PostMessageAsync and verifies 5 different scenarios.

Code example could be found here: <https://github.com/TetianaGP/neuraspace.test.task.git>

[Question 1.cs](https://github.com/TetianaGP/neuraspace.test.task/blob/main/Question%201.cs)

private async Task<JObject> PostMessageAsync(double poc, int missDistance, DateTime tcaUtc)

{

var client = new RestClient(BaseUrl);

var request = new RestRequest("/message", Method.Post);

request.AddJsonBody(new

{

satellite\_id = "abc234sghu54646jgh6mbgy367",

chaser\_object\_id = "abc234hhtyhtyhyth23367123",

probability\_of\_collision = poc,

miss\_distance = missDistance,

expected\_date\_of\_collision = DateTime.UtcNow.AddDays(1).ToString("yyyy-MM-ddTHH:mm:ssZ"),

});

var response = await client.ExecuteAsync(request);

Assert.That(response.IsSuccessful, Is.True);

return JObject.Parse(response.Content);

}

[TestCase(0.0001, 10000, 14, TestName = "LowAndHighBoundaries\_Condition1\_ReturnsWarningStatus")]

[TestCase(0.01, 5000, 5, TestName = "HighBoundaryCondition1\_UpperBoundsCondition2\_ReturnsWarningStatus")]

[TestCase(0.0002, 5000, 4.9, TestName = "MidRangeCondition1\_BoundsCondition2\_ReturnsWarningStatus")]

[TestCase(0.00009, 4999, 4.9, TestName = "HighBoundsCondition2\_ReturnsWarningStatus")]

[TestCase(0.00009, 1000, 1, TestName = "LowValues\_InsideCondition2\_ReturnsWarningStatus")]

public async Task SendRequestWithValidBoundaryValues\_ReturnsWarning\_Test(double poc, int missDistance, double daysUntilTca)

{

var tcaDate = DaysFromNow(daysUntilTca);

var result = await PostMessageAsync(poc, missDistance, tcaDate);

Assert.That(result["status"]?.ToString(), Is.EqualTo("Warning"));

}

[TestCase(-0.01, -1, -1, "Error", false, TestName = "InvalidNegativeInputs")]

public async Task SendRequestWithInvalidValues\_NegativeTest(double poc, int missDistance, double daysUntilTca)

{

try

{

var tcaDate = DaysFromNow(daysUntilTca);

var result = await PostMessageAsync(poc, missDistance, tcaDate);

}

catch (Exception ex)

{

Assert.That(message, Is.Not.EqualTo("OK"), "Message should not be OK for invalid input");

Assert.Fail($"Exception thrown for invalid inputs: {ex.Message}");

}

}

### **Question 2**

### Assumptions 2:

* Test environments are properly configured and stable for all testing activities.
* Manual and automated testing resources are sufficient to cover their respective testing areas.
* A comprehensive test plan or test strategy is already defined and approved by the Test Manager.
* Existing automation frameworks and tools are set up and available for use.

### Answer:

**When and how would you do manual tests?**

* QA should be involved as early as possible, participating in User Story refinement starting from Sprint 1 — even before development begins
* During each sprint, manual testing is performed alongside user story development
* Additionally, manual regression testing is done for user stories or features that are not yet automate
* For each User Story, QA defines acceptance criteria, identifies risks, and creates test case scenarios or checklists.

**What types of tests would those be?**

* Exploratory Testing: for quick and early feedback on the developed User Story
* Accessibility, Usability, Localization, and Internationalization Testing: to validate layout, interaction, visual clarity, cultural correctness, ease of use, etc.
* User Acceptance Testing (UAT): against acceptance criteria to verify if the feature meets user needs
* Functional and Regression Testing: only for user stories not covered by automation.

**When and how would you do QA Automation?**

* Begin automation from Sprint 1, after manual verification of key features is completedD
* During the sprint, start by automating high-priority and repeatable test scenarios, then gradually add other functional test cases related to the developed user stories
* QA follows the Testing Pyramid principle: focus on robust API/integration tests, then add UI tests that cannot be covered by API tests
* Build and maintain a Smoke Test Suite that runs after every commit or deployment
* Integrate automated tests into the CI pipeline to run on a regular basis (e.g., daily, on every merge, or as needed) according to project requirements
* After deployment, integrate relevant tests into the Regression Suite.

Assumptions 3:

* Documentation or feature description is sufficient for testing
* Test data are available and up to date
* Test environments (Dev, QA, Pre-prod, Prod) are stable
* The team resources are sufficient and test team has knowledge of the selected tech stack (e.g., C#, API, Playwright, BDD, etc.)
* A CI/CD pipeline is already set up and can be extended for automated testing.
* The project has the necessary budget and infrastructure for tools like Azure DevOps, if selected.

#### Doubts3 :

* the mobile application is a responsive web app or a native/hybrid app, which affects the choice of testing tool (Playwright vs. Appium)?
* not fully clear how to measure full end-to-end test coverage

### Question 3

**Main focus would be on API testing**, as it serves as the backbone for all architectural components (web, mobile, and service). Ensuring API reliability is crucial for overall system integrity.

To ensure all components are tested in an automated and repeatable process, considering independent deployments, I would propose the following tooling strategy:

#### 1. API Testing

**C# with BDD (e.g., Specflow/Reqnroll) + RestSharp + NUnit/xUnit**  
This approach nicely covers both living documentation and provides a clean, code-based framework suitable for testing APIs in their pure form. It integrates well with CI/CD pipelines for automated regression and smoke testing.

**Postman (optional)**Useful for quick manual tests, exploratory testing, and API contract validation. Postman collections can be automated within CI/CD pipelines as well, though maintaining comprehensive documentation might be more challenging.

#### 2. Web and Mobile Web UI Testing

**Playwright**  
 Provides reliable, fast, and cross-browser automation for web applications, including mobile web. It supports parallel execution and integrates seamlessly with CI/CD pipelines, ensuring UI tests run automatically on every deployment.

#### 3. Mobile Application Testing (optional)

**Appium**  
 If the UI of the native mobile application requires testing, Appium enables cross-platform automation and supports both Android and iOS. It can be integrated into CI/CD pipelines, although it demands more setup and maintenance effort.

#### 4. CI/CD Integration (if not setup before)

**GitHub Actions / Azure DevOps**  
 Used to automate test execution and deployment across environments (dev, QA, pre-production, and production). Ensures that no feature proceeds without being fully tested and that test reports are consistently generated for each release.

If the project is large, has the budget for Azure DevOps, and requires rich test reporting with pass/fail charts, trend analysis, logs, screenshots, and historical runs, then Azure DevOps is a better fit.

### 

### Question 4

Written in JSON format collection for Postman:

<https://github.com/TetianaGP/neuraspace.test.task.git>

Question 4 - Celestrak Object Validation.postman\_collection.json

### Question 5

Written in JSON format collection for Postman:

<https://github.com/TetianaGP/neuraspace.test.task.git>

Question 5 - Space Object API Regression Tests.postman\_collection.json

### Doubts:

* Is the validation for the Cospar ID correct? The requirement says it should be the launch year followed by **3 digits**, but the sample data shows a format like 4160-YDP4. This seems to be a four-digit year, a hyphen, and a four-character alphanumeric string, which doesn't match the stated rule.
* In the sample data, one of the objects has a launchDate of 2002-10-11 and a decay date of 1991-07-28. Since the decay date occurs before the launch date, this is a logical inconsistency. Should the API be validating that the decay date is always after the launch date?
* The requirements specify that objectType should only accept the values "RocketBody," "Payload," or "Debris." However, the sample data includes an objectType of "Unknown". Is this a valid value, or is this an error in the response?
* The requirements state that period, inclination, apogee, perigee, dryMass, and launchMass should be **positive numbers**. New object could be created with a value of 0 and even negative values. Should all of these values be strictly greater than zero?
* The requirements state that launchDate and decay dates should be in the format YYYY-MM-DD. However, the sample response uses a more detailed date format, such as 1986-09-17T21:27:11.4824034. This format is also correct and we should be validating against it, right?