

**Diagnostics**

*Updated 2/24/2017*

**Why do FTC Teams need a Diagnostics Program?**

Robots that compete in the FIRST FTC program have a lot of moving parts, both literally (motors, wheels, arms etc) and figuratively (sensors, controllers, power levels etc). There are so many things that could go wrong and often do. They tend to go wrong just before a match or worse: during a match. It is difficult for any one person to understand the robot fully and to fix things in the moment under the pressure of the competition. Sometimes every minute or second counts. Team members can use any help they can get to diagnose problems and receive recommendations for fixes.

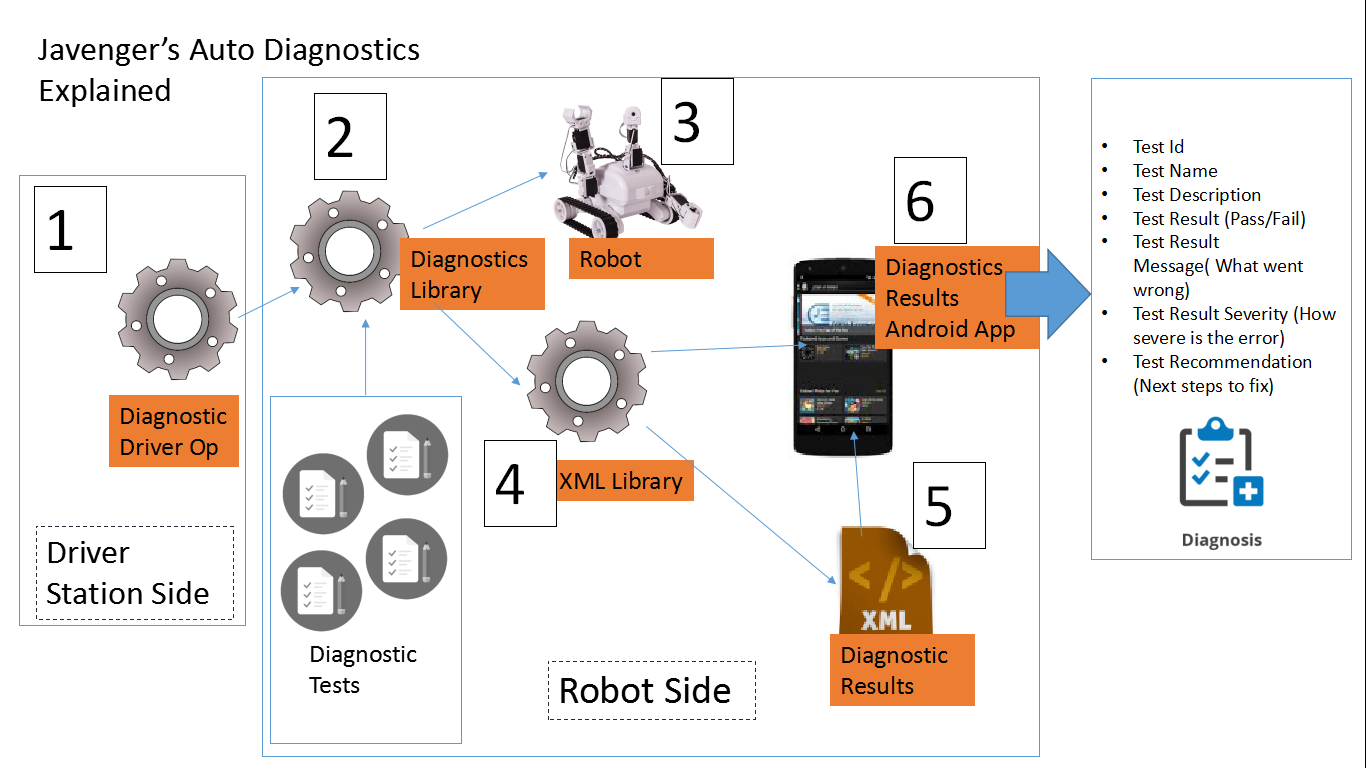
The Javengers diagnostics program can perform various health checks on your robot to determine if its ready to go. It uses automatic and manual checks to check states of the wiring connections, the sensors, and the motors. By finding and correcting these faults before a match, failure of your robot due to problems such as sensor malfunctions or wiring disconnections can be minimized. This will keep your robot safe during the match and help keep your scores high!

**Requirements for a Diagnostics Program**

Like any software, before you can start coding a diagnostics program, you must understand the requirements. Here are the basic requirements of the Javengers diagnostics program:

1. The program must incorporate automatic and manual tests. Examples of automatic tests include moving the motor and testing whether the motor moved using encoders, or turning around to see whether values of the ultrasonic sensor changed, **without human intervention**. An example of a manual test would be having the robot wait until you touch the touch sensor and see whether the touch sensor has sensed a touch.
2. The tests must be **quick** and **easy** to execute. For example, to make the motor testings quick and easy, the movements of the motors should be relatively short so that all mechanisms can be tested in a short amount of time.
3. The program should be **flexible** and **extensible** for new tests. By incorporating a robot class and robot array, it is easy to add new tests. These concepts will be explained later.
4. The program must be **adaptable** and shouldn’t break when tests change. By defining clearly the properties and results of each test, it is easy to change the test procedures by keeping the same outputs.
5. The diagnostics app that displays the results of the tests should be **user friendly**. It should be easy to navigate and view the results of the tests. The manual procedures should also provide detailed instructions so that anyone on the team can run the diagnostics program.
6. Finally, the results of the tests should include **recommendations** for fixing any problems identified.

**The Javengers’ Diagnostic Program Architecture**



1. **Diagnostics Driver Op**

The Diagnostics Driver Op uses methods from the Diagnostics Library to run all of the Diagnostic Tests and to write the Test Results on the XML file. This appears as another Op on the driver station as a Tele Op Program.

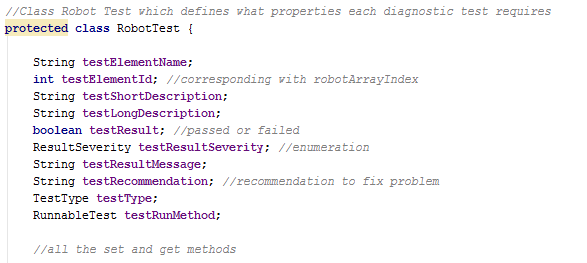
1. **Diagnostics Library**

The Diagnostics Library contains most of the code used to in the Diagnostics Framework.

The first aspect of the diagnostics program is the Robot Test class. This class defines what properties each diagnostic test requires. Examples of the diagnostic test requirements would be a

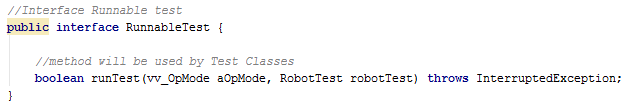
* Test ID
* Test Name
* Test Description (long, short, or both)
* Test Type (automatic or manual)
* Test Result (passed or failed)
* Test Result Severity (low, medium, high, critical)
* Test Recommendation that tells the user the most probable cause of the test failure and the way to fix it
* Diagnostics Test Class which is specific for each diagnostic test.

These conditions will be used to format the XML File that will be read by the Diag Results App. The Robot Test class contains methods to set and get the values previously stated.

Robot Test Class which has the test metadata which will be displayed in the Diagnostics Result App. This class also contains the actual test: testRunMethod

Each Diagnostic Test Class has code to run a certain diagnostic test such as testing the left front motor or the color sensors. These classes implement an interface called RunnableTest. RunnableTest contains a method called *runTest* which returns a boolean and passes an object of the Robot Test Class. The important part of using the interface is that when the method, *runTest* is called, the test procedure when runTest is called is different. This decouples the method from its invocation.

RunnableTest Interface:

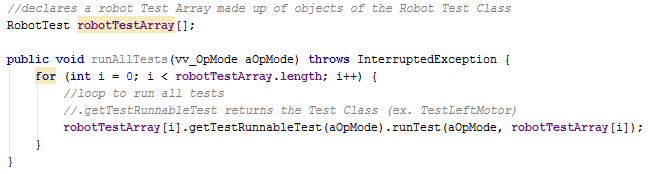


Here is an example of a Diagnostic Test Class which tests the Front Left Wheel Motor:



To run all of these diagnostics test, a robotTestArray made up of Robot Test Class objects must be created. Each index will correspond to the Test ID of the diagnostics test. First, each index of the array, which is an object of the Robot Class, must be instantiated. Then, each index must create a test with one index corresponding to the diagnostics test ID. In order to run the tests, each index must run through the method *runTest*. To avoid the issue of having to add code each time you add a test, a loop is incorporated which runs through all of the indexes of the robotTestArray.

Declaration of the robotTestArray and the *runTest* method:



1. **Robot**

The robot has the elements we want to test such as motors, sensors, lifts, etc. that we want to test. The Diagnostics Library uses methods to test each of the aspects of the robot.

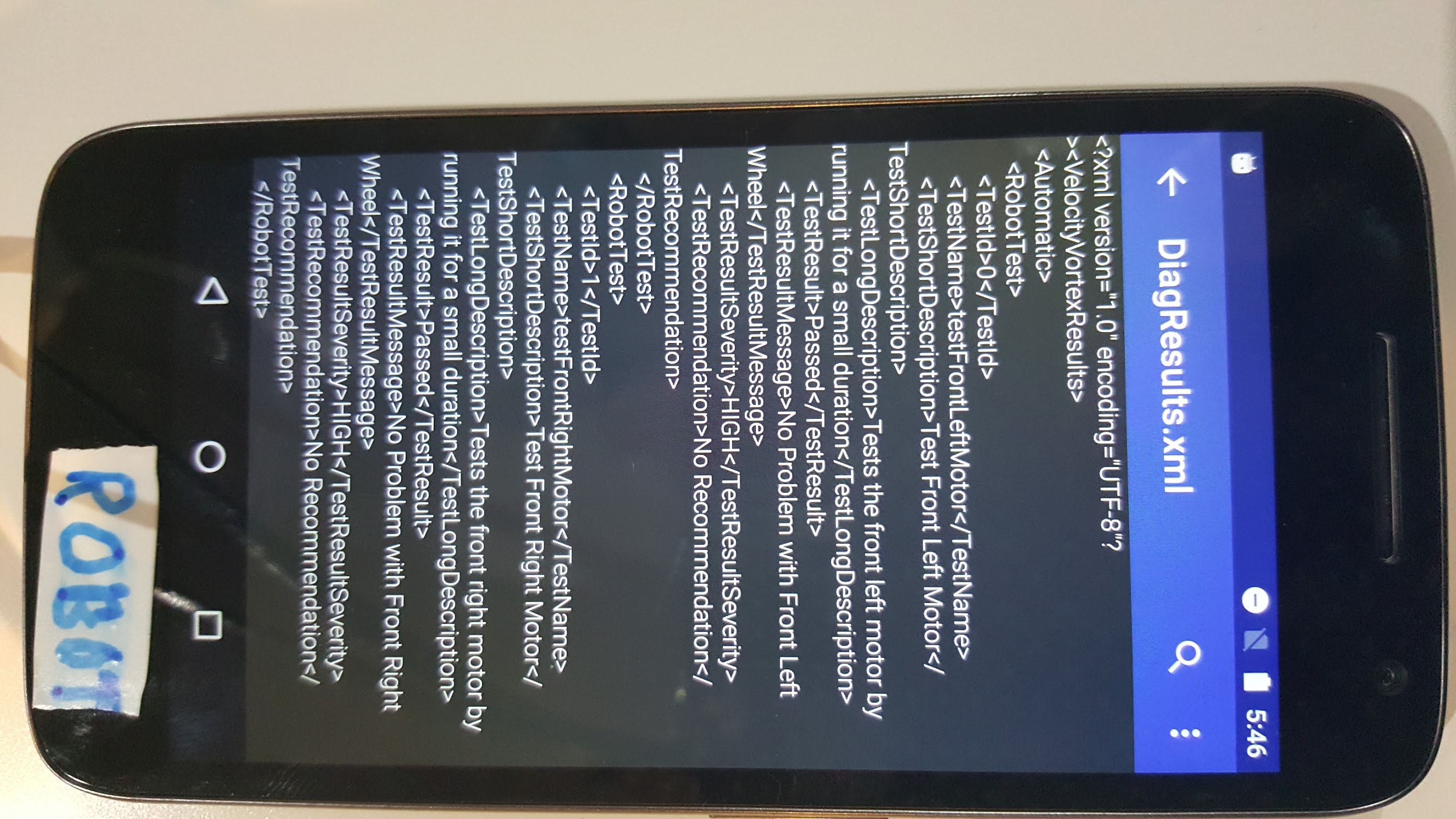
1. **XML Library**

The XML Library contains methods which creates, writes, and read an XML File. These methods are used in the Diagnostics Library inside methods that write and read all of the Diagnostic Tests’ results onto the XML File.

1. **Diagnostic Results XML File**

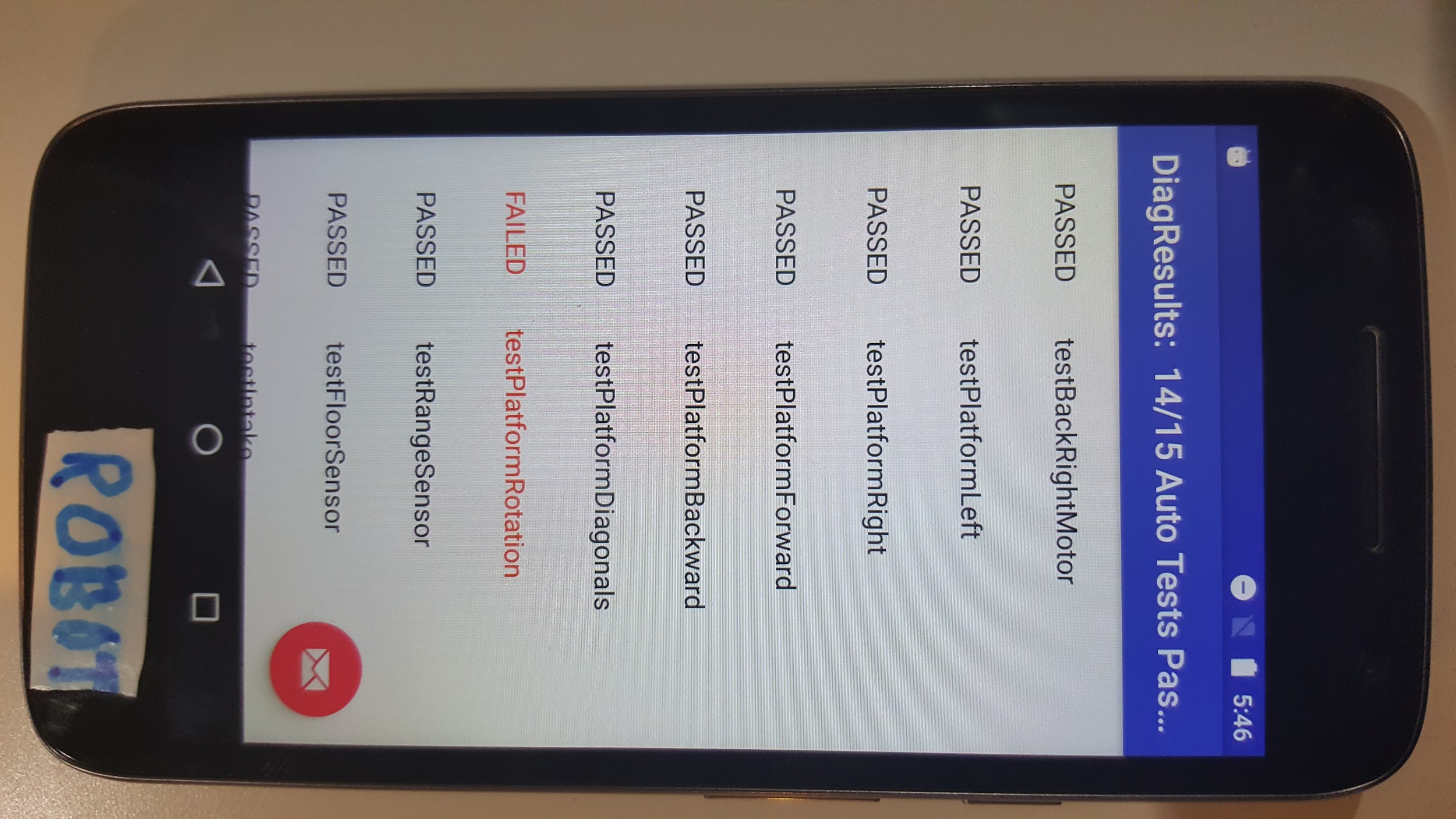
This XML File is written after all of tests have run on the Diagnostic Driver Opp. It contains the Test Results such as the Test Result (passed or fail), the Test Severity, the Test Recommendation, etc.

Here is a screenshot of the DiagResults.xml:

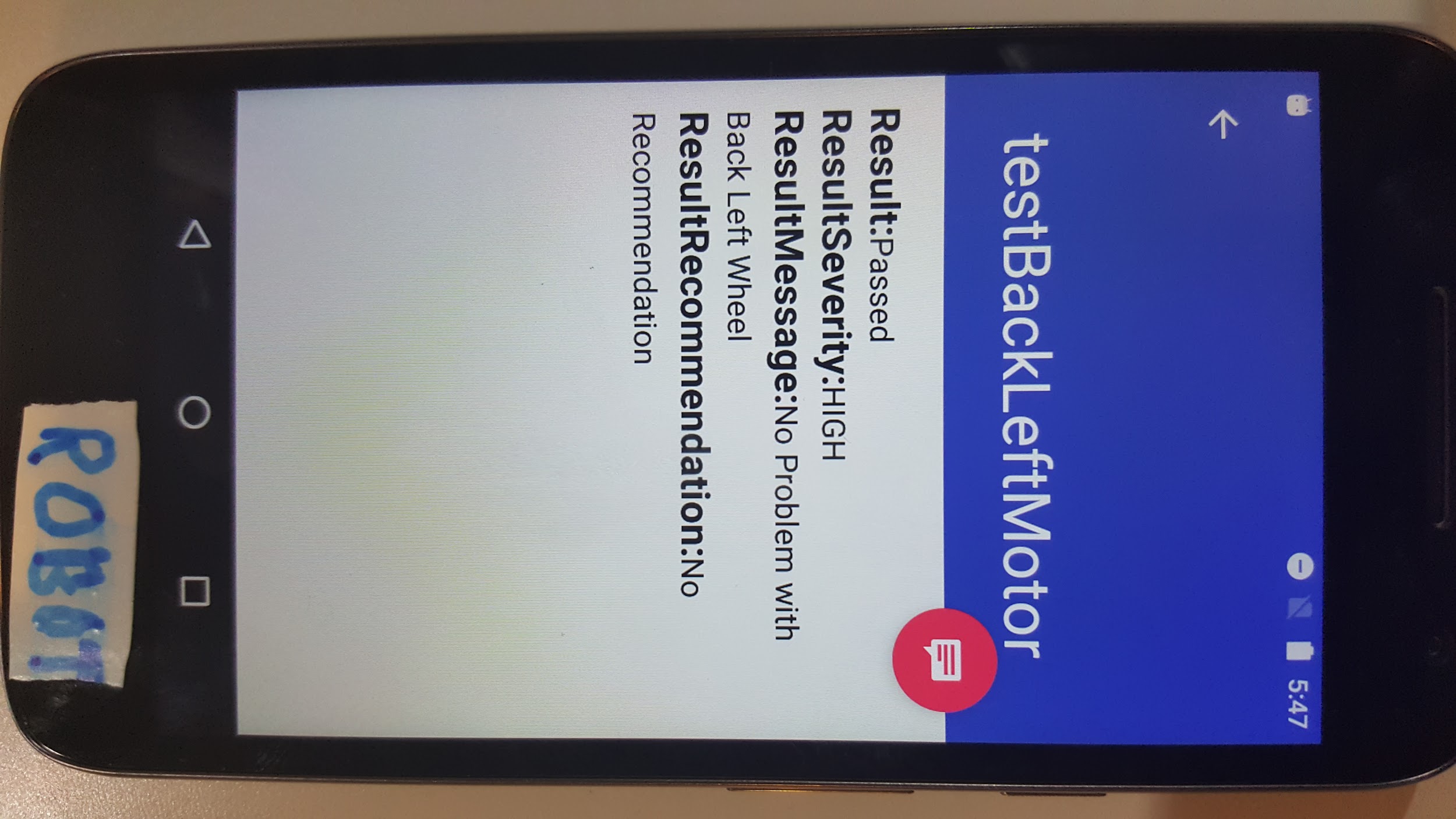
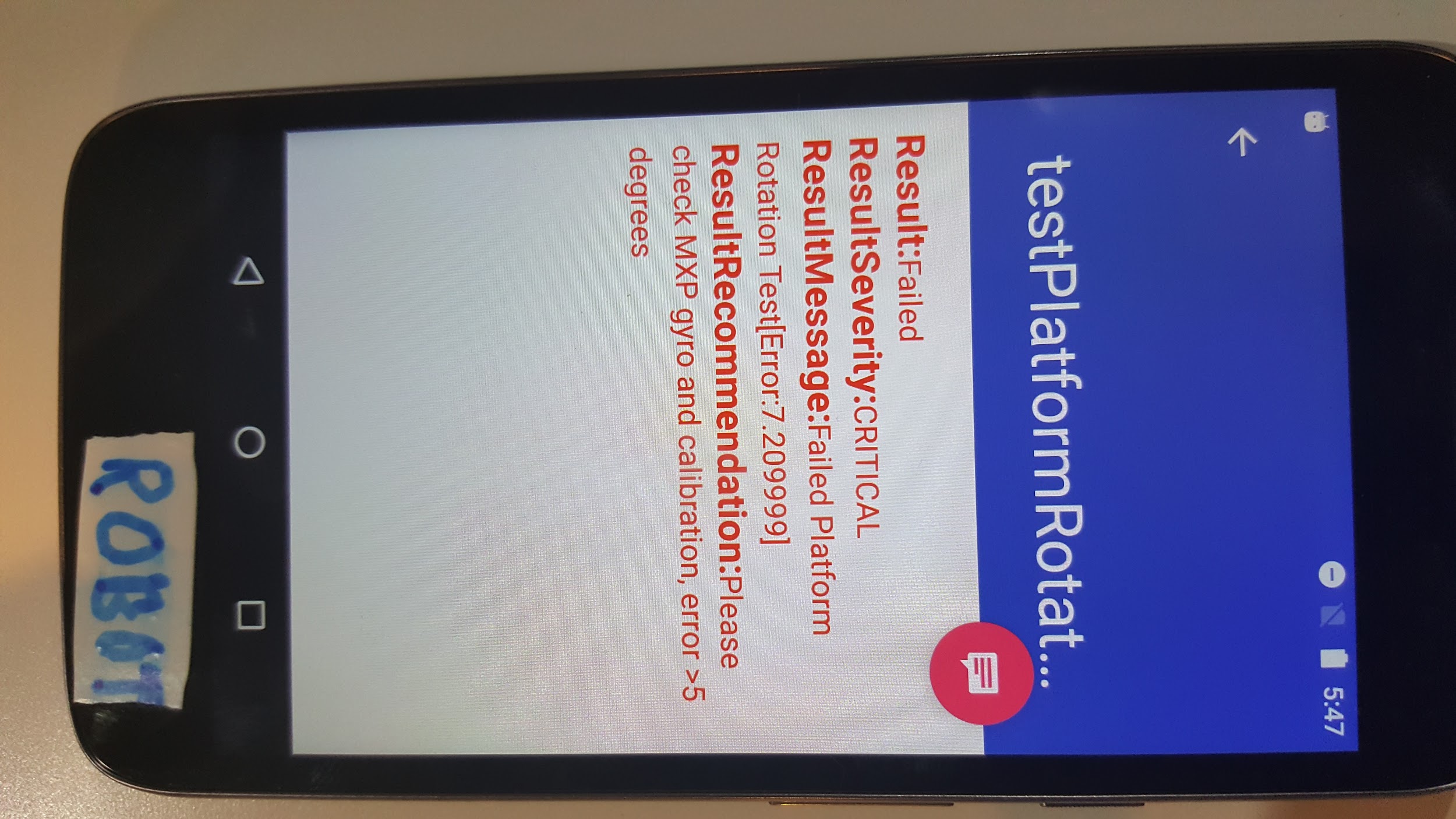


1. **Diagnostics Results Android App**

The Diag Results App opens up this XML file and displays the results using the Master-Detail template. It lists each test and whether it has passed or failed. When you click on the test, more details will pop up displaying the description, the recommendation, and the severity.

This is the list of the types of test and whether it passed or failed:

This is a screenshot of the details of a test (passed and failed):



**Conclusion**

Our team has successfully implemented the diagnostics framework to incorporate 16 automatic tests of the motors and sensors. Our next step forward is to incorporporate more diagnostic tests such as manual tests and create a similar program called Diagnostic Choices which would allows us to choose what tests we would like to run.

We look forward to collaborating with other teams to further develop our Diagnostics Framework[[1]](#footnote-0). If your team has any questions or would like to collaborate with us, please contact us at [admin@pacar-robotics.org](mailto:admin@pacar-robotics.org).

1. We are releasing this software ideas represented here under the MIT Open Source License. [↑](#footnote-ref-0)