Is This a Bug or an Obsolete Test?

What is the problem?

```
public class Account{
  private String thisAcnt;
  private double balance;
  public Account(String acnt, double amt) {
     this.balance=amt;
     this.Acnt=acnt;}
  public double getBalance(){
     return this.balance;}
  public void setBalance(double balance){
     this.balance=balance;}
  public void sendto(String account) {
    // send money to target account}
  public boolean deposit(double amt) {
    if (amt > 0) {
      setBalance(getBalance() + amt);
      return true;}
    else
      return false;}
  public boolean withdraw(double amt) {
      double fee=amt*0.1;
      if(getBalance()>=amt+fee && amt>0){
          setBalance(getBalance()-amt-fee);
          return true;
      }
      else return false;
  }
  public boolean transfer(double amt, String anoAcnt) {
      double fee=0.01;  // should be fee=amt*0.1
      if(getBalance()>=amt+fee && amt>0){
        withdraw(amt+fee);
        sendto(anoAcnt);
        return true;
      }
      else return false;
  }
```

previous version

later version

```
public class Testcases
 Account a;
 protected void setUp()
    a=new Account(100.0,"user1");
 protected void tearDown()

 public void test1()
  a.transfer(50.0,"user2");
  a.withdraw(40.0);
  assertEquals(9.5,a.getBalance());

 public void test2()
  a.withdraw(40.0);
  assertEquals(56,a.getBalance();//should
be 60
 ...
```

```
public class Account{
 private String thisAcnt;
 private double balance;
 public Account(String acnt, double amt) {
   this.balance=amt;
   this.Acnt=acnt;}
 public double getBalance(){
   return this.balance;}
 public void setBalance(double balance){
   this.balance=balance;}
 public void sendto(String account) {
  // send money to target account}
 public boolean deposit(double amt) {
  if (amt > 0) {
   setBalance(getBalance() + amt);
   return true;}
  else
   return false;}
 public boolean withdraw(double amt) {
    double fee=amt*0.1;
    if(getBalance()>=amt+fee && amt>0){
        setBalance(getBalance()-amt-fee);
        return true;
    }
    else return false;
}
 public boolean transfer(double amt, String anoAcnt) {
    double fee=0.01;  // should be fee=amt*0.1
    if(getBalance()>=amt+fee && amt>0){
      withdraw(amt+fee);
      sendto(anoAcnt);
      return true;
    }
    else return false;
}
```

previous version

later version

```
public class Testcases
 Account a;
 protected void setUp()
    a=new Account(100.0,"user1");
 protected void tearDown()

 public void test1()
  a.transfer(50.0,"user2");
  a.withdraw(40.0);
  assertEquals(9.5,a.getBalance());

 public void test2()
  a.withdraw(40.0);
  assertEquals(56,a.getBalance();//should
be 60
  ...
```

```
public class Account{
 private String thisAcnt;
 private double balance;
 public Account(String acnt, double amt) {
   this.balance=amt;
   this.Acnt=acnt;}
 public double getBalance(){
   return this.balance;}
 public void setBalance(double balance){
   this.balance=balance;}
 public void sendto(String account) {
  // send money to target account}
 public boolean deposit(double amt) {
  if (amt > 0) {
   setBalance(getBalance() + amt);
   return true;}
  else
   return false;}
 public boolean withdraw(double amt) {
    double fee=amt*0.1;
    if(getBalance()>=amt+fee && amt>0){
       setBalance(getBalance()-amt-fee);
       return true;
    }
    else return false;
 }
 public boolean transfer(double amt, String anoAcnt) {
    double fee=0.01;  // should be fee=amt*0.1
    if(getBalance()>=amt+fee && amt>0){
      withdraw(amt+fee);
      sendto(anoAcnt);
      return true;
    }
    else return false;
 }
```

previous version

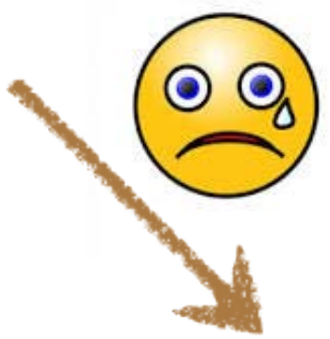
```
public class Account{
 private String thisAcnt;
 private double balance;
 public Account(String acnt, double amt) {
   this.balance=amt;
   this.Acnt=acnt;}
 public double getBalance(){
   return this.balance;}
 public void setBalance(double balance){
   this.balance=balance;}
 public void sendto(String account) {
  // send money to target account}
 public boolean deposit(double amt) {
  if (amt > 0) {
   setBalance(getBalance() + amt);
   return true;}
  else
   return false;}
 public boolean withdraw(double amt) {
    if(getBalance()>=amt && amt>0){
       setBalance(getBalance()-amt);
       return true;
    }
    else return false;
 }
 public boolean transfer(double amt, String anoAcnt) {
    double fee=0.1;  // should be fee=amt*0.1
    if(getBalance()>=amt+fee && amt>0){
      withdraw(amt+fee);
      sendto(anoAcnt);
      return true;
    }
    else return false;
 }
```

later version

```
public class Testcases
 Account a;
 protected void setUp()
    a=new Account(100.0,"user1");
 protected void tearDown()

 public void test1()
  a.transfer(50.0,"user2");
  a.withdraw(40.0);
  assertEquals(9.5,a.getBalance());

 public void test2()
  a.withdraw(40.0);
  assertEquals(56,a.getBalance();//should
be 60
  ...
```



```
public class Account{
 private String thisAcnt;
 private double balance;
 public Account(String acnt, double amt) {
   this.balance=amt;
   this.Acnt=acnt;}
 public double getBalance(){
   return this.balance;}
 public void setBalance(double balance){
   this.balance=balance;}
 public void sendto(String account) {
  // send money to target account}
 public boolean deposit(double amt) {
  if (amt > 0) {
   setBalance(getBalance() + amt);
   return true;}
  else
   return false;}
 public boolean withdraw(double amt) {
    double fee=amt*0.1;
    if(getBalance()>=amt+fee && amt>0){
       setBalance(getBalance()-amt-fee);
       return true;
    }
    else return false;
 }
 public boolean transfer(double amt, String anoAcnt) {
    double fee=0.01;  // should be fee=amt*0.1
    if(getBalance()>=amt+fee && amt>0){
      withdraw(amt+fee);
      sendto(anoAcnt);
      return true;
    }
    else return false;
 }
```

previous version

```
public class Account{
 private String thisAcnt;
 private double balance;
 public Account(String acnt, double amt) {
   this.balance=amt;
   this.Acnt=acnt;}
 public double getBalance(){
   return this.balance;}
 public void setBalance(double balance){
   this.balance=balance;}
 public void sendto(String account) {
  // send money to target account}
 public boolean deposit(double amt) {
  if (amt > 0) {
   setBalance(getBalance() + amt);
   return true;}
  else
   return false;}
 public boolean withdraw(double amt) {
    if(getBalance()>=amt && amt>0){
       setBalance(getBalance()-amt);
       return true;
    }
    else return false;
 }
 public boolean transfer(double amt, String anoAcnt) {
    double fee=0.1;  // should be fee=amt*0.1
    if(getBalance()>=amt+fee && amt>0){
      withdraw(amt+fee);
      sendto(anoAcnt);
      return true;
    }
    else return false;
 }
```

later version

```
public class Testcases
 Account a;
 protected void setUp()
    a=new Account(100.0,"user1");
 protected void tearDown()

 public void test1()
  a.transfer(50.0,"user2");
  a.withdraw(40.0);
  assertEquals(9.5,a.getBalance());

 public void test2()
  a.withdraw(40.0);
  assertEquals(56,a.ge alance();//should
be 60
  ...
```

Who knows?

An obsolete test?

A bug in the source code?

```
public class Account{
 private String thisAcnt;
 private doubl
 public Accou
   this.balance=
   this.Acnt=ac
 public double
   return this.b
 public void setBa
   this.balance=balance;}
 public void sendto(String account) {
  // send mo
 public boo
  if (amt >
   setBalan
   return t
  else
   return fa
 public boolean
   double fee=amt*0.1;
   if(getBalance()>=amt+fee && amt>0){
     setBalance(getBalan ()-amt-fee);
     return true;
   }
   else return false;
 }
 public boolean transfer(double amt, String anoAcnt) {
   double fee=0.01;  // sh ul be fee=amt*0.1
   if(getBalance()>=amt+f e && amt>0){
     withdraw(amt+fee);
     sendto(anoAcnt);
     return true;
   }
   else return false;
 }
```

previous version

```
public class Account{
 private String thisAcnt;
                                    amt) {
                                    nce){
 public void sendto(String account) {
                                    unt}
                                    amt) {
                                    t);
                                    uble amt) {
   if(getBalance()>=a          & amt>0){
     setBalance(getB          ce()-amt);
     return true;
   }
   else return false;
 }
 public boolean transfer(double amt, String anoAcnt) {
   double fee=0.1;  // s ould be fee=amt*0.1
   if(getBalance()>=am  fee && amt>0){
     withdraw(amt+fee);
     sendto(anoAcnt);
     return true;
   }
   else return false;
 }
```

later version

Problem Description

- Given a failing execution, is it caused by a bug in the source code or an obsolete test case?

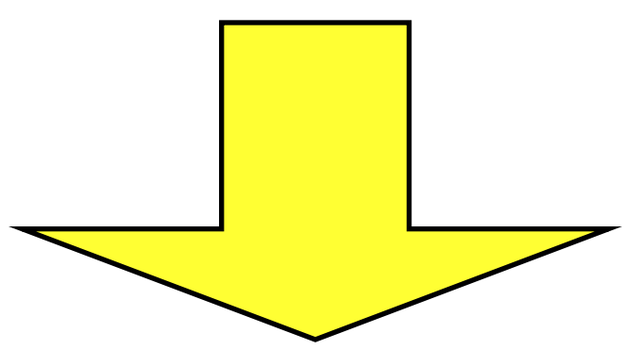
Why it is important?

- Without knowing the cause of a failure, how to decide whether repairing a test or debugging in the source code?
  - Test repair
    - repairing broken tests rather than removing or ignoring obsolete test cases
    - typical works, including [Galli:ICSM04],[Daniel:ASE09],[Daniel:ISSTA10], etc
  - Debugging
    - identifying the locations of faults and fixing the faults
    - typical works, including [Jones: ASE06],[Liblit: PLDI03],[Weimer:ICSE09],[Kim: ICSE13], etc

Our Approach

Basic Idea

- Classifying the cause (i.e., buggy code/obsolete test code) of a regression test failure



- Learning a classifier based on the features related to failures

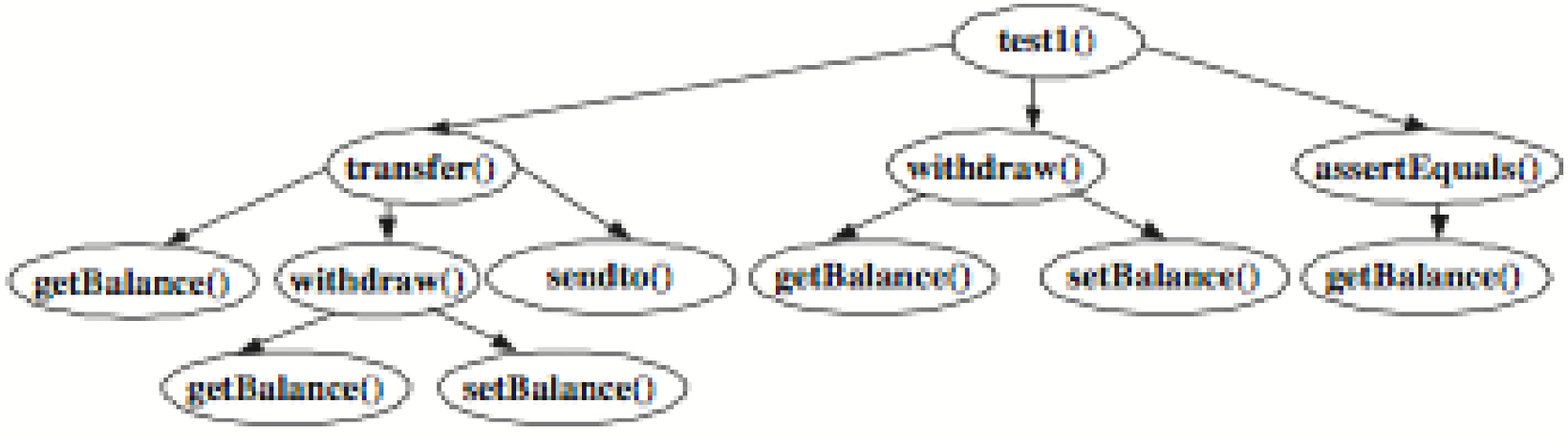
Basic Process

- Collect failure-inducing tests
  - In practice, these tests can be stored in the software repository
  - These tests are taken as training instances in building a classifier
- Determine feature values and failure causes
  - What are the features that may contribute to classifying the cause of a regression failure?
  - Using some (static) tools to acquire the feature values
- Train a classifier
  - Best-first Decision Tree Algorithm

Features in a classifier

- Features that are possible related to the failure of a regression test failure
- Three categories:
  - Complexity Feature
  - Change Feature
  - Testing Feature

Complexity Feature



- How complexity the interaction between the test and the software under test is?
  - Maximum depth of the call graph
  - Number of methods called in the graph

Change Feature

- The change between the current version and its previous version of the software under test
  - File Change, the ratio of modification on the files containing the methods called (in)directly by the failure inducing test.

$$FlieChange(t) = \frac{\sum_{\forall f \in F(t)} Change(f_0, f_1)}{\sum_{\forall f \in F(t)} maximum\{|f_0|, |f_1|\}}$$

Testing Feature

- Testing results of all the executed tests
  - Type of failure:
    - Count of plausible nodes in the call graph
    - Existence of highly fault-prone node in the call graph
    - Product Innocence

Basic Process

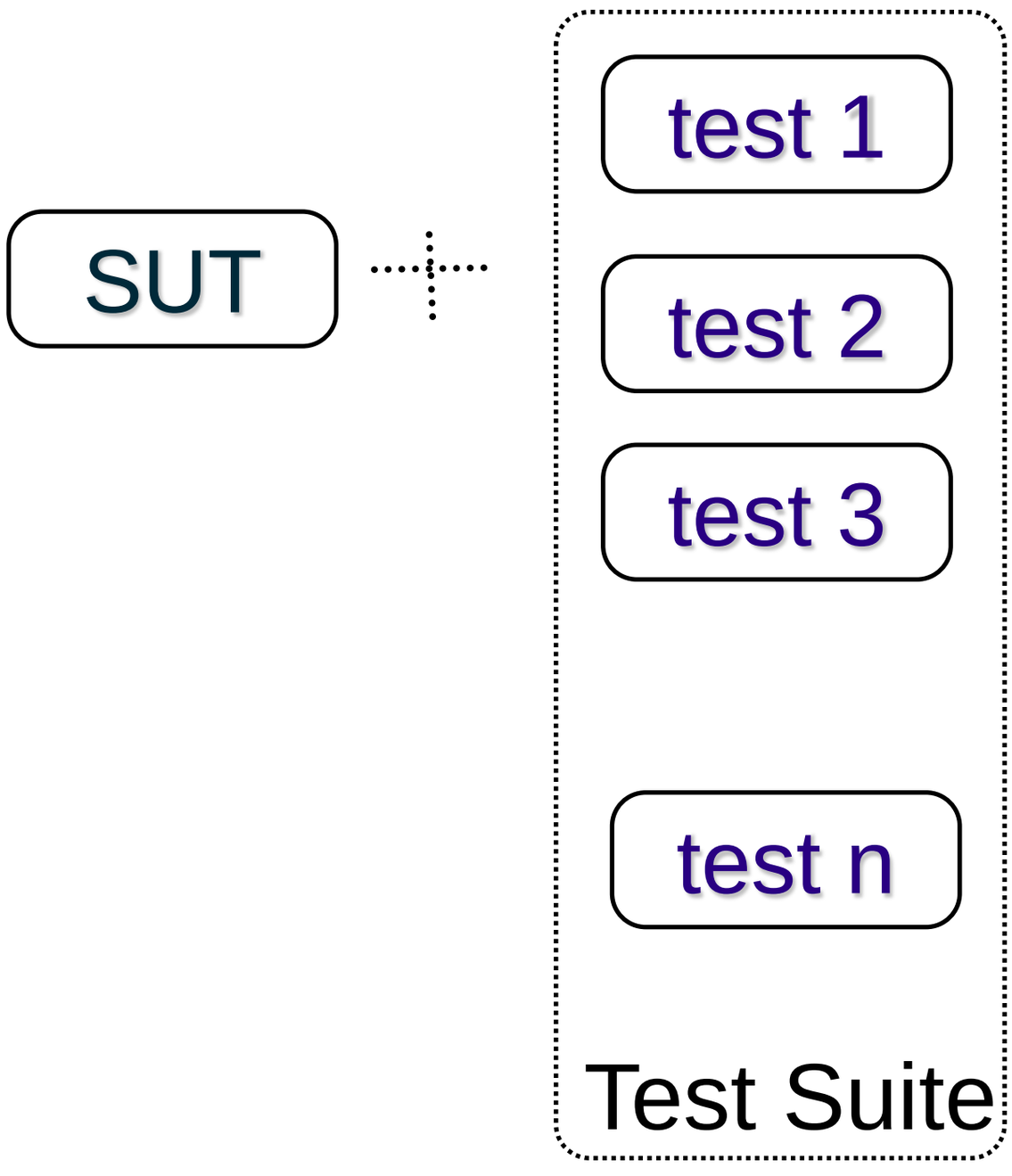
- Collect failure-inducing tests
- Determine feature values and failure causes
- Train a classifier

Evaluation

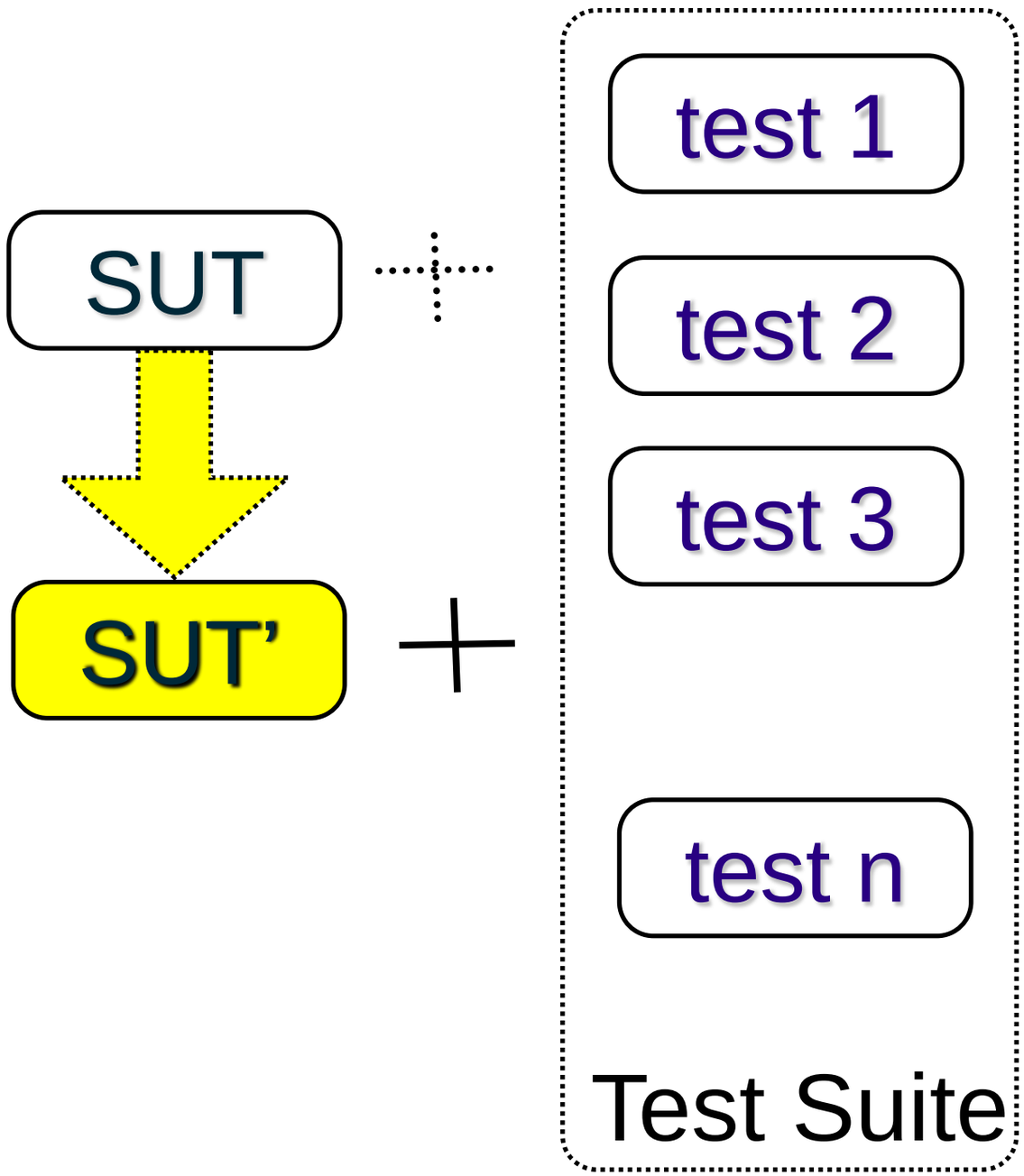
Research Questions

- RQ1: Is our approach effective in classifying the cause of regression test failures when being applied within one version of a program?
- RQ2: Is our approach effective in classifying the cause of regression test failures when being applied between two versions of a program?
- RQ3: Is our approach effective in classifying the cause of regression test failures when being applied across different programs?

Experimental Design

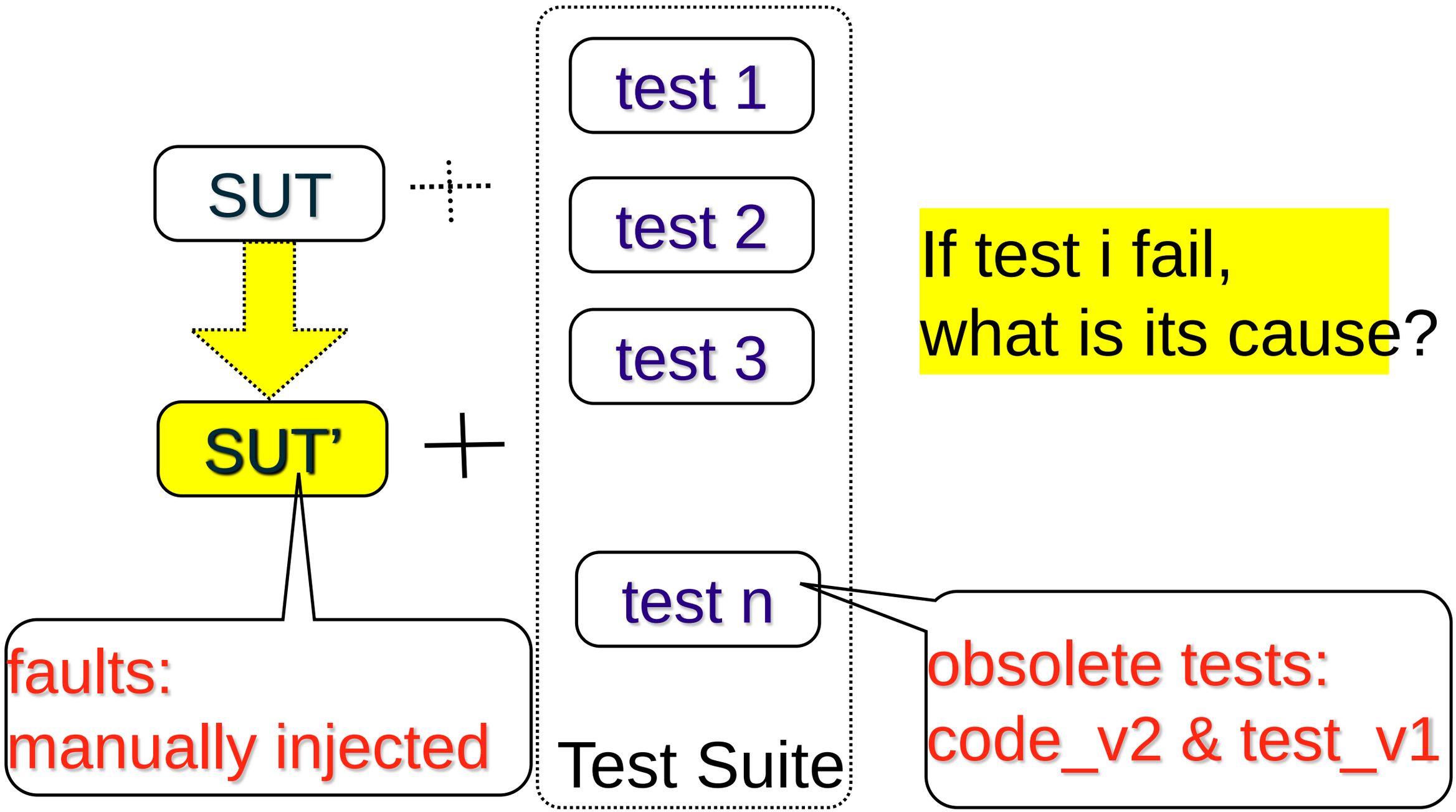


Experimental Design



If test i fail, what is its cause?

Experimental Design



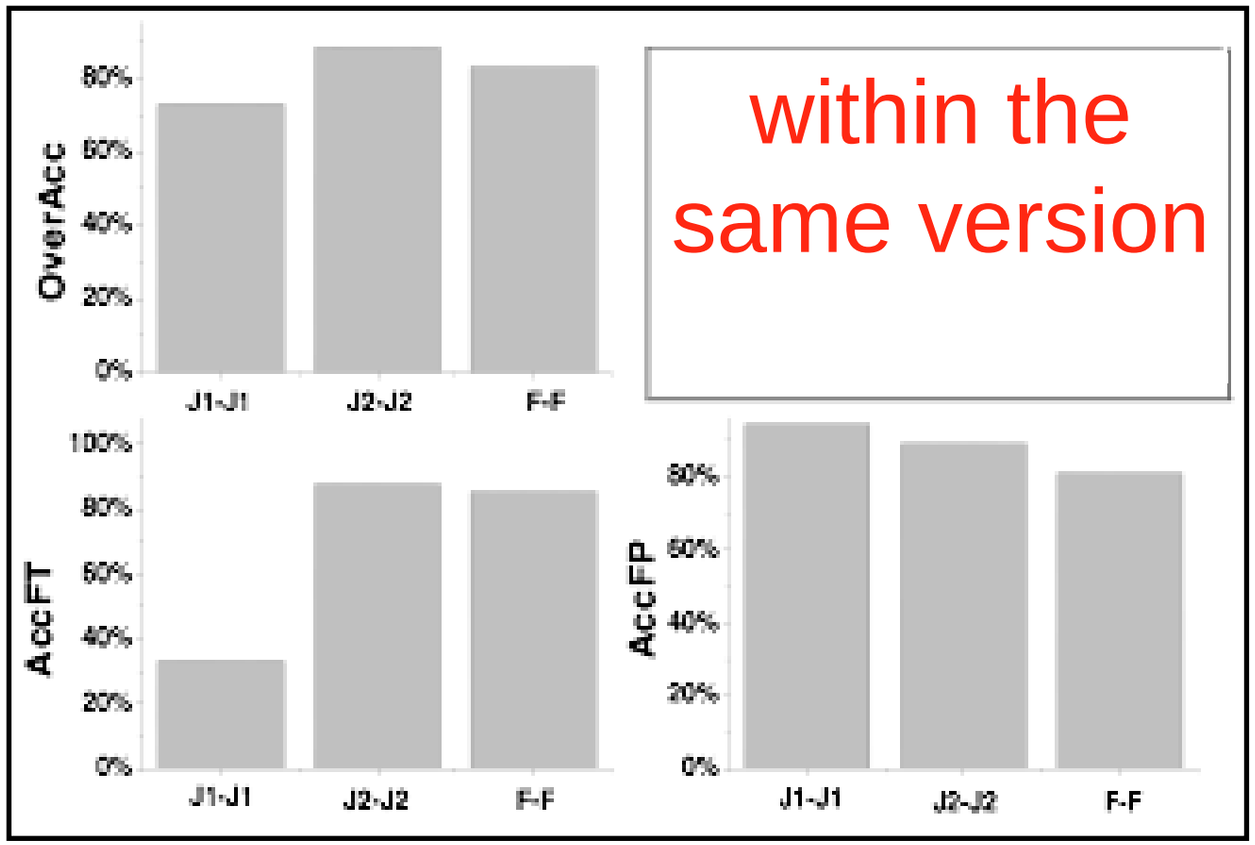
If test i fail, what is its cause?

Experimental Design

Program	Product Code				Test Code			
	#Files	#LOC	#Classes	#Methods	#Files	#LOC	#Classes	#Methods
Jfreechart 1.0.0	463	68,761	465	6,038	273	26,847	273	1,751
Jfreechart 1.0.7	538	80,927	540	7,335	356	42,052	356	2,634
Jfreechart 1.0.13	585	91,101	587	8,296	383	47,930	383	3,078
Freecol 0.10.3	578	94,031	579	6,757	85	13,022	85	493
Freecol 0.10.5	602	95,404	603	7,001	87	13,229	87	497

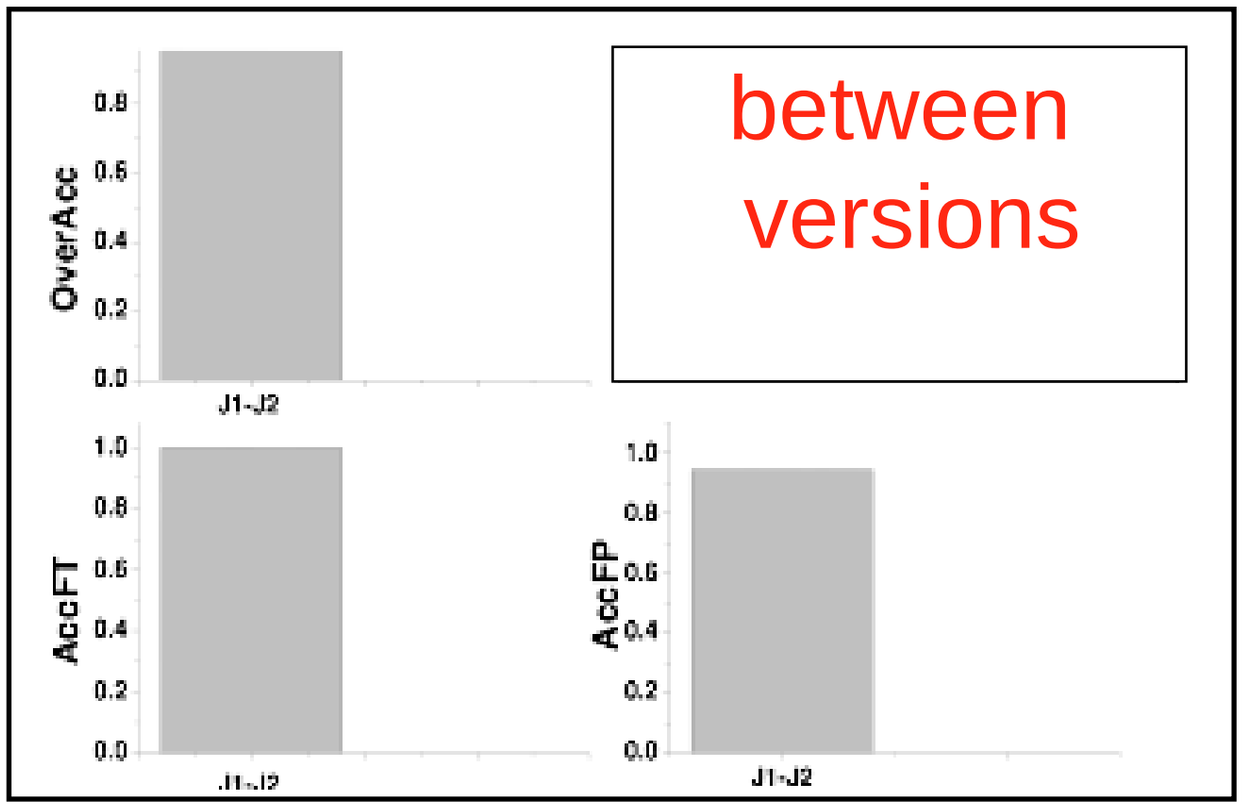
Set of instances

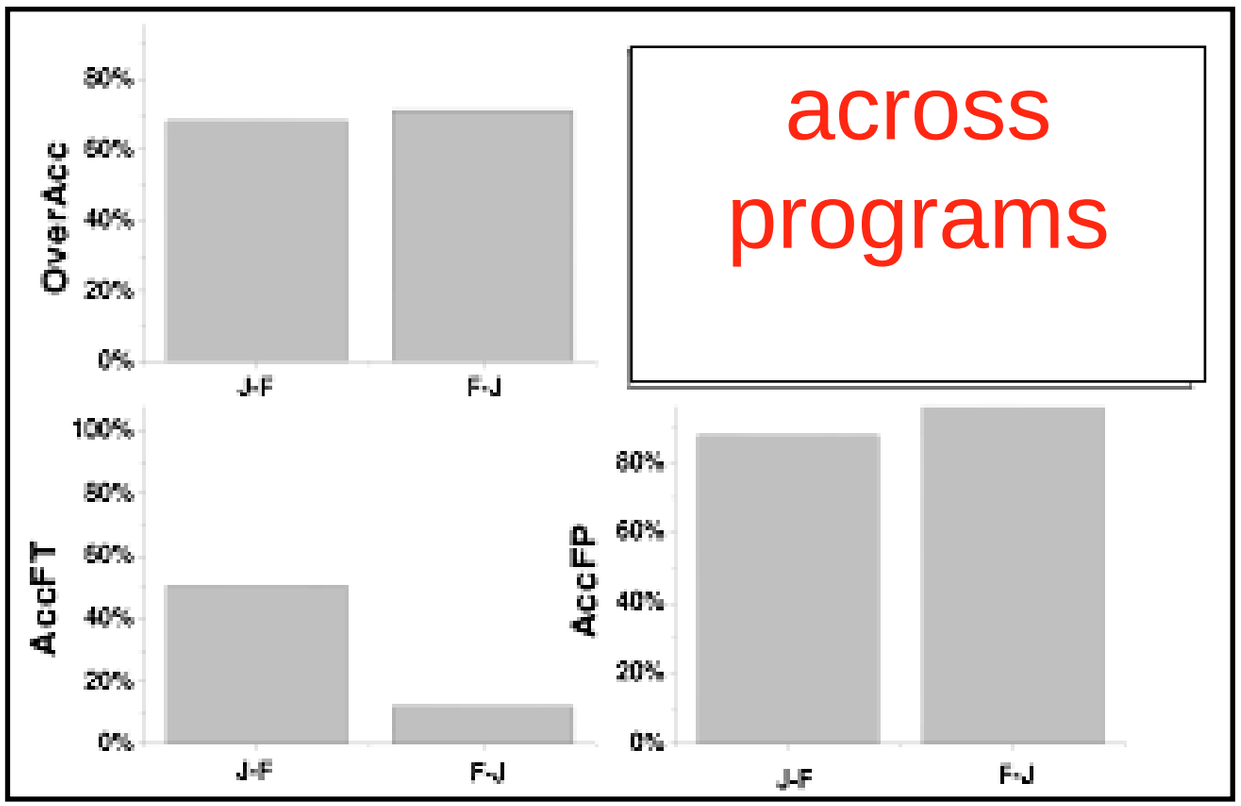
- Training instances: build a learner
- Testing instances: evaluate the learner
  - Within the same version of a program
  - Between versions of a program
  - Cross programs

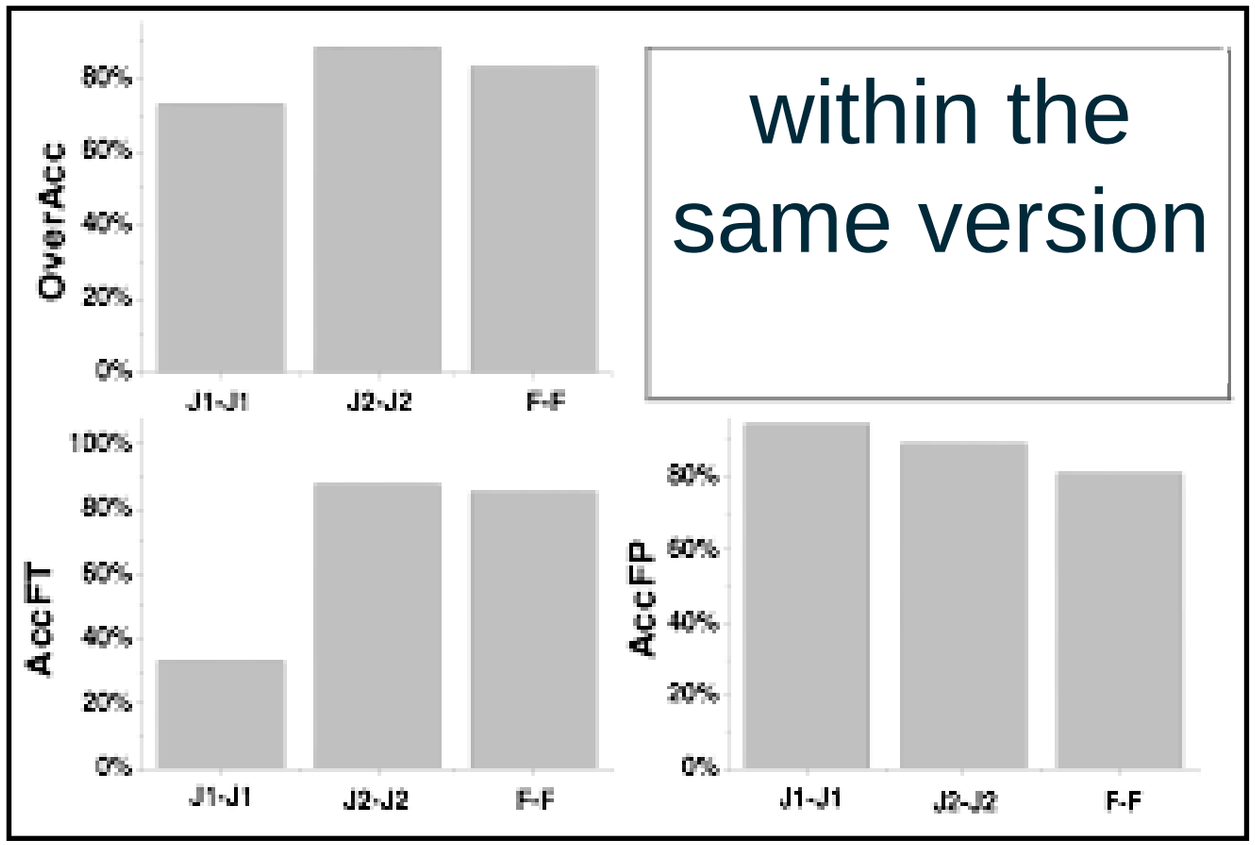


OverAcc: Overall accuracy
AccFT: Accuracy of faults in the test code
AccFP: Accuracy of faults in the product code

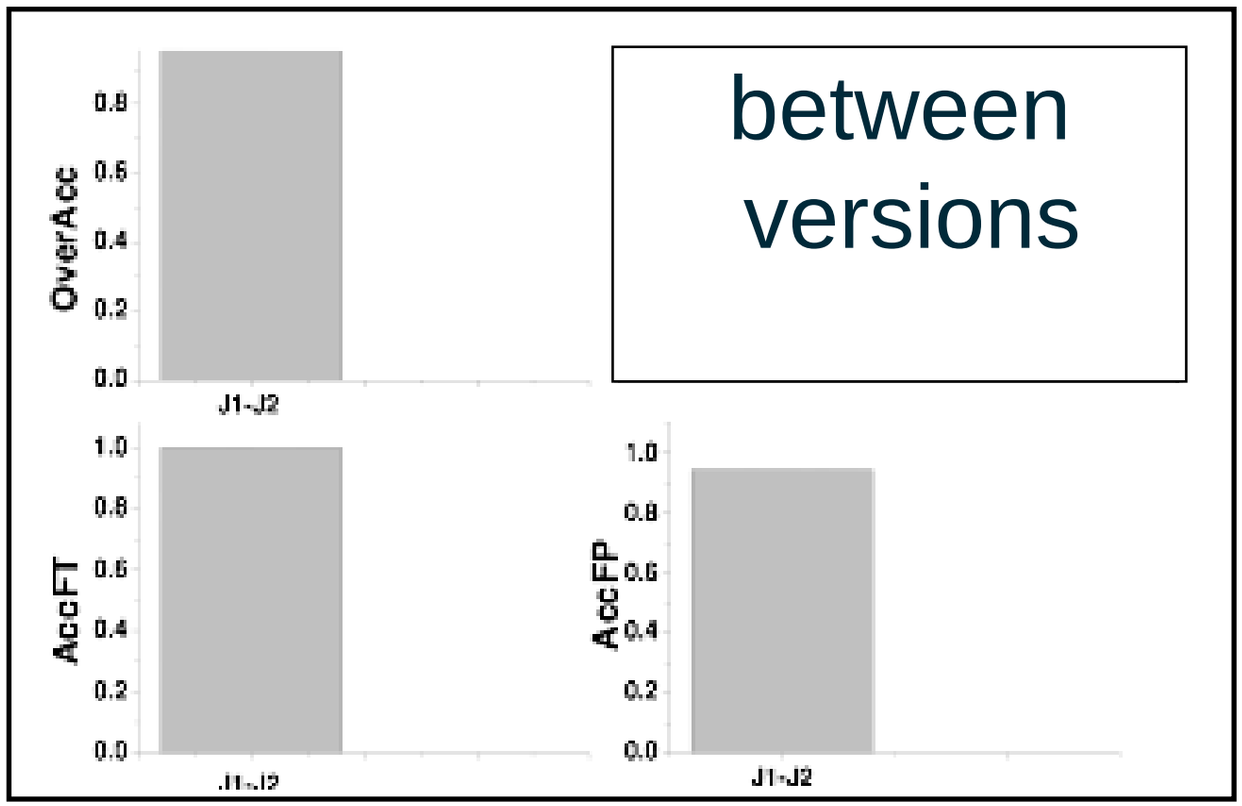
X-Y: X is used as the training instances,
Y is used as the testing instances

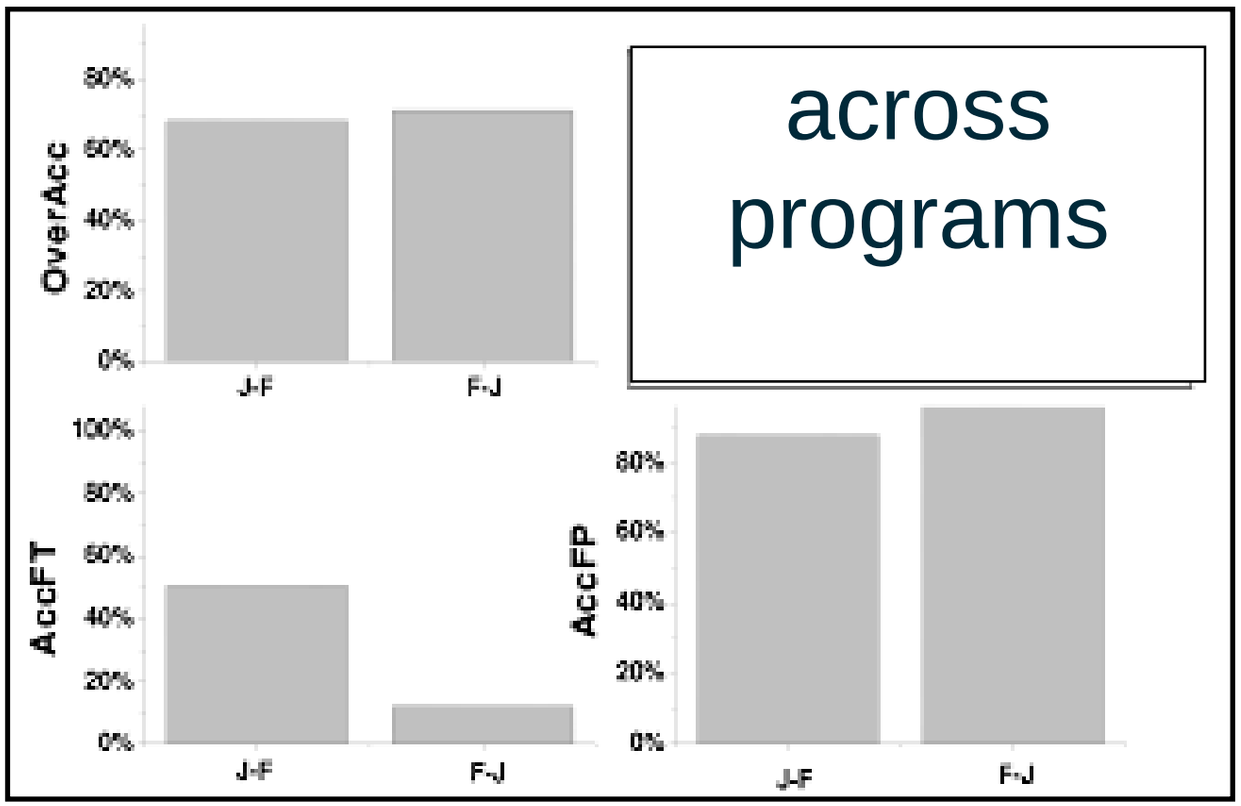






Effective when being applied within the same versions, or between versions





Take Home Message

- The first piece of research trying to classify the cause of a regression test failure as a bug in the product code or an obsolete test
- The proposed machine-learning based approach has been evaluated to be effective when being applied within the same program (including the same version and different versions)
- Preliminary research and future work

Thanks a lot!