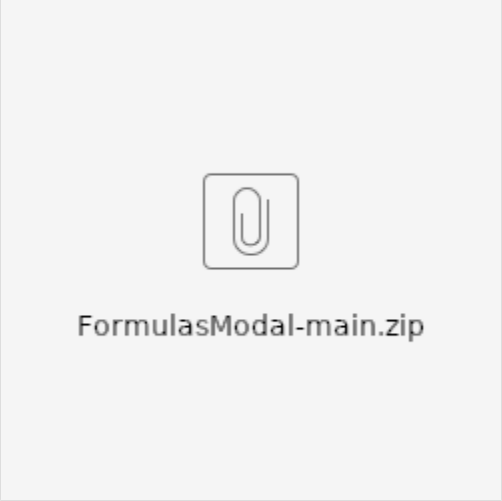
Coding Assignment

Implementation

Source Code

Our source code is in the zip file below.



Quality Review

Process

As a group, we went through the source code files. We looked for camel casing for variables, appropriate comments, tabs for spacing, appropriate variable names, unneeded extra lines or spaces, and code that needs refactoring.

Issues

File	Issues
FormulasModal.pro	• There are unnecessary empty new lines in the code.

databaseconnection.h	• An issue determined from analyzing the code in the file is that none of the variables in the file are in camel case form.
formulas.h	• An issue with this file is that there are needed comments to explain a general summary of how the private methods in the slots work.
main.h	• One issue for this file is that it might not be needed since it might not be important for the application.
unittests.h	• One issue for the file is that it has unused private variables.
VehicleLookUp.h	• One issue for this file is that it needs comments that explain a general summary of how the private methods in the slots work. Another issue with the file is that it contains extra empty new lines between the slots definition.
databaseconnection.cpp	• An issue for this file is that it needs camel casing for the db variables, query string, single row, row value, and the query results in databaseconnection.cpp
formulas.cpp	• Need to remove extra line at the top of the file • Need to add comments for each function to give a general summary of what the function is doing • Need appropriate spaces around operators and operands • Need to add a new line before each comment • Has repeated pointers to line edits throughout the code • Has extra empty lines • Need comments for the float variables for each method • Need camel case for variable in calculateComLong function • Need comments for QLine edits in methods • Need comments and camel case variable for method calculateMinimumSpeed • Need comments and appropriate spaces and new lines for method calculateMinimumSpeed • Add a comment with an explanation for setters and getters • Has inappropriate tabs for the setters and getters methods • Need camel casing for acceleration drag for the getters and setters • Has inappropriate new lines for the slot methods • Need comments for the save as pdf button method • Has an unused variable in the generate html method • Need comment to explain the replacing of < or > to prevent html injection for method generate html method • Need consistency with the placing of the brackets for each method
main.cpp	• Need appropriate new lines
unittests.cpp	• Remove unneeded comments • Need comments • Need to clean up extra new lines and spaces • Need comments with a summary for each method • Need to place opening brackets on new line for each method and if statement • Need appropriate spacing around math operators, piping operators, and operands • Need to remove the redundant if statement in the for loop for method testCalcualteVelocity
VehicleLookUp.cpp	• Need new line before each comments • Need to remove unused code • Need to remove extra new lines and spaces • Need summary comments above each method • Need camel casing for variables near lines 368 • Need to fix placing of opening brackets, for if statements and methods, onto a new line • Need to fix the placing of the else, in if else statements, onto a new line
vehicleweights.cpp	• Need an extra line above the comments

Developer's Guide

Software Required for Development

- Qt Creator
- MySQL

See Qt and MySQL Installation and Setup for instructions on installing the required software and setting up Qt with MySQL.

GitHub Code Repository

All the code and images used in the project are stored in the mfp426/MathCalc GitHub repository. Contact Mike Flamm to get access to this repository.

Unit Tests

To run the unit tests, set the boolean variable *test* in main.cpp equal to *1*. To run the application, set *test* equal to *0*. The unit tests output to the "Application Output" whether they failed or passed.

Source Code Files

- FormulasModal.pro: This file is used by Qt to manage what should be included in the project when it is built.
- centerofmasssolutions.cpp, centerofmasssolutions.h, formulaLayout.cpp, formulaLayout.h, formulas.cpp, formulas.h, minimumSpeed.cpp, minimumSpeed.h, VehicleLookUp.cpp, VehicleLookUp.h, vehicleweights.cpp, vehicleweights.h: These files define the classes used for the custom widgets in the application.
- databaseconnection.cpp, databaseconnection.h: These files define the connection to the MySQL database.
- main.cpp, main.h: These files define the *main* function that is the start point for the application. This contains the code to launch the application or run the unit tests depending on the value of the *test* variable.
- centerOfMassSolutions.ui, formulaLayout.ui, formulas.ui, minimumSpeed.ui, VehicleLookUp.ui, vehicleweights.ui: These files define what the user interface looks like for each widget. These files are managed by Qt and should not be edited directly. To edit the user interface, double click the file name in Qt Creator to bring up the user interface in Design mode.
- formulas.cpp, formulas.h: These files contain the function definitions that do the calculations in the application and generate the PDF report. These files also contain the "slot" function definitions that Qt calls when the appropriate "signal" is emitted. The "slots" are automatically connected to "signals" based on the slot name. For example, *on_CurbWeightLineEdit_textChanged()* is a "slot" that is called whenever the text in CurbWeightLineEdit is changed. For more information about "signals and slots" in Qt, see https://doc.qt.io/qt-5/signalsandslots.html.
- VehicleLookUp.h, VehicleLookUp.cpp: These files contain the function definitions for specifying the vehicle to search the database for and filling in the line edits with the information.
- unittests.h, unittests.cpp: These files contain the function definitions for running the unit tests.

Validating User Input

Qt provides an easy way to validate user input from line edits using regular expressions. For example, in VehicleLookUp.cpp, the constructor sets a validator to use for validating the input from the CurbWeightLineEdit using the following line of code: ui->CurbWeightLineEdit->setValidator(new QRegExpValidator(QRegExp("[0-9]*"), ui->CurbWeightLineEdit));. This uses the *setValidator()* function of the line edit. The validator is the regular expression "[0-9]*" which matches any amount of digits 0 through 9. This ensures that the user can only enter a positive integer number in the line edit. For more information about Qt regular expressions for validating input, see https://doc.qt.io/qt-5/qregexpvalidator.html.

Adding Additional Calculations

1. Add the needed line edits, labels, and other widgets needed to the user interface using Qt Creator's Design mode. The file formulas.ui is the main widget for the application, so add your new widgets to this widget or one of its child widgets.
2. Add a function to formulas.cpp and formulas.h to perform the desired calculation. Refer to *calculateComLong()* in formulas.cpp for an example calculation function.
3. Add the "slot" functions that will call the calculation function when appropriate. Refer to *on_overallWeight_textChanged()* in formulas.cpp for an example "slot" function.

Administrator's Manual

The code and user interface for our project was created and manipulated using Qt creator IDE. This project was largely concerned with accessing a database stored using Amazon AWS cloud services. MYSQL was installed to provide Qt with the means of connecting to the database.

The following Instructions should allow your Qt program connect with databases:

1. Use this link to go to the MYSQL installer: https://dev.mysql.com/downloads/installer/
2. Download the larger of the two MYSQL Installers
3. You will be prompted to create an account but you can continue the installation without one by clicking the print below that says "No thanks, just start my download."
4. After the download is complete, run the MYSQL Installer
5. In the "Choosing a Setup Type" menu, select "Developer Default", then press "next"
6. You should be navigated to a list of software requirements, click them and press "execute" to fulfill them
7. In the "Installation" menu for the MYSQL products click "Execute"
8. At this point MYSQL installation is complete, "Product Configuration" will not be necessary for this project

Building QMySQL Driver

Downloading MySQL allows for the building of a QMySQL Driver. This is essential in providing Qt with the extra functionality of interacting with a database.

Use the following instructions to build your QMySQL Driver:

1. Start by going to the project file for the Qt MySQL driver found using the following path: C:\Qt\5.15.1 \Src\qtbase\src\plugins\sqldrivers\mysql\mysql.pro (This path will vary depending on your version of Qt)
2. Click "Projects" on the left, click "Manage Kits" at the top, then select "Desktop Qt 5.15.1 MinGW 8.1 64-bit" and apply it
3. Remove this line "QMAKE_USE += mysql" from the project file
4. Add these lines of code to the file save the file and close it:

    INCLUDEPATH+="C:\Program Files\MySQL\MySQL Server 8.0\include"

    LIBS+="C:\Program Files\MySQL\MySQL Server 8.0\lib\libmysql.lib"

5. In your command prompt execute the following commands:

```
C:\Qt\5.15.1\mingw81_64\bin\qtenv2.bat
cd C:\Qt\5.15.1\Src\qtbase\src\plugins\sqldrivers
qmake sqldrivers.pro
cd mysql
qmake mysql.pro
mingw32-make
mingw32-make install
```

6. Then copy "C:\Program Files\ MYSQL\ MYSQL Server 8.0\lib\libmysql.dll" to C:\Windows"

7. In Qt, build the code with the same Desktop Qt 5.15.1 MinGW 8.1 64-bit -kit you used for mysql.pro

8. In the folder that contains the folder for your Qt project source code, locate the build folder for your project that should be created after the build. Copy the file C:\Qt\5.15.1\Src\qtbase\src\plugins\sqldrivers to the build folder's debug folder

Testing

Acceptance Tests

Test Plan:

The goal of the acceptance tests is to determine if the application can work the same way that the acceptance criteria, from the user stories, requires the application to run. Therefore, the testing results will be compared with the acceptance criteria, for each of the user stories, in order to determine if the tests pass. Our acceptance criteria include:

- The investigator can enter year, model, make, and trim, and after clicking a button, the text boxes get filled with the selected vehicle's information.
- The solution boxes, that display results from the math calculations, should display correct results.
- The investigator can change the different weights for a vehicle, in which the edits lead to updating the redistributed weights.
- The investigator can click button to add a new formula, fill in the needed variables by typing or selecting a value. As a result, the solution boxes are updated with formula solutions.
- The investigator can click a button and a pdf report with the vehicle information and results is created.
- The investigator can add additional vehicles, and be able to calculate the math formulas for each of those vehicles.
- The investigator is incapable of breaking the application, despite providing inappropriate or incorrect data.

Therefore, we will be testing how the application responds to user input, because this helps determine if the application is properly searching for data, placing the searched data into the appropriate textboxes, while also performing the math calculations using that data, and displaying the results of the math calculations in the proper textboxes. We will also test if the values in the textboxes update whenever the user inserts or the changes the values for textboxes, mainly for the weights and height of the wheels. This is being tested for because it will show that the application can update values of variables when given new inputs. Testing will also be done the applications ability to create a pdf report of the vehicle information, and results from the math calculations. This is because the an acceptance criteria allows the investigator to create a report of such data. Testing will not cover the ability for the application to add multiple vehicles, and calculate the math formulas for each, because the group never intended accomplish the user story with this done criteria. Testing will also not cover edge cases, such as dividing by zero, because our customer decided that it shouldn't be our biggest priority.

The test cases will be:

- The investigator selects a make, model, year and trim from the first set of dropdowns, and then selects the get vehicle info button.
    - **Reason:** This test case will be used because it will show if the application is using the inputs for searching data, and displaying the returned searched data properly.
- Given that the investigator first used the first set of dropdowns for searching, the investigator selects a make, model, year and trim from the second set of dropdowns, and then selects the get vehicle info button.
    - **Reason:** This test case will be used because it will show if the application is using the inputs for searching data, and displaying the returned searched data properly. It will also show if the second search helps fill out some of the blank textboxes that the first search could not fill.
- Given that the investigator has selected a make, model, trim, and year, but has yet to select the get vehicle info button, the investigator selects "Select a year" in the year dropdown, and then selects the get vehicle info button.
    - **Reason:** This test case will determine if the application is capable of receiving unexpected inputs from the user.
- The investigator changes the value in the textbox that contains the overall weight for a vehicle.
    - **Reason:** This test case will help determine if the application will update existing calculated values based upon the changes the user makes to the inputs.
- The investigator changes the value in the textbox that contains the weight on the front left tire.

    - **Reason:** This test case will also determine if the application will update existing calculated values based upon the changes the user makes to the inputs.
- The investigator selects the save as pdf button.
    - **Reason:** This test case will show if the application can save the all of its information in a report that will be saved as a pdf.
- Given that the application has inputs for the curb weight, wheel base, track width front and rear, overall length, width, height, and weight distribution, the investigator inputs data for the height of center of the front wheels, height of center of the right wheels raised, and weight on the front right wheels raised.
    - **Reason:** This test case will show if the application can calculate formulas without needing complete input from the program.
- The investigator inserts data for the skid distance and the acceleration drag.
    - **Reason:** This test case shows that the application can calculate additional formulas, and that it can calculate using input data only from the user.

Deliverables:

Test Case	Test Results
The investigator selects a make, model, year and trim from the first set of dropdowns, and then selects the get vehicle info button.	When the investigator performs this action, the data, that pertains to the selected vehicle, is inserted into the text boxes that associate with the data. Because not all of the data is returned from the search, some of the text boxes do not get filled with data. If the search provides enough data, then the text boxes, that display results from the math calculations, will fill with the results of those calculations, otherwise they will remain blank.
Given that the investigator first used the first set of dropdowns for searching, the investigator selects a make, model, year and trim from the second set of dropdowns, and then selects the get vehicle info button.	When the investigator performs this action, the data, that pertains to the selected vehicle, is inserted into the text boxes that associate with the data. Because not all of the data is returned from the search, some of the text boxes do not get filled with data. If a text box is already filled in, due to data returned from the first search, then the textbox will not get filled by data from the second search. If the search provides enough data, then the text boxes, that display results from the math calculations, will fill with the results of those calculations, otherwise they will remain blank.
Given that the investigator has selected a make, model, trim, and year, but has yet to select the get vehicle info button, the investigator selects "Select a year" in the year dropdown, and then selects the get vehicle info button.	When the investigator performs this action, the application crashes.
The investigator changes the value in the textbox that contains the overall weight for a vehicle.	When the investigator performs this action, the values for the remaining weight distributions, and axel weights, update to accommodate the new overall weight.
The investigator changes the value in the textbox that contains the weight on the front left tire.	When the investigator performs this action, the values for the overall weight, and the front axle weight will update to accommodate.
The investigator selects the save as pdf button.	When the investigator performs this action, the system directory will open, and the investigator will be allowed to name and save a pdf report of all information displayed in the application.
Given that the application has inputs for the curb weight, wheel base, track width front and rear, overall length, width, height, and weight distribution, the investigator inputs data for the height of center of the front wheels, height of center of the right wheels raised, and weight on the front right wheels raised.	When the investigator performs this action, the application calculates the center of mass height, and display the result in the appropriate textboxes.
The investigator inserts data for the skid distance and the acceleration drag.	When the investigator performs this action, the application calculates the minimum speed and the velocity.

Unit Tests

Test Plan:

The goal of the unit tests is to determine if the calculations for the math formulas are correctly performed. In order to determine if the calculations are accurate, the results for each calculation from the application and the unit tests will be compared. Therefore, testing will focus mainly on the formula calculations. This is because the formula calculations are an important part of the program because using the formulas it is the purpose for the application. The tests will determine the accuracy for the formulas that calculate the weight distributions, horizontal center of mass, center of mass height, minimum speed, and velocity. The tests will cover these formulas because these are the formulas that the customer provided. Although it is needed, the testing will not cover many of the edge cases, such as dividing by zero, because our customer decided that it shouldn't be our biggest priority. A note about the results in the deliverables is that the expected values are the results from the tests, the received values are the results from the application, and the Boolean values are used to determine if the tests passed.

Deliverables:

Formula Calculation	Unit test code	Results
Weight distributions	`void unitTests::testWeightDistributions() {` `QVector<QString> testVals1;` `QVector<QString> variables;` `variables << "overall weight: " << "front axle weight: " << "rear axel weight: "` `<< "front left weight:" << "front right weight:" << "rear left weight:"` `<< "rear right weight:" ;` `int index = 0;` `testVals1 << "40/60" << "4000";` `for(int i = 0; i < testVals1.length(); i++){` `QVector<float> testCalcResults;` `QVector<float> programResults;` `index++;` `if(index % 2 == 0){` `//store the hard coded values` `float hardCodeCurbWeight, hardCodeFrontDist, hardCodeRearDist;` `//store the test calcs` `float testFrontAxleWeight, testRearAxleWeight,` `testFrontTireWeight, testRearTireWeight, testOverallWeight;` `//program's calculation` `formulas.SETweightDistributionLE(testVals1[i-1]);` `formulas.SETcurbWeightLE(testVals1[i]);` `formulas.calculateWeightDistributions();` `programResults.append(formulas.GETоverallWeightLEval());` `programResults.append(formulas.GETfrontAxleWeightLEval());` `programResults.append(formulas.GETrearAxleWeightLEval());` `programResults.append(formulas.GETfrontLeftWeightLEval());` `programResults.append(formulas.GETfrontRightWeightLEval());` `programResults.append(formulas.GETrearLeftWeightLEval());` `programResults.append(formulas.GETrearRightWeightLEval());` `//test calculation` `hardCodeCurbWeight = testVals1[i].toFloat();` `hardCodeFrontDist = testVals1[i-1].split("/")[0].toFloat();` `hardCodeRearDist = testVals1[i-1].split("/")[1].toFloat();` `qDebug() << "TEST " << index/2 << ":";` `qDebug() << "Curb Weight:\t" << hardCodeCurbWeight;` `qDebug() << "Front Distribution:\t" << hardCodeFrontDist;` `qDebug() << "Rear Distribution:\t" << hardCodeRearDist;` `qDebug() << "-------------------------------------------------------";` `qDebug() << "Variable\t\t\tExpected\tReceived\tPassed";` `qDebug() << "-------------------------------------------------------";` `testOverallWeight = hardCodeCurbWeight;` `testFrontAxleWeight = hardCodeCurbWeight * (hardCodeFrontDist/100.0);` `testFrontAxleWeight = formulas.roundPt5(testFrontAxleWeight);` `testRearAxleWeight = hardCodeCurbWeight * (hardCodeRearDist/100.0);` `testRearAxleWeight = formulas.roundPt5(testRearAxleWeight);` `testFrontTireWeight = testFrontAxleWeight/2;` `testFrontTireWeight = formulas.roundPt5(testFrontTireWeight);` `testRearTireWeight = testRearAxleWeight/2;` `testRearTireWeight = formulas.roundPt5(testRearTireWeight);` `testCalcResults.append(testOverallWeight);` `testCalcResults.append(testFrontAxleWeight);` `testCalcResults.append(testRearAxleWeight);` `testCalcResults.append(testFrontTireWeight);` `testCalcResults.append(testFrontTireWeight);` `testCalcResults.append(testRearTireWeight);` `testCalcResults.append(testRearTireWeight);` `displayResults(testCalcResults, programResults, variables);` `}` `}` `}`	TEST 1 : Curb Weight: 4000 Front Distribution: 40 Rear Distribution: 60 --- Variable Expected Received Passed --- "overall weight: " 4000 4000 True "front axle weight: " 1600 1600 True "rear axel weight: " 2400 2400 True "front left weight:" 800 800 True "front right weight:" 800 800 True "rear left weight:" 1200 1200 True "rear right weight:" 1200 1200 True ==========
Horizontal center of mass		TEST 1 : Front Axle Weight: 1600

	`` void unitTests::testComLong(){ QVector<QString> testVals; QVector<QString> variables; variables << "Com Long: "; int index = 0; testVals << "1600" << "200" << "4000"; for(int i = 0; i < testVals.length(); i++){ QVector<float> testCalcResults; QVector<float> programResults; index++; if(index % 3 == 0){ float hardCodeFrontAxleWeight, hardCodeWheelBase, hardCodeOverallWeight; float testCOMLong; formulas.SETfrontAxleWeightLE(testVals[i-2]); formulas.SETwheelBaseLE(testVals[i-1]); formulas.SEToverallWeightLE(testVals[i]); formulas.calculateComLong(); programResults.append(formulas.GETcomLongSolutionsLEval()); hardCodeFrontAxleWeight = testVals[i-2].toFloat(); hardCodeWheelBase = testVals[i-1].toFloat(); hardCodeOverallWeight = testVals[i].toFloat(); qDebug() << "TEST " << index/3 << ":"; qDebug() << "Front Axle Weight:\t" << hardCodeFrontAxleWeight; qDebug() << "Wheel Base:\t" << hardCodeWheelBase; qDebug() << "Overall Weight:\t" << hardCodeOverallWeight; qDebug() << "---"; qDebug() << "Variable\t\tExpected\tReceived\tPassed"; qDebug() << "---"; testCOMLong = (hardCodeFrontAxleWeight * hardCodeWheelBase) / hardCodeOverallWeight; testCOMLong = formulas.roundPt5(testCOMLong); testCalcResults.append(testCOMLong); displayResults(testCalcResults, programResults, variables); } } } ```	Wheel Base: 200 Overall Weight: 4000 --- Variable Expected Received Passed --- "Com Long: " 80 80 True ==========
Center of mass height	``` void unitTests::testCalculateComHt() { QVector<QString> testVals1; QVector<QString> testVals2; QVector<QString> variables; int index = 0; float hardCodeOverallWeight, hardCodeFrontAxleWeight, hardCodeWheelBase, hardCodeHeightCenterFrontWheels, hardCodeHeightCenterRearWheelsRaised, hardCodeWeightFrontWheelsRearRaised; float testComHt, testWeightDiff, testHeightDiff; QString testComHtStringVersion; variables << "Com Ht "; //values in groups of six: testVals1 << "4000" << "1600" << "200" << "12" << "123" << "123" << "400022" << "160023" << "2003" << "122" << "12233" << "12233"; for(int i = 0; i < testVals1.length(); i++) { QVector<float> testCalcResults; QVector<float> programResults; index++; if(index % 6 == 0){ formulas.SETtoverallWeightLE(testVals1[i-5]); formulas.SETfrontAxleWeightLE(testVals1[i-4]); formulas.SETwheelBaseLE(testVals1[i-3]); formulas.SETheightCenterFrontWheelsLE(testVals1[i-2]); formulas.SETheightCenterRearWheelsRaisedLE(testVals1[i-1]); formulas.SETweightFrontWheelsRearRaisedLE(testVals1[i]); formulas.calculateComHt(); programResults.append(formulas.GETcomHtSolutionsLEval()); hardCodeOverallWeight = testVals1[i-5].toFloat(); hardCodeFrontAxleWeight = testVals1[i-4].toFloat(); hardCodeWheelBase = testVals1[i-3].toFloat(); ```	TEST 1 : Overall Weight: 4000 Front Axle Weight: 1600 Wheel Base: 200 Height Center Front Wheels: 12 Height Center Rear Wheels raised: 123 Weight Center Front Wheels rear raised: 123 --- Variable Expected Received Passed --- "Com Ht " -98.5 -98.5 True ========== TEST 2 : Overall Weight: 400022 Front Axle Weight: 160023 Wheel Base: 2003 Height Center

```
hardCodeHeightCenterFrontWheels = testVals1[i-2].toFloat();
hardCodeHeightCenterRearWheelsRaised = testVals1[i-1].toFloat();
hardCodeWeightFrontWheelsRearRaised = testVals1[i].toFloat();
qDebug() << "TEST " << index/6 << ":";
qDebug() << "Overall Weight:\t\t\t" << hardCodeOverallWeight;
qDebug() << "Front Axle Weight:\t\t\t" << hardCodeFrontAxleWeight;
qDebug() << "Wheel Base:\t\t\t" << hardCodeWheelBase;
qDebug() << "Height Center Front Wheels:\t\t" <<hardCodeHeightCenterFrontWheels;
qDebug() << "Height Center Rear Wheels raised:\t" <<
hardCodeHeightCenterRearWheelsRaised;
qDebug() << "Weight Center Front Wheels rear raised:\t"
<<hardCodeWeightFrontWheelsRearRaised;
qDebug() << "-------------------------------------------------------";
qDebug() << "Variable\t\tExpected\tReceived\tPassed";
qDebug() << "-------------------------------------------------------";
testWeightDiff = hardCodeWeightFrontWheelsRearRaised - hardCodeFrontAxleWeight;
testHeightDiff = hardCodeHeightCenterRearWheelsRaised -
hardCodeHeightCenterFrontWheels;
testComHt = hardCodeHeightCenterFrontWheels + ((testWeightDiff *
hardCodeWheelBase * qSqrt((hardCodeWheelBase * hardCodeWheelBase) -
(testHeightDiff * testHeightDiff))) / (hardCodeOverallWeight * testHeightDiff));
testComHt = formulas.roundPt5(testComHt);
testComHtStringVersion.setNum(testComHt);
testComHt = testComHtStringVersion.toFloat();
testCalcResults.append(testComHt);
displayResults(testCalcResults, programResults, variables);
}
}
}
```

```
Front Wheels: 122
Height Center
Rear Wheels
raised: 12233
Weight Center
Front Wheels
rear raised: 12233
-------------------------
-------------------------
---------
Variable
Expected
Received      Pas
sed
-------------------------
-------------------------
---------
"Com Ht "
-2.14748
e+09  -2.14748
e+09 True
==============
==============
==========
```

Minimum speed

```
void unitTests::testCalculateMinimumSpeed() {
  QVector<QString> testVals1;
  QVector<QString> testVals2;
  QVector<QString> variables;
  variables << "minimumSpeed";
  int index = 0;
  //values in groups of two: first is acceleration drag, and the second is skid
distance
  testVals1 << "2" << "100"
  << "1" << "30"
  << "3" << "2622133478"
  << "0.3" << "0.5";
  float hardCodeSkidDistance, hardCodeAccelDrag;
  float testMinSpeed;
  QString testMinSpeedStringVersion;
  for(int i = 0; i < testVals1.length(); i++){
    QVector<float> testCalcResults;
    QVector<float> programResults;
    index++;
    if(index % 2 == 0){
      formulas.SETskidDistanceLE(testVals1[i]);
      formulas.SETAccelDragLE(testVals1[i - 1]);
      formulas.calculateMinimumSpeed();
     programResults.append(formulas.GETminimumSpeedLEval());
     hardCodeSkidDistance = testVals1[i].toFloat();
     hardCodeAccelDrag = testVals1[i-1].toFloat();
     qDebug() << "TEST " << index/2 << ":";
     qDebug() << "Skid Distance:\t" << hardCodeSkidDistance;
     qDebug() << "Acceleration Drag:\t" << hardCodeAccelDrag;
     qDebug() << "-----------------------------------------------";
     qDebug() << "Variable\t\tExpected\tReceived\tPassed";
     qDebug() << "-----------------------------------------------";
     testMinSpeed = qSqrt(30 * hardCodeSkidDistance * hardCodeAccelDrag);
     testMinSpeed = formulas.round2Places(testMinSpeed);
     testMinSpeedStringVersion.setNum(testMinSpeed);
     testMinSpeed = testMinSpeedStringVersion.toFloat();
     testCalcResults.append(testMinSpeed);
```

```
TEST 1 :
Skid Distance:
100
Acceleration
Drag: 2
-------------------------
-------------------------

Variable
Expected
Received Passed
-------------------------
-------------------------

"minimumSpeed"
77.46       77.46
  True
==============
==============
====
TEST 2 :
Skid Distance: 30
Acceleration
Drag: 1
-------------------------
-------------------------

Variable      Ex
pected Received
Passed
-------------------------
-------------------------

"minimumSpeed"
30        30      T
rue
==============
==============
=======
TEST 3 :
Skid Distance:
2.62213e+09
Acceleration
Drag: 3
```

```
     displayResults(testCalcResults, programResults, variables);
    }
  }
}
```

```
----------
----------

Variable
Expected
Received Passed
----------
----------

"minimumSpeed"
485790   485790
  True
==========
==========
=========
TEST 4 :
Skid Distance: 0.5
Acceleration
Drag: 0.3
----------
----------

Variable          E
xpected
Received Passed
----------
----------

"minimumSpeed"
2.12        2.12
True
==========
==========
==========
```

Velocity

```
void unitTests::testCalculateVelocity() {
QVector<QString> testVals;
QVector<QString> variables;
variables << "velocity";
float const conversion = 1.466;
float hardCodeMinSpeed, testVelocity;
QString testVelocityStringVersion;
int index = 0;
//each test value is minimum speed
testVals << "100" << "2" << "30" << "0.5" << "2001" << "23342232423";
for(int i = 0; i < testVals.length(); i++){
QVector<float> testCalcResults;
QVector<float> programResults;
index++;
if(index % 1 == 0){
formulas.SETminimumSpeedLE(testVals[i]);
formulas.calculateVelocity();
programResults.append(formulas.GETvelocityLEval());
hardCodeMinSpeed = testVals[i].toFloat();
qDebug() << "TEST " << index << ":";
qDebug() << "Minimum speed:\t" << hardCodeMinSpeed;
qDebug() << "-----------------------------------------------";
qDebug() << "Variable\t\tExpected\tReceived\tPassed";
qDebug() << "-----------------------------------------------";
testVelocity = conversion * hardCodeMinSpeed;
qDebug() << "Calculated velocity: " << testVelocity;
testVelocity = formulas.round2Places(testVelocity);
qDebug() << "Rounded calculated velocity: " << testVelocity;
//conversion used for large numbers
testVelocityStringVersion.setNum(testVelocity);
qDebug() << "String version of rounded calc velocity: " <<
testVelocityStringVersion;
testVelocity = testVelocityStringVersion.toFloat();
testCalcResults.append(testVelocity);
displayResults(testCalcResults, programResults, variables);
}
}
}
```

```
TEST 1 :
Minimum speed:
100
----------
----------

Variable
Expected
Received Passed
----------
----------

"velocity"
146.6    146.6
 True
==========
==========
=====
TEST 2 :
Minimum speed: 2
----------
----------

Variable
Expected
Received Passed
----------
----------

"velocity"
2.93      2.93
  True
==========
==========
=======
TEST 3 :
Minimum speed:
30
----------
----------

Variable
Expected
Received Passed
----------
```

```
----------

"velocity"
43.98     43.98
 True
==========
==========
======
TEST 4 :
Minimum speed:
0.5
----------
----------

Variable
Expected
Received Passed
----------
----------

"velocity"
0.73       0.73
   True
==========
==========
=========
TEST 5 :
Minimum speed:
2001
----------
----------

Variable
Expected
Received Passed
----------
----------

"velocity"
2933.47  2933.47
 True
==========
==========
==========
TEST 6 :
Minimum speed:
2.33422e+10
----------
----------

Variable
Expected
Received       Pas
sed
----------
----------

"velocity"
-2.14748
e+07   -2.14748
e+07  True
==========
==========
==========
```

Testing Quality Review

Process

As a group, we went through the application. We checked that the user story acceptance criteria was met and that the unit tests passed.

Issues

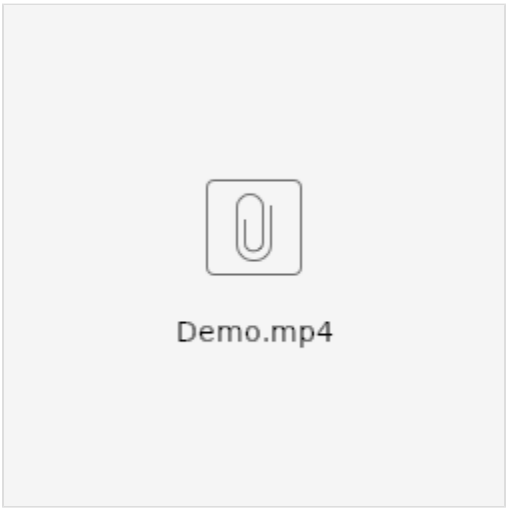
- There are not enough tests for some of the formula calculations
- Not much coverage for edge cases in unit tests

- The user can not add additional formulas as specified in user stories
- The user can not select a value for a variable as specified in user stories
- If the user clicks either "Get vehicle Info" button without entering the vehicle make, model, and year, the application crashes
- If the user selects a year, but then goes back to "Select a year," the model list is still populated and the user can still select a model
- If the user divides by 0, a large value is given in the solutions

Technical Metric Collection

- Estimated story points: 44 points for all stories
- Actual lines of code: 2123 lines of code including blank lines in code files
- Product size: 5 completed user stories
- Defects: 3 defects found during quality review

Demonstration



Word Count

Tabitha Holloway: 3230 words

Itai Kumengisa: 390

Christopher Butler: 545 words