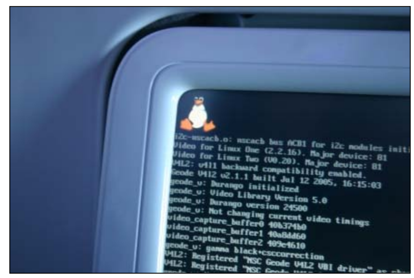
Testing of Evolving Software: Achievements, Challenges, and Promises

Mary Jean Harrold
ADVANCE Professor of Computing
College of Computing
Georgia Institute of Technology

High Cost of Software Failure

Airplane entertainment system (2008)

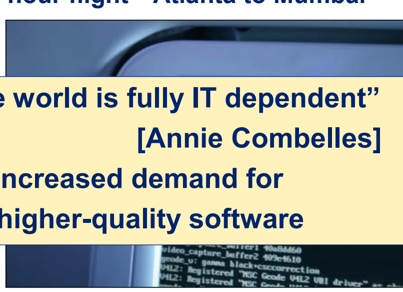
- **Failed for me and most passengers**
- **16 hour flight—Atlanta to Mumbai**



High Cost of Software Failure

Airplane entertainment system (2008)

- **Failed for me and most passengers**
- **16 hour flight—Atlanta to Mumbai**



"The world is fully IT dependent"
[Annie Combelles]

→ increased demand for higher-quality software

Resulted in Increased Research In

Software testing, verification, validation

Software Engineering
- FSE
- ESEC/FSE
- ICSE
- ICSM

Languages/ Compilers
- PLDI
- POPL
- OOPSLA
- ECOOP

Testing, Verification, Validation
- ISSTA (annual)
- ICST

Resulted in Increased Research In

Software testing, verification, validation

Uniqueness of ICST
- reporting good results in testing, verification, validation

and
- bridging research and practice

Testing, Verification, Validation
- ISSTA (annual)
- ICST

Testing Evolving Software

- **Interest**
- **Problems**
  - Achievements
  - Challenges
- **Industry—academic collaboration**

Initial Interest by Chance

- Searched for problem
- Discovered data-flow testing
- Led to
  - using data-flow information to select test cases to rerun
  - researching other regression testing problems

The Real Eye Opener

Early 1990s
- Industry/academic testing conference
- Gave invited talk on regression testing

In the audience
Roger Sherman,
Director of Test, Microsoft

After talk
- Discussed my work
- Invited me to visit

At Microsoft
- Collaborated with developers and testers
- Integrated my algorithm into MS tools
- Solved some problems, identified new problems

Collaboration With Industry

Common Problem

- Changes require rapid modification and testing for quick release (time to market pressures)
- Causing released software to have many defects

(now part of EMC) • Healthcare

Research Question

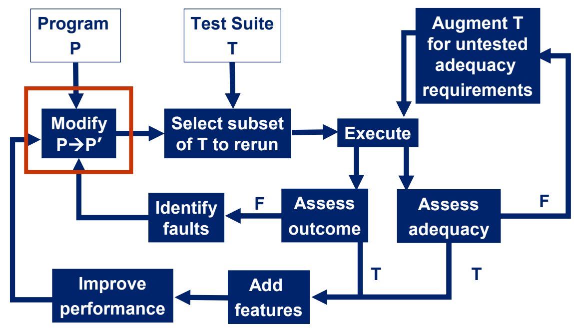
How can we test well to gain confidence in the changes in an efficient way before release of changed software?

• Tata Consultancy • Spacecraft

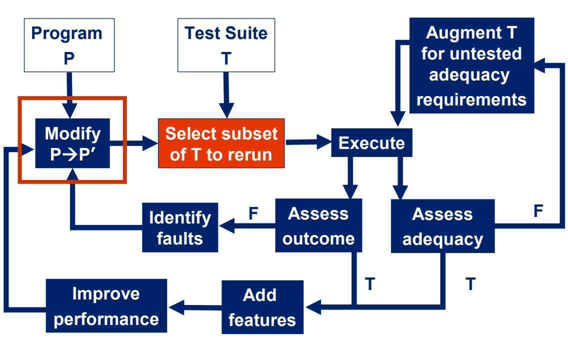
Approach

- Concentrate testing around the changes
- Automate (if possible) the regression testing process

Testing Evolving Software



Select Subset of T to Rerun



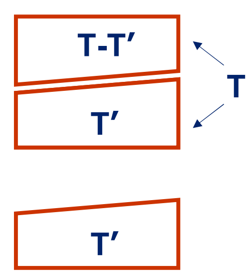
Select Subset of T to Rerun

Which test cases in T should be rerun to test P′?

Solution

Partition T into two subsets

- run one on P′
- don't run the other



General Test Selection Technique

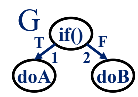
1. Construct representation G for P

P can be
- procedural
- object-oriented
- database
- Web-based
- other

G can be
- low level—e.g., code
- higher level—e.g., UML diagrams, state-transition diagrams, architectural diagrams
- other information—e.g., CVS repository

General Test Selection Technique

1. Construct representation G for P

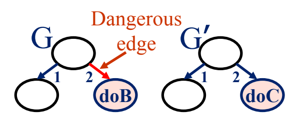


2. Associate test cases in T with entities in G

M

edge \ TC	t1	t2	t3
e1	X		
e2		X	X

3. Build G' and compare G and G' to find dangerous entities



4. Select test cases based on dangerous entities

M

edge \ TC	t1	t2	t3
e1	X		
e2		X	X

Empirical Evaluation

Goal: To determine savings in execution time

Subjects

C Program	Versons	Procedures	KLOC	Test Cases
Empire	5	766	50	1033
Java Programs	Versions	Classes	KLOC	Test Cases
Daikon	5	824	167	200
JBoss	5	2,403	~1K	639

Procedure

For each pair of versions v_i, v_{i+1}, measured

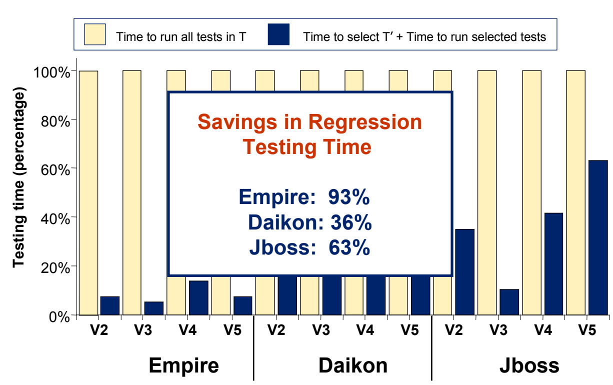
A. Time to re-run v_{i+1} on all test cases in T

B. Time to select T′ (using DejaVu) + Time to run T′ on v_{i+1}

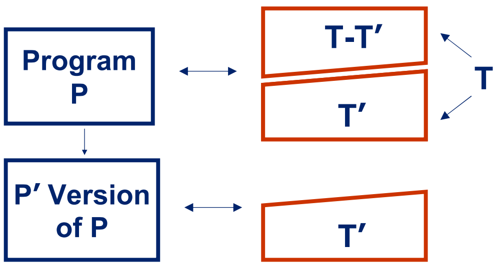
Compare times

- Save if B < A

Savings in Testing Time Using DejaVu



Select Subset of T to Rerun

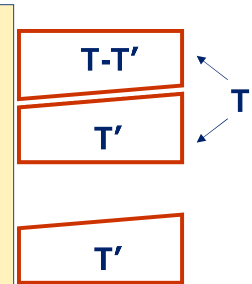


Select Subset of T to Rerun

What if
- T′ has too many test cases for allotted time?
- want to run most important test cases in T′ first?

Solution

Order (prioritize) T′
- find faults earlier
- get coverage earlier
- etc.

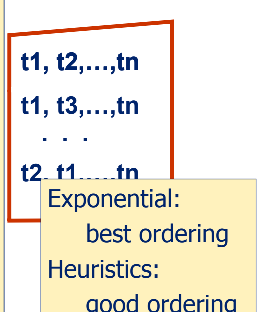


Select Subset of T to Rerun

What if
- T′ has too many test cases for allotted time?
- want to run most important test cases in T′ first?

Solution

Order (prioritize) T′
- find faults earlier
- get coverage earlier
- etc.



Achievements: Research

- Application to
  - models of the system
  - kinds of programs
  - programming languages
- Representations using
  - types of program information
- Empirical evidence of
  - effectiveness
  - efficiency

  } many examples

Achievements: Industry

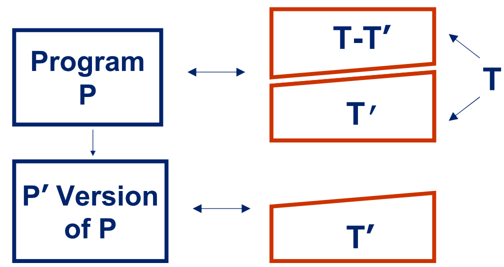
In 2002 OOPSLA talk, Bill Gates

- in-house testing tool—selects and prioritizes regression tests
- “the impact had been very dramatic."

In 2004 talk at Georgia Tech, Jim Gray

- prioritizing regression tests being used throughout Microsoft with significant impact

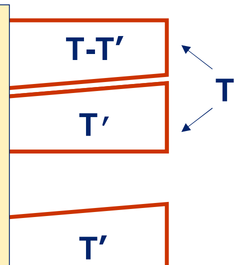
Challenges: Assumptions



Challenges: Assumptions

Selection/prioritization algorithms assume

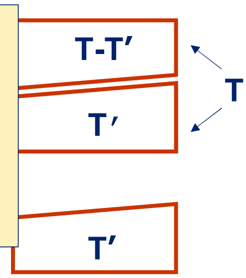
1. obsolete test cases removed from T
2. deterministic behavior of P, P′
3. non-distributed development of P, P′



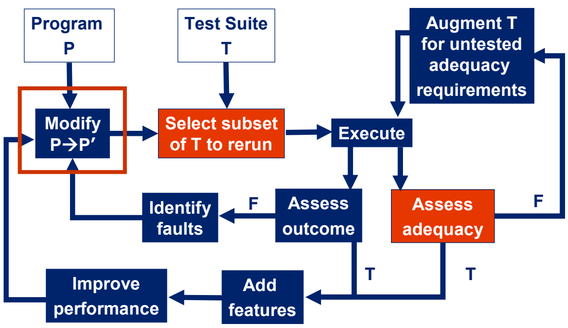
Challenges: Process

Regression testing

1. performed at system level
2. performed manually
3. part of a larger process that is difficult to change



Assess Adequacy



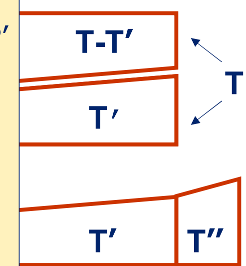
Assess Adequacy

How well do T, T′,T″ or any test suites exercise P′ with respect to changes?

Solution

Change-impact criteria

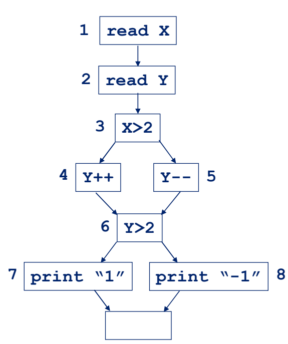
- to assess existing test suite
- augment test suite



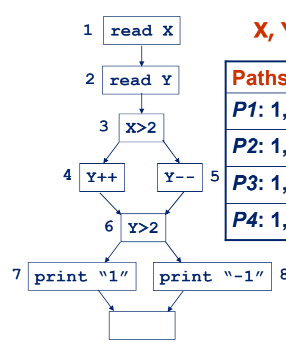
Program and Control-flow Graph (CFG)

```
Program Example
1.  read X
2.  read Y
3.  if (X > 2)
4.     Y ++
    else
5.     Y --

6.  if (Y > 2)
7.      print "1"
    else
8.     print "-1"
```



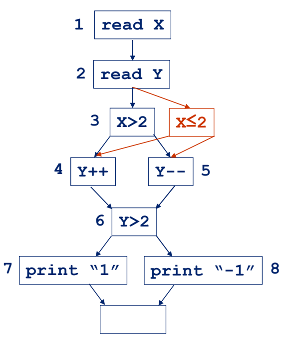
Program and Control-flow Graph (CFG)



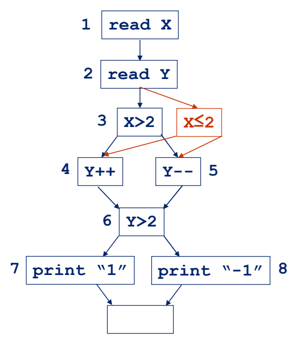
X, Y in [1,10] → 100 test cases

Paths	Conditions	Tests
P1: 1,2,3,4,6,7	X>2, Y>1	72
P2: 1,2,3,4,6,8	X>2, Y≤1	8
P3: 1,2,3,5,6,7	X≤2, Y>3	14
P4: 1,2,3,5,6,8	X≤2, Y≤3	6

Program and Control-flow Graph (CFG)



Assess Adequacy



Want

- inputs that show difference between P and P′
- criteria that require such test cases

→ Change-impact criteria

Change-Impact Criteria

Criteria for change-impact propagation (PIE model [Voas] for change propagation)

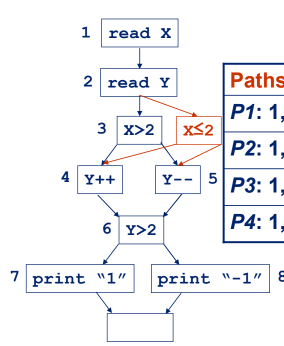
- **Execution of the change**
- **Infection of the state after change**
- **Propagation of state to output where it can be observed**

Want

- inputs that show difference between P and P′
- criteria that require such test cases

→ Change-impact criteria

Change-Impact Criteria



Y must be 2 or 3

Paths	Conditions	Tests
***P1*: 1,2,3,4,6,7**	**X>2, Y>1**	**16/72**
***P2*: 1,2,3,4,6,8**	**X>2, Y≤1**	**0/8**
***P3*: 1,2,3,5,6,7**	**X≤2, Y>3**	**0/14**
***P4*: 1,2,3,5,6,8**	**X≤2, Y≤3**	**4/6**

20/100 test cases propagate difference

Change-Impact Criteria

Criteria for change-impact propagation (PIE model for change propagation)

- Execution of the change
- Infection of the state after change
- Propagation of state to output where it can be observed

20/100 test cases propagate difference

Criterion

- Random

  20% chance of finding fault

Change-Impact Criteria

Criteria for change-impact propagation (PIE model for change propagation)

- **Execution of the change**
- Infection of the state after change
- Propagation of state to output where it can be observed

20/100 test cases propagate difference

Coverage Criteria

- Change or statement

  1 → 20%

  >1 → increase chances but not because of criterion

  Generally, better than random

Change-Impact Criteria

Criteria for change-impact propagation (PIE model for change propagation)

- **Execution of the change**
- Infection of the state after change
- **Propagation of state** to output where it can be observed

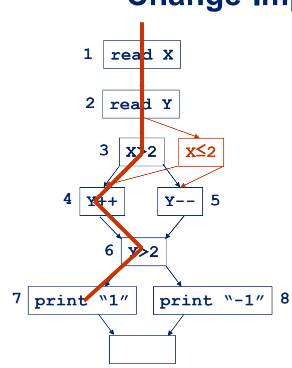
20/100 test cases propagate difference

Coverage Criteria

- DU-Pairs

(1,(3,4),X), (2,4,Y)	*P1, P2*	
(1,(3,5),X), (2,5,Y)	*P3, P4*	4/6
(2,(6,7),Y)	*P1, P3*	
(2,(6,8),Y)	*P2, P4*	
(4,(6,7),Y)	*P1*	16/72
(4,(6,8),Y)	*P2*	0/8

Change-Impact Criteria



20/100 test cases propagate difference

Coverage Criteria

- Dependence chains (all paths)

***P1*: 1,2,3,4,6,7**	**16/72**
***P2*: 1,2,3,4,6,8**	**0/8**
***P3*: 1,2,3,5,6,7**	**0/14**
***P4*: 1,2,3,5,6,8**	**4/6**

Change-Impact Criteria

Criteria for change-impact propagation (PIE model for change propagation)

- **Execution of the change**
- Infection of the state after change
- **Propagation of state to a point where it can be observed**

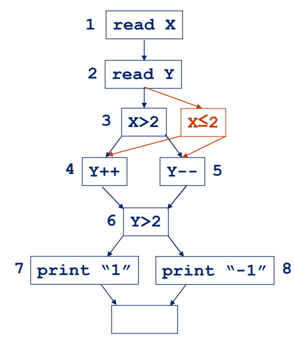
20/100 test cases propagate difference

Coverage Criteria

- Dependence chains (all paths)

***P1*: 1,2,3,4,6,7**	
***P2*: 1,2,3,4,6,8**	
***P3*: 1,2,3,5,6,7**	
***P4*: 1,2,3,5,6,8**	

Change-Impact Criteria



S3

Infection:

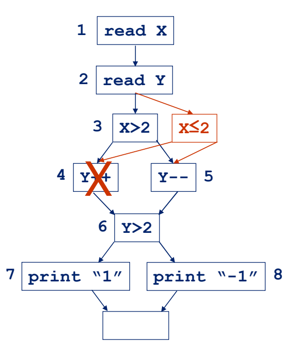
Path condition in P after S3 and path condition in P′ after S3′ differ

Condition for infection:

(X≤2) and not (X>2)

→ any value of X shows difference

Change-Impact Criteria



S4

Infection:

Value of state after execution of S4 in P and S4′ in P′ must differ

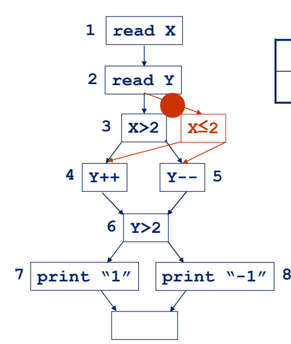
Condition for infection:

- S4 in P, Y=Y+1
- corresponding location in P′, Y=Y

→ Y≠Y+1

→ any value of Y

Change-Impact Criteria



P **Infection** P′

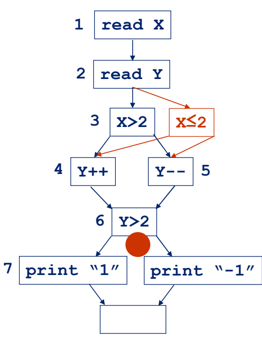
PC	SS(n)
true	X_0 Y_0

PC′	SS′(n)
true	X_0 Y_0

PC—path condition
SS—symbolic state

Perform symbolic execution from before change to get conditions

Change-Impact Criteria



Propagation

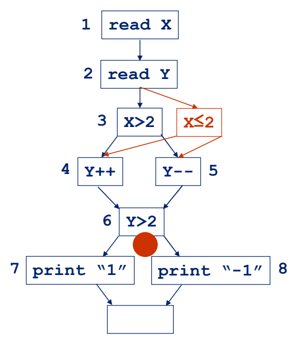
P

PC	SS(n)
true	X_0 Y_0
$X_0 > 2$ $X_0 \le 2$	X_0 Y_0
$X_0 > 2$ or $X_0 \le 2$	X_0 $Y_0 + 1$ X_0 $Y_0 - 1$
. . .	. . .

P′

PC′	SS′(n)
true	X_0 Y_0
$X_0 \le 2$ $X_0 > 2$	X_0 Y_0
$X_0 \le 2$ or $X_0 > 2$	X_0 $Y_0 + 1$ X_0 $Y_0 - 1$
. . .	. . .

Change-Impact Criteria



Propagation

Resulting conditions that show differences

C1: $Y_0+1>2$ *and* $Y_0-1 \le 2$

Y_0 in {2,3}

C2: $Y_0+1 \le 2$ *and* $Y_0-1 > 2$

Y_0 in {}

Change-Impact Criteria

Propagation

But

- symbolic execution on the entire program is expensive
- may not scale to large programs
- etc.

New technique has two ways to improve efficiency

1. Performs Partial Symbolic Execution

Beginning immediately before change

- computes conditions in terms of relative variables immediately before change
- avoids symbolic execution from beginning of program to chang

Don't need to solve conditions—can still monitor for their satisfaction

2. Performs Partial Symbolic Execution

For some distance instead of to output statements

- computes conditions on states at intermediate points (i.e., distances)
- bounds depth using slicing-like dependences, avoids symbolic execution to outputs

Greater distances improve confidence in propagation to output

Change-Impact Criteria

Criteria for change-impact propagation (PIE model for change propagation)

- **Execution of the change**
- **Infection of the state after change**
- **Propagation of state to a point where it can be observed**

Propagation

Resulting conditions that show differences

C1: $Y_0+1>2$ *and* $Y_0-1 \le 2$
Y_0 in {2,3}

C2: $Y_0+1 \le 2$ *and* $Y_0-1 > 2$
Y_0 in {}

Insert Probes to Record Coverage

```
Program Example'
1.  read X
2.  read Y
3.  if (X ≤ 2)
4.    Y ++
    else
5.    Y --

6.  if (Y > 2)
7.     print "1"
    else
8.    print "-1"
```

To record adequacy (coverage of conditions)

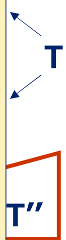
Instrument modified program so that probes check for satisfaction of condition before change (e.g., before S3′)

Achievements: Research

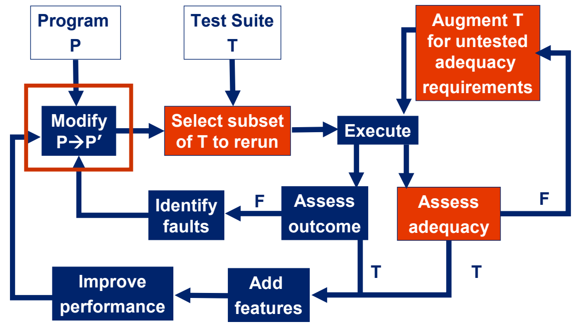
- Change test criteria
  - based on different program entities
  - varying strength and effectiveness
- New change test criteria (early results)
  - based on entities *and* state
  - show promise for improving adequacy of test suite around changes
  - can be performed on large programs—only area around change evaluated
  - empirical evaluation encouraging

Challenges

1. more research—e.g.,
   - handling multiple changes
   - tracking actual propagation
   - handling false positives
2. more engineering—e.g.,
   - efficient partial symbolic evaluation
   - effective condition representation
   - various analyses
3. more experimentation—e.g.,
   - determining good distance



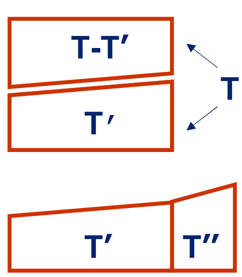
Augment T for … Requirements



Augment T for … Requirements

How can we get test cases to satisfy unsatisfied conditions?

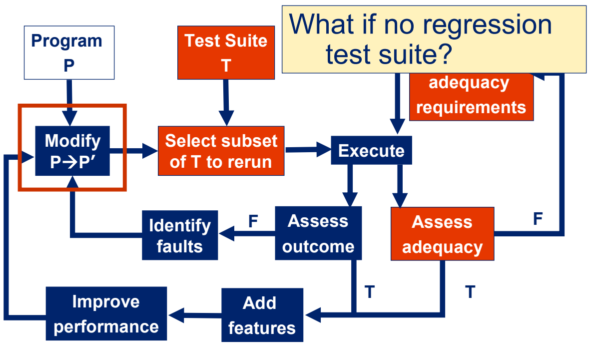
Solution
Provide assistance for generating test cases for these conditions



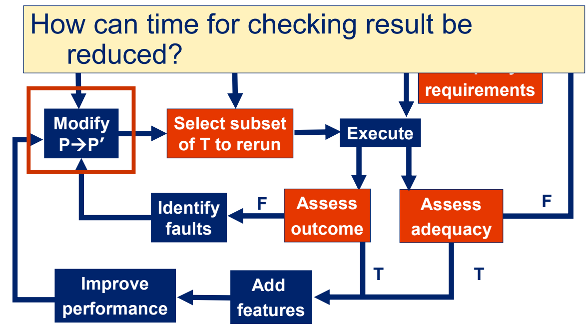
Challenges

- Developers use unsatisfied conditions to create new test cases
- Automatically generate test cases to satisfy conditions
  - use regression test suite and apply existing techniques
  - ensure that the techniques are efficient for large programs
  - accommodate large numbers of infeasible paths

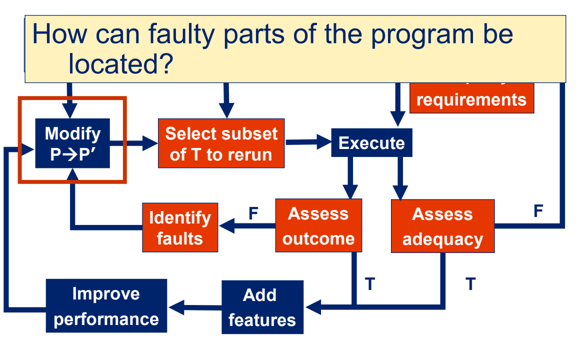
Create Regression Test Suite



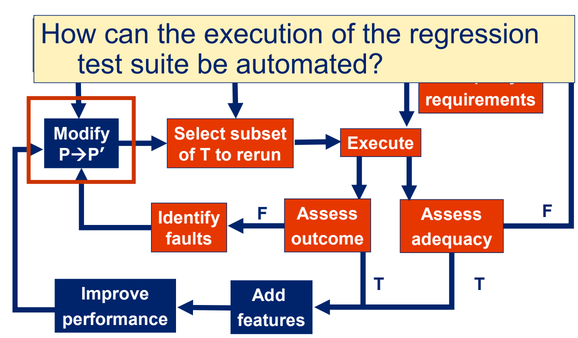
Assess Outcome



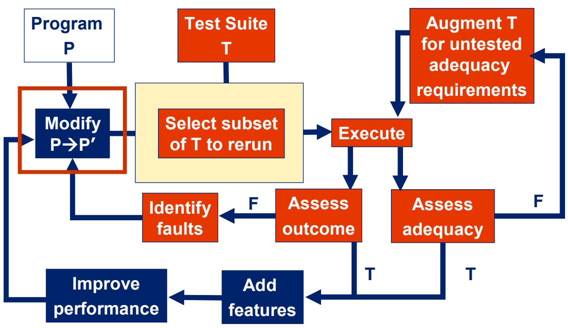
Identify Faults



Automate Execution



Industry-Academic Collaboration



Lessons Learned

To solve important problems—have impact

- interact with industry
- many ideas come from collaborations

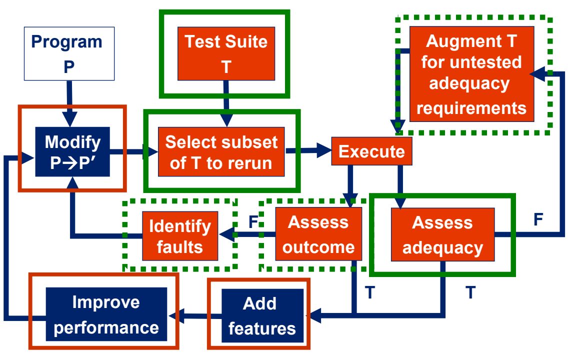
To transfer results to industry

- show effectiveness on real systems (their systems)
- requires extensive change to process

To evaluate research in practice

- integrate prototype into industrial environment
- make prototype usable for developers
- identify internal champion to support evaluation

Promise: What We Can Expect



Questions?