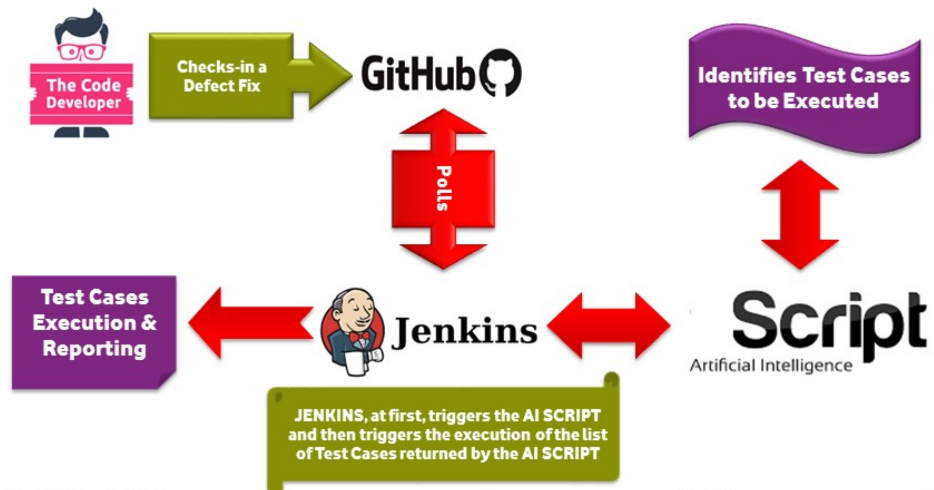
Smart regression suite

<u>What exactly is intelligent regression suite.</u>

Imagine a situation where a regression suite is generated automatically for every change made by developer; and this would be done in matter of seconds. Your next daily run would have precise regression suite with 100% functional coverage.

<u>What we expect to achieve here is:</u>

- ✓ A system which would intelligently identify set of test cases to execute for every changes made by developers;
- ✓ Improved Regression Test Suite;
- ✓ To extend the advantages of the Test Automation;
- ✓ Implement the CI concept with more effective testing of application;
- ✓ Create a system with minimal human intervention and optimized use of available resources.



<u>This can be achieved in the following way:</u>

1. Use code coverage tool to generate test coverage of existing/new test cases.
2. Finding areas of a program not exercised by a set of test cases.
3. Create additional test cases to increase coverage.
4. Determining a quantitative measure of code coverage, which is an indirect measure of quality.
5. Extend the Code Coverage capabilities to intelligently select regression test cases
6. Auto selection of regression test cases based on check-ins done by developer.

Steps:

- ✓ For Demo purpose we have written the following code
- ✓ Integrated with Jacoco Plugin for measuring test coverage
- ✓ Executed test cases and measured coverage.
- ✓ Analysed the coverage
- ✓ Wrote additional test cases
- ✓ Executed and re-measure the test coverage

```
1 package DemoPkg;
2
3 public class Methods {
4
5 public static int add(int a, int b) {
6
7     int c = a+b;
8     System.out.println("Addition result: "+c);
9     return c;
10 }
11
12 public static int sub(int a, int b) {
13
14     int c = a-b;
15     if(c<0) {
16         c = (c*(-1));
17         System.out.println("The negetive value is converted to positive");
18     }
19     System.out.println("Subtraction result: "+c);
20     return c;
21 }
22
23 public static int mul(int a, int b) {
24
25     int c = a*b;
26     System.out.println("Multiplication result: "+c);
27     return c;
28 }
29
30 public static int div(int a, int b) {
31     int c=0;
32     if (a>b) {
33         c = a/b;
34         System.out.println("Division result: "+c);
35     }
36     else {
37         System.out.println("Invaid Input");
38     }
39     return c;
40 }
41
42 }
```

Details:

- Following test cases were written to test above code

TestCase Name	Automated
TC-01	Yes
TC-02	Yes
TC-03	Yes
TC-04	Yes
TC-06	No
TC-07	Yes
TC-08	Yes
TC-09,TC-10	Yes
TC-11	No

- Code-Coverage: Code Coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs.

  Code coverage analysis is the process of:

✓ Finding areas of a program not exercised by a set of test cases,
✓ Creating additional test cases to increase coverage, and
✓ Determining a quantitative measure of code coverage, which is an indirect measure of quality

- Next Step is to execute the test case and measure coverage. PSB, code coverage graph for your reference.

MainClass (Feb 23, 2018 3:53:39 PM)

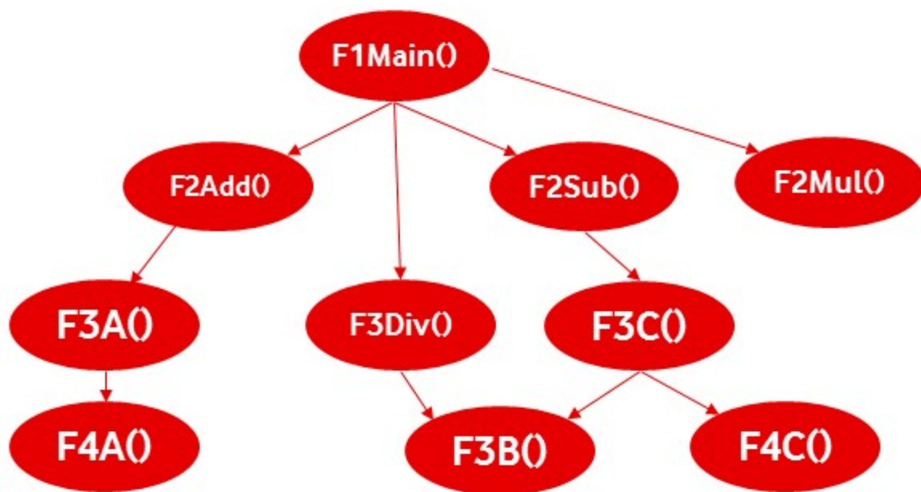
Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
DemoProj	80.3 %	94	23	117
src	80.3 %	94	23	117
DemoPkg	80.3 %	94	23	117
Methods.java	79.0 %	64	17	81
Methods	79.0 %	64	17	81
div(int, int)	41.7 %	10	14	24
add(int, int)	100.0 %	15	0	15
mul(int, int)	100.0 %	15	0	15
sub(int, int)	100.0 %	24	0	24
MainClass.java	83.3 %	30	6	36
MainClass	83.3 %	30	6	36
main(String[])	90.9 %	30	3	33

- As the test cases are getting executed we create a database having test case to function mapping. This data is generated for every test case executed. Following data base was created after our execution. For now, we are generating data up to two levels.

Function	TestCase Name	Automated
F2Add	TC-01	Yes
F2Sub	TC-02	Yes
F2-Mul	TC-03	Yes
F2-Div	TC-04	Yes
F3-A	Tc-06	No
F3-B	TC-07	Yes
F4-A	TC-08	Yes
F4-B	TC-09,TC-10	Yes
F4-C	TC-11	No

- Create Functional Dependency:

We developed a script that generates the structure of the given code with respect to the mapping of all the functions available in the test application as below:



- Merging code coverage data with functional dependency we can generate the following data base which can help us in identifying test cases:

Sr.No	Function	Level1-TC	Level2-TC	Level3-TC
1	F2Add	TC-01	TC-06	TC-08
2	F2Sub	TC-02	TC-07,	TC-09,TC-10,TC-11
3	F2-Mul	TC-03		
4	F2-Div	TC-04	TC-11	
5	F3-A	TC-06	TC-08	
6	F3-B	TC-07	TC-09,TC-11,TC-10	
7	F4-A	TC-08		
8	F4-B	TC-09,TC-10		
9	F4-C	TC-11		

Now if there is a fix in function F3-B, we can instantly find the test cases to be executed for testing this function, i.e. TC-07, TC-09, TC-10 & TC-11. Duplicity of TC is taken of. This can be done with the help of above data base. Once integrated with devops it would execute automatically.

Flow Chart representation of the Intended End-to-End System