An Architecture for Test Case Prioritization Based on Change and Effect Graphs Using Bayesian Networks

Authors: Ekincan Ufuktepe and Tugkan Tuglular

Presented by: Ekincan Ufuktepe

3rd Workshop on Dependability at IZTECH

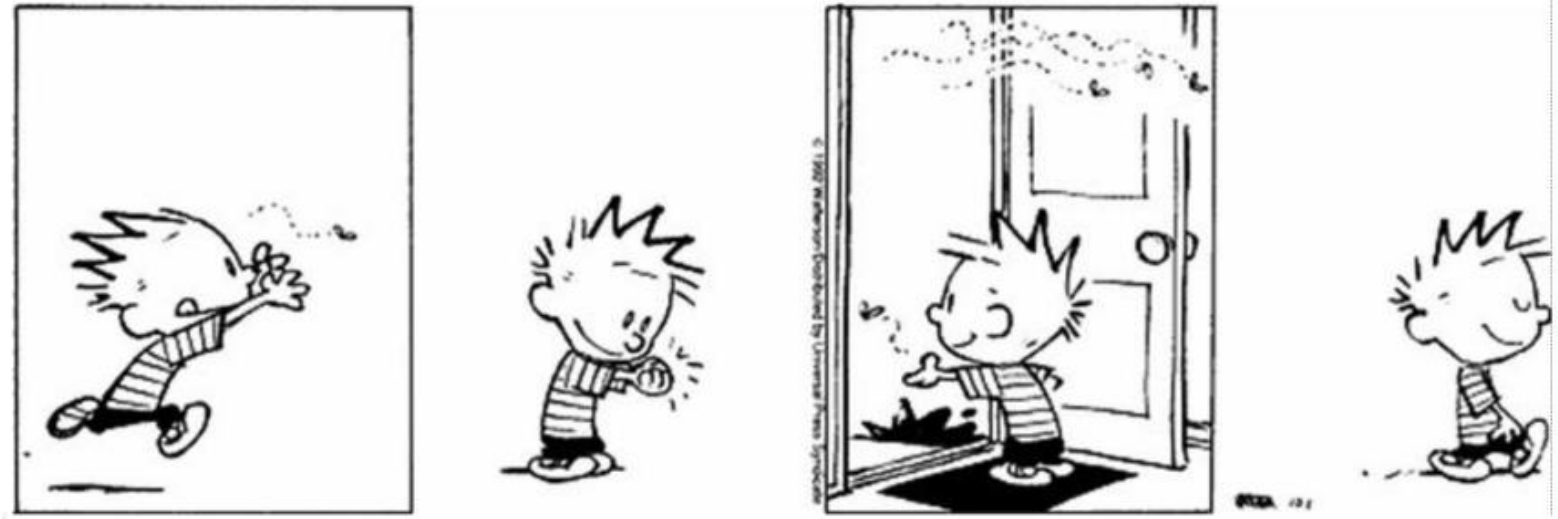
8 May 2017

Outline

1. Introduction
2. Test Case Prioritization
3. Bayesian Network
4. Our Previous Architecture
5. Architecture
    1. Change Effect Graph
    2. Change percentage of a code
    3. Probability of change potential
6. Conclusion

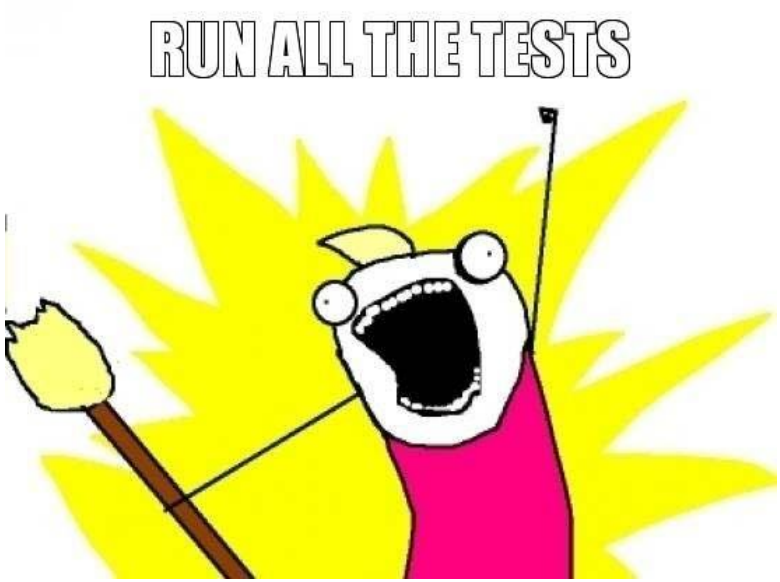
Introduction

Regression:
"when you fix one bug, you introduce several newer bugs."



Introduction

- When a software is modified, to reduce the risks, we use ***Regression Testing***.
- What do we do?
  - ***Re-run*** the test cases after modification.
- Re-run all test cases?



Introduction

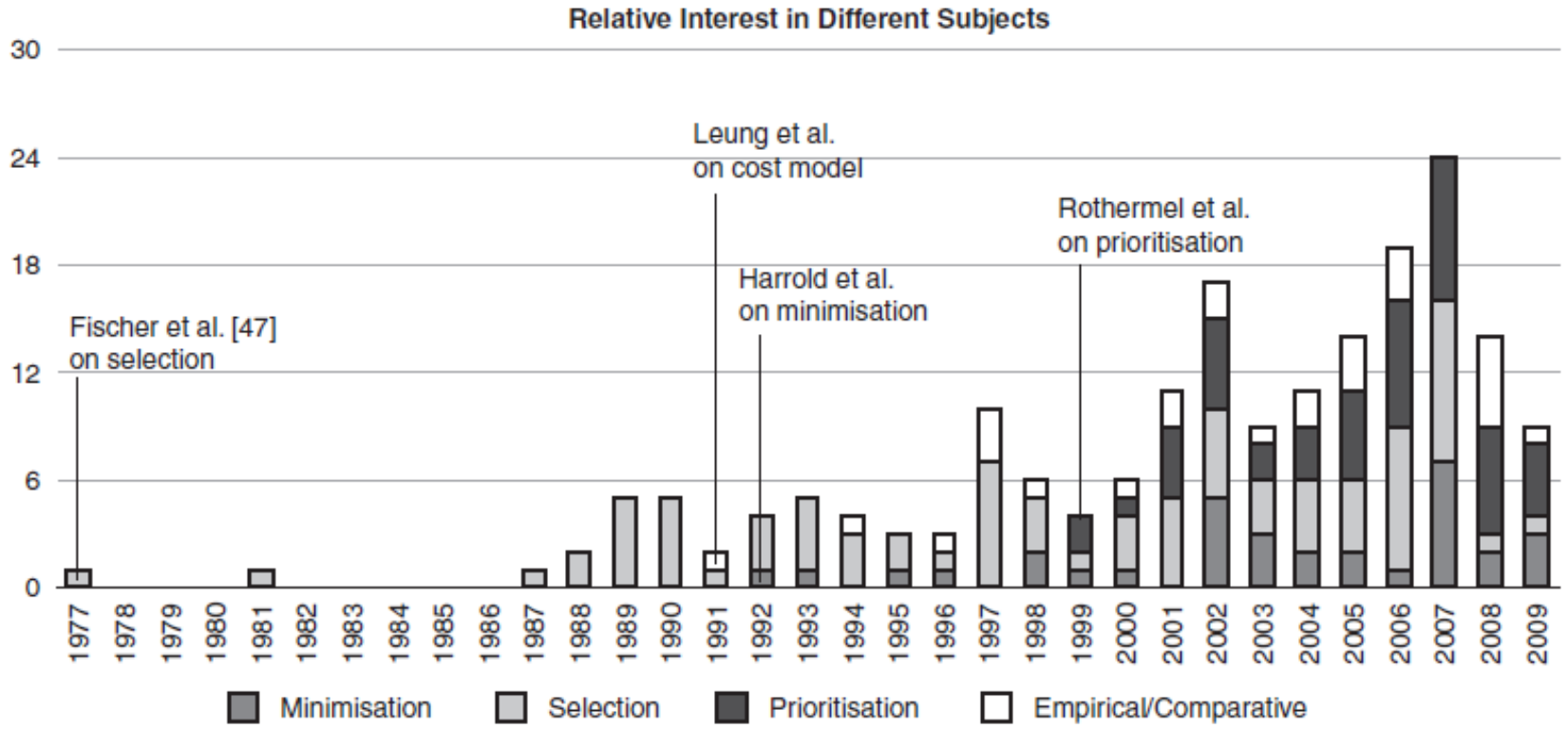
- What could go worse?
  - Running the entire test suite for an industrial project has reported that the execution has taken **seven weeks**. (Report year: 2000)[1,2]
  - Google runs **~100.000.000 test cases per day**.[3]
  - Google performs more than **20 code changes per minute** and **50% of the code changes every month**. [3]

[1]Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Prioritizing test cases for regression testing. Software Engineering Notes, 25(5):102–112, 2000.

[2]Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, 28(2):159–182, 2002.

[3]Ashish Kumar. Development at the speed and scale of google. International Software Development Conference, 2010, Presentation slides are available at: qconsf.com/sf2010/

Test Case Prioritization



[4]S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," Software Testing, Verification and Reliability, vol. 24, no. 8, pp. n/a–n/a, 2010

Test Case Prioritization

- To reduce the cost of ***Regression Testing*** there are three techniques that could be categorized:
  - Regression Test case Selection (RTS).
  - Regression Test case Minimization (RTM).
  - Regression Test case Prioritization (RTP).

Test Case Prioritization

- ***Regression Test case Selection*** focuses on covering the changed code between versions of software under test.
- ***Regression Test case Minimization*** aims to identify redundant test cases and to remove them from the test suite in order to reduce the size of the test suite.
- ***Regression Test case Prioritization*** focuses on identifying the ideal ordering of test cases.
  - Enhances the **rate of fault detection**.
  - Provide the **maximum coverage sooner**.
  - Formally; its objective is to find the best permutation of the test suite.

Test Case Prioritization

- There are two well known and basic prioritization techniques;
  - Total Technique (TT) – ***Iterative approach***
  - Additional Technique (AT) – ***Greedy approach***
- Both select one test case on iteration.
- ***TT:*** Prioritize test cases through maximizing the total number of entities.
- ***AT:*** Selects the test case that covers the highest number of entities that have **not been covered yet**.

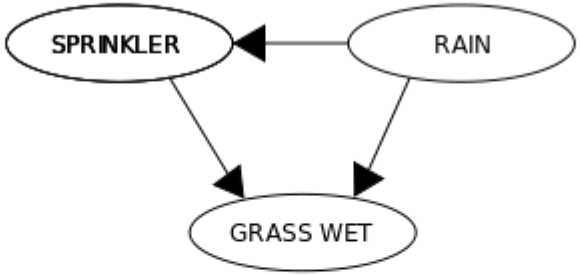
Bayesian Network

- A Bayesian network is a graphical model that encodes probabilistic relationships among variables of interest. When used in conjunction with statistical techniques, the graphical model has several advantages for data analysis[5].

[5] D. Heckerman. A Tutorial on Learning With Bayesian Networks, Technical Report, Microsoft Research, no. MSR-TR-96-06, 1996, http://research.microsoft.com/apps/pubs/?id=69588

Bayesian Network

RAIN	SPRINKLER T	SPRINKLER F
F	0.4	0.6
T	0.01	0.99



RAIN T	RAIN F
0.2	0.8

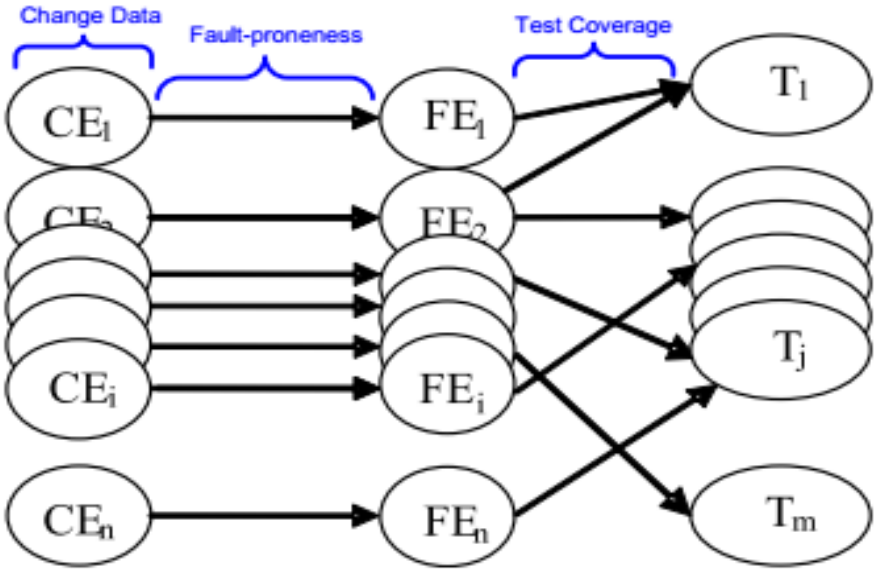
SPRINKLER	RAIN	GRASS WET T	GRASS WET F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

[6] http://en.wikipedia.org/wiki/Bayesian_network

Our Previous Architecture

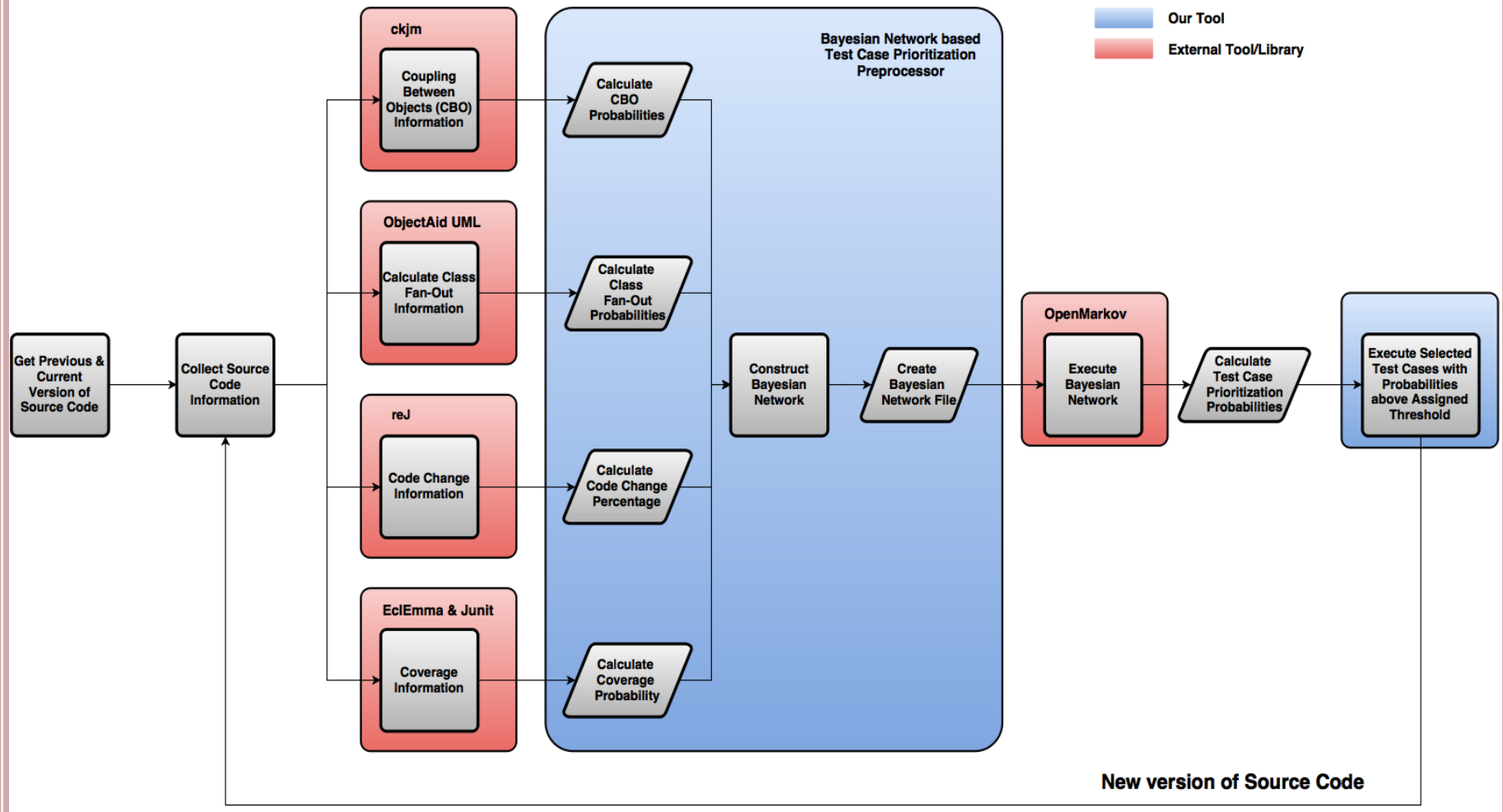
- There are 3 types of node in the Bayesian Network structure:
  - Class Nodes
  - Fault-proneness Nodes
  - Test Case Nodes

Our Previous Architecture



[7] S. Mirarab and L. Tahvildari, Fundamental Approaches to Software Engineering, vol. 4422 of Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007

Our Previous Architecture



[8] Ufuktepe, E. and Tuglular, T., 2016, June. Automation Architecture for Bayesian Network Based Test Case Prioritization and Execution. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual* (Vol. 2, pp. 52-57). IEEE.

Architecture

- The proposed architecture utilizes three types of information
  - Call graph (Dependency)
  - Change percentage of a code (between two versions)
  - Potential of change probability (Dataflow analysis)

- These information will be used in a probabilistic model: ***Bayesian Network***

Change Effect Graph

- ***Call graph*** is a graphical model, which represents calling relationships between subroutines of a program.

- Call graph is used to construct the layout of software.

- It also shows dependecies of each method.

Change Effect Graph

- Call graph is only used to extract the **nodes** and **edges**.
- The edge directions are reformed depending on the **change relationship between caller and callee**.
- Since call graph edge directions could be changed, instead of mentioning graph as a call graph we call it ***change effect graph***.

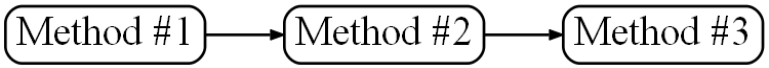
Change Effect Graph

- Let's assume that
  - Caller is Node #1
  - Callee is Node #2

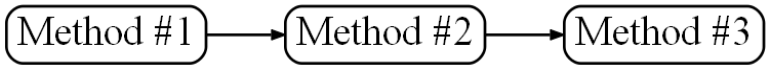
Caller Status	Callee Status	Condition	Relation
Unchanged	Unchanged	-	Node#1 → Node#2
Unchanged	Changed	-	Node#1 ← Node#2
Changed	Unchanged	-	Node#1 → Node#2
Changed	Changed	Caller Change > Callee Change	Node#1 → Node#2
Changed	Changed	Caller Change = Callee Change	Node#1 → Node#2
Changed	Changed	Caller Change < Callee Change	Node#1 ← Node#2

Change Effect Graph

Original Call Graph

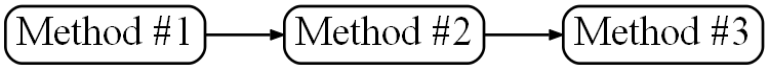


Method #1 Unchanged – Method #2 Unchanged

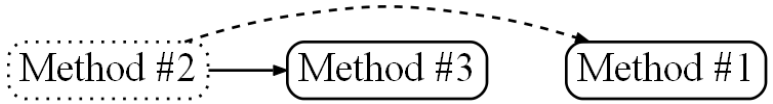


Change Effect Graph

Original Call Graph

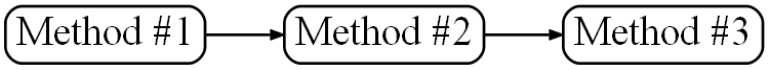


Method #1 Unchanged – Method #2 Changed

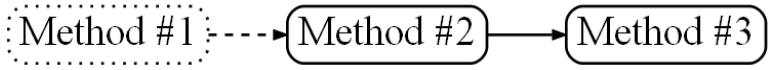


Change Effect Graph

Original Call Graph

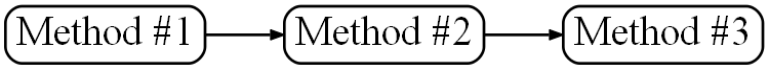


Method #1 Changed – Method #2 Unchanged



Change Effect Graph

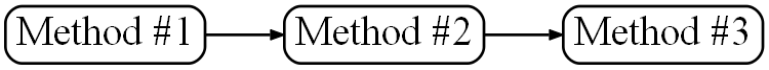
Original Call Graph



Method #1 Changed > Method #2 Changed

Change Effect Graph

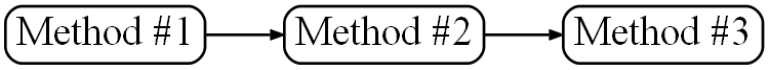
Original Call Graph



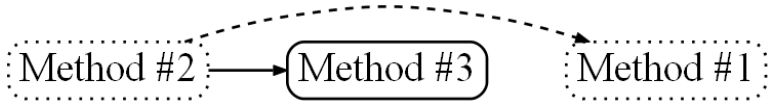
Method #1 Changed = Method #2 Changed

Change Effect Graph

Original Call Graph



Method #1 Changed < Method #2 Changed



Change percentage of a code

- To calculate the change percentage of a code we need 2 (Current & Previous) versions of a code.

- For more precise calculation of change percentage, rather than analyzing the source codes, ***bytecodes*** (Java) of two versions are compared.
  - Avoid difference in coding styles.

Change percentage of a code

- The change percentage is simply calculated by changed lines of bytecode.

- Change percentages are calculate by method level.

$$\text{Change Percentage} = \frac{Total\ Changed\ Lines\ of\ bytecode}{Total\ Lines\ of\ bytecode}$$

Probability of change potential

- A change in a method can effect both it's ***caller*** and ***callee***.
  - A changed caller method can effect it's callee by **passing an input to its parameter**.
  - A changed callee method can effect it's caller by it's **returned value**.

- Therefore, dataflow analysis is performed.

Probability of change potential

```
1.  // Godbach Conjuncture Test
2.  public void goldbach(int a, int b)
3.  {
4.       System.out.println("First input: "+a);
5.       System.out.println("Second input: "+b);
6.       int c;
7.       c = sum(a, b);
8.       if((isPrime(a) && isPrime(b)) && (a>0 && b>0))
9.       {
10.           System.out.println("Your both inputs are prime, let's test Goldbach's
    Conjuncture");
11.           if(c%2 == 0)
12.                System.out.println("Goldbach's Conjuncture Satisfied");
13.           else
14.                System.out.println("Goldbach's Conjuncture Failed");
15.      }
16.      else
17.           System.out.println("At least one of your input is not a prime or greater
    than 0");
18. }
19.
20. // Summation method
21. public int sum(int a, int b)
22. {
23.      int sum;
24.      sum = a + b;
25.      return sum;
26. }
27.
28. // Primality check method
29. public boolean isPrime(int num)
30. {
31.      // Goldbach assumes that 1 is prime
32.      if (num==1)
33.           return true;
34.      else
35.      {
36.           for(int i=2;i<=Math.sqrt(num);i++)
37.           {
38.                if(num%i==0)
39.                     return false;
40.           }
41.           return true;
42.      }
43. }
```

Probability of change potential

- Caller → Callee
  - Caller has an influence on Callee.
  - $$Caller's\ effect\ on\ Callee = \frac{LOC\ where\ Callee\ Method's\ parameter\ first\ used\ line\ to\ end\ of\ its\ scope}{Total\ Lines\ of\ Code\ of\ Callee\ Method}$$

Probability of change potential

```
1.  // Godbach Conjuncture Test
2.  public void goldbach(int a, int b)
3.  {
4.       System.out.println("First input: "+a);
5.       System.out.println("Second input: "+b);
6.       int c;
         c = sum(a, b);
7.       if((isPrime(a) && isPrime(b)) && (a>0 && b>0))
8.       {
9.            System.out.println("Your both inputs are prime, let's test Goldbach's
    Conjuncture");
10.           c = sum(a, b);
11.           if(c%2 == 0)
12.                System.out.println("Goldbach's Conjuncture Satisfied");
13.           else
14.                System.out.println("Goldbach's Conjuncture Failed");
15.      }
16.      else
17.           System.out.println("At least one of your input is not a prime or greater
    than 0");
18. }
19.
20. // Summation method
21. public int sum(int a, int b)
22. {
23.      int sum;
24.      sum = a + b;
25.      return sum;
26. }
27.
28. // Primality check method
29. public boolean isPrime(int num)
30. {
31.      // Goldbach assumes that 1 is prime
32.      if (num==1)
33.           return true;
34.      else
35.      {
36.           for(int i=2;i<=Math.sqrt(num);i++)
37.           {
38.                if(num%i==0)
39.                     return false;
40.           }
41.           return true;
42.      }
43. }
```

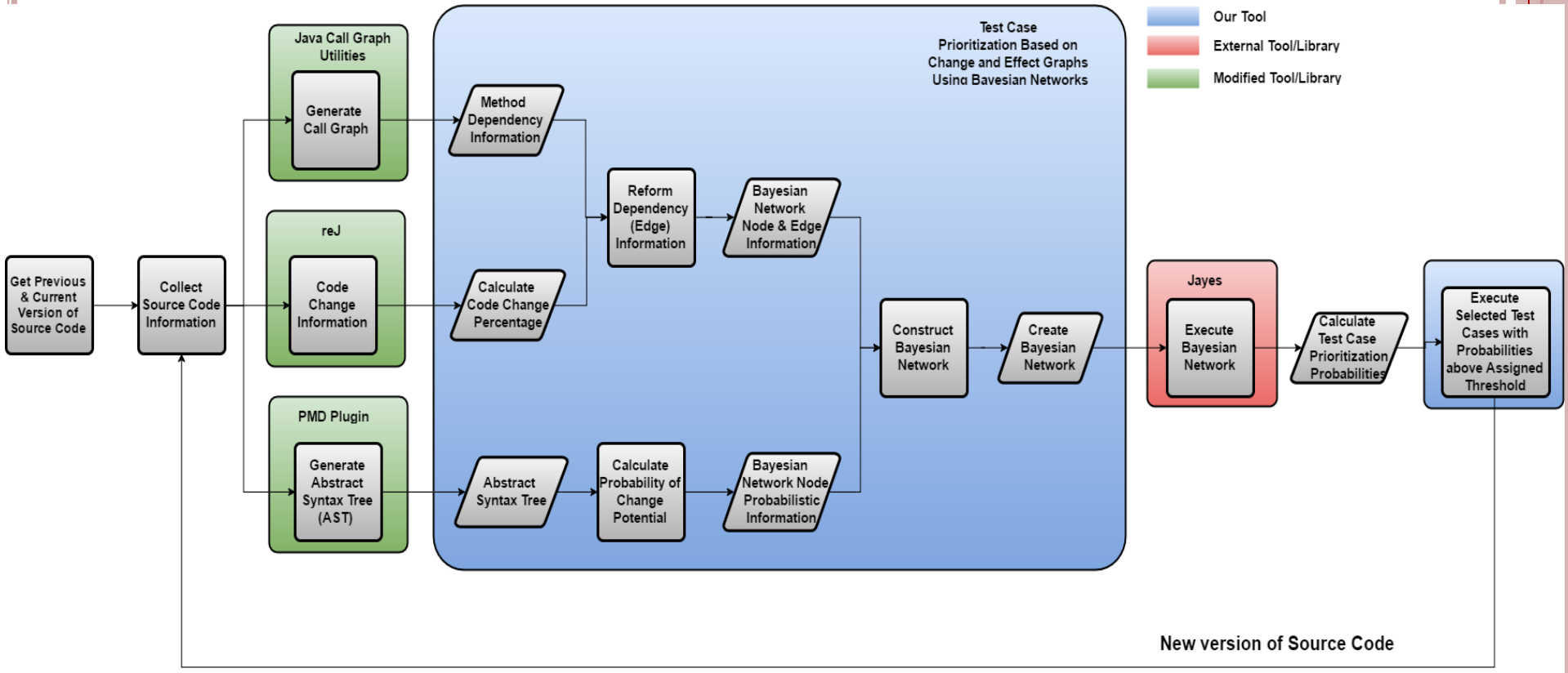
Probability of change potential

- Caller ← Callee
  - Callee has an influence on Caller
  - $$Callee's\ effect\ on\ Caller = \frac{LOC\ from\ the\ line\ where\ Caller\ Method\ has\ first\ called\ Callee\ to\ end\ of\ its\ scope}{Total\ Lines\ of\ Code\ of\ Caller\ Method}$$

Probability of change potential

```
1.  // Godbach Conjuncture Test
2.  public void goldbach(int a, int b)
3.  {
4.        System.out.println("First input: "+a);
5.        System.out.println("Second input: "+b);
6.        int c;
7.        c = sum(a, b);
8.        if((isPrime(a) && isPrime(b)) && (a>0 && b>0))
9.        {
10.             System.out.println("Your both inputs are prime, let's test Goldbach's
    Conjuncture");
11.             if(c%2 == 0)
12.                  System.out.println("Goldbach's Conjuncture Satisfied");
13.             else
14.                  System.out.println("Goldbach's Conjuncture Failed");
15.       }
16.       else
17.             System.out.println("At least one of your input is not a prime or greater
    than 0");
18. }
19.
20. // Summation method
21. public int sum(int a, int b)
22. {
23.       int sum;
24.       sum = a + b;
25.       return sum;
26. }
27.
28. // Primality check method
29. public boolean isPrime(int num)
30. {
31.       // Goldbach assumes that 1 is prime
32.       if (num==1)
33.             return true;
34.       else
35.       {
              for(int i=2;i<=Math.sqrt(num);i++)
36.           for(int i=2;i<num/2;i++)
37.           {
38.                if(num%i==0)
39.                     return false;
40.           }
41.           return true;
42.       }
43. }
```

Architecture



Conclusion

- By combining the information below;
  - Change information
  - Dependency
- A probabilistic model Bayesian Network has been used to prioritize test cases.
- As a result, ***Rate of fault detection*** is expected to be increased.

Thank You