TestNG Retry Tests

We retry tests mainly to **handle flaky or intermittent failures**—tests that sometimes fail and sometimes pass, even when there's no change in the code. Here's a breakdown of **why we retry tests**:

✅ 1. Handle Flaky Tests

Tests might fail due to non-deterministic issues such as:

- **Timing issues** (e.g., async calls not completed)
- **Network latency**
- **Browser driver instability** (in UI testing)
- **Dependency on external services**
  Retrying can help distinguish between real bugs and temporary hiccups.

✅ 2. Improve Test Stability

Retries help reduce false negatives, especially in Continuous Integration (CI) pipelines. This keeps the pipeline **green and stable**, making it easier to spot genuine issues.

✅ 3. Avoid Wasting Developer Time

If tests fail randomly, developers may waste time investigating issues that aren't real bugs. Retrying tests can:

- Automatically resolve temporary failures
- Allow focus on **actual problems**

✅ 4. Account for Environmental Variability

Tests that depend on:

- External APIs
- Cloud services
- Database or file I/O
  can behave unpredictably. Retrying can help reduce noise caused by **environmental factors**.

✅ 5. Support Parallel Testing

In parallel execution, resource contention or race conditions may cause occasional failures. Retrying can help filter these out.

⚠️ When Not to Rely on Retries

While retries are useful, **they shouldn't be a crutch**:

- If a test fails consistently, it signals a **real issue**.
- If a test is flaky, it should be **fixed**, not ignored.

💡 Best Practices

- Set a retry limit (e.g., max 2 retries).
- Log retry attempts clearly.
- Tag flaky tests and isolate them for fixing.
- Use tools/frameworks that support retries (e.g., TestNG, PyTest with plugins, JUnit extensions).

Would you like code examples for retrying in tools like **Selenium with TestNG or PyTest**?

Creating a **Retry Analyzer** is a common approach in **TestNG (Java)** to automatically retry failed test cases. It's especially useful when dealing with flaky tests in Selenium automation.

✅ Step-by-Step: Create a Retry Analyzer in TestNG

🔹 Step 1: Create a RetryAnalyzer Class

```
java

import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

public class RetryAnalyzer implements IRetryAnalyzer {
    private int retryCount = 0;
    private static final int maxRetryCount = 2;  // Retry up to 2 times

    @Override
    public boolean retry(ITestResult result) {
        if (retryCount < maxRetryCount) {
            retryCount++;
            System.out.println("Retrying test: " + result.getName() + ", attempt: " +
(retryCount + 1));
            return true;
        }
        return false;
    }
}
```

🔹 Step 2: Associate RetryAnalyzer with Your Tests

You can apply it in two ways:

✅ Option A: Use at Test Method Level

```
java

import org.testng.annotations.Test;
import org.testng.Assert;

public class MyTest {

    @Test(retryAnalyzer = RetryAnalyzer.class)
    public void testMethod() {
        System.out.println("Executing test...");
        Assert.fail("Failing the test intentionally");
    }
}
```

✅ Option B: Apply to All Tests Using a Listener (Optional)

If you want to apply retry logic globally, you can use a **TestNG Listener**:

```java
import org.testng.IAnnotationTransformer;
import org.testng.IRetryAnalyzer;
import org.testng.annotations.ITestAnnotation;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

public class RetryListener implements IAnnotationTransformer {

    @Override
    public void transform(ITestAnnotation annotation, Class testClass,
                          Constructor testConstructor, Method testMethod) {
        IRetryAnalyzer retry = annotation.getRetryAnalyzer();
        if (retry == null) {
            annotation.setRetryAnalyzer(RetryAnalyzer.class);
        }
    }
}
```

✅ Then register the listener in your **`testng.xml`** or use **`@Listeners`**:

```java
@Listeners(RetryListener.class)
public class MyTest {
    @Test
    public void testMethod() {
        System.out.println("Executing test...");
        Assert.fail("Failing the test intentionally");
    }
}
```

🔍 Notes:

- Always limit retries to avoid infinite loops.
- Retry should only be used for **handling known flaky issues**, not to hide real bugs.
- Log retry attempts clearly for debugging.

Would you like a **Maven project structure** or **log reporting integration** (like with ExtentReports or Allure)?

To **attach retry logic to test methods** in **TestNG**, you directly associate your test method with a `RetryAnalyzer` class using the `@Test` annotation.

Here's a step-by-step explanation with a working example.

✅ 1. Create the RetryAnalyzer Class

```
java

import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

public class RetryAnalyzer implements IRetryAnalyzer {

    private int retryCount = 0;
    private static final int maxRetryCount = 2;  // Retry 2 times max

    @Override
    public boolean retry(ITestResult result) {
        if (retryCount < maxRetryCount) {
            retryCount++;
            System.out.println("Retrying test " + result.getName() + " for the " + (retryCount +
1) + " time.");
            return true;  // retry the test
        }
        return false;  // do not retry
    }
}
```

✅ 2. Attach the RetryAnalyzer to a Test Method

You do this with the `@Test(retryAnalyzer = RetryAnalyzer.class)` annotation.

```
java

import org.testng.Assert;
import org.testng.annotations.Test;

public class SampleTest {

    @Test(retryAnalyzer = RetryAnalyzer.class)
    public void testRetryLogic() {
        System.out.println("Executing testRetryLogic");
        Assert.fail("Intentional failure to test retry logic");
    }
}
```

🔁 How It Works:

- The test method `testRetryLogic()` will run.
- If it fails, TestNG will call the `retry()` method of `RetryAnalyzer`.
- If `retry()` returns `true`, TestNG retries the test.
- This will repeat until the test passes or the retry count reaches `maxRetryCount`.

📝 Output Example:

```
bash


Executing testRetryLogic
Retrying test testRetryLogic for the 2 time.
Executing testRetryLogic
Retrying test testRetryLogic for the 3 time.
Executing testRetryLogic
```

🔒 Important Notes:

- You can attach the retry logic to **multiple test methods** by repeating the annotation.
- You can also apply the retry analyzer globally using a **listener** (`IAnnotationTransformer`) if needed.

Would you like help attaching this logic to all test methods globally or integrating with reports like **ExtentReports**?