Testing Evolving Software
Current Practice and Future Promise

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

Aristotle Research Group

Georgia Tech College of Computing

Problems for Evolving Software

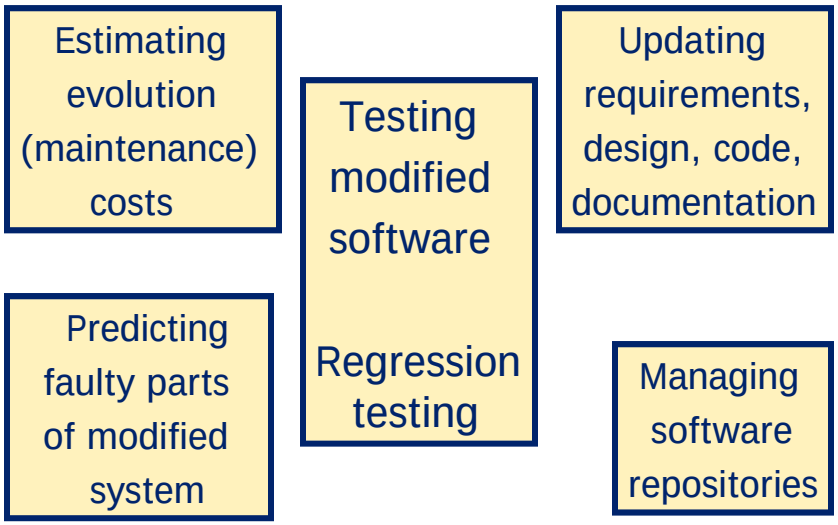
Estimating evolution (maintenance) costs

Updating requirements, design, code, documentation

Predicting faulty parts of modified system

Managing software repositories

Problems for Evolving Software



Testing Evolving Software

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

Collaboration With Industry

Common Problem

- Changes require rapid modification and testing for quick release
- Causing released software to have many defects
- Lucent Technologies

Research Question

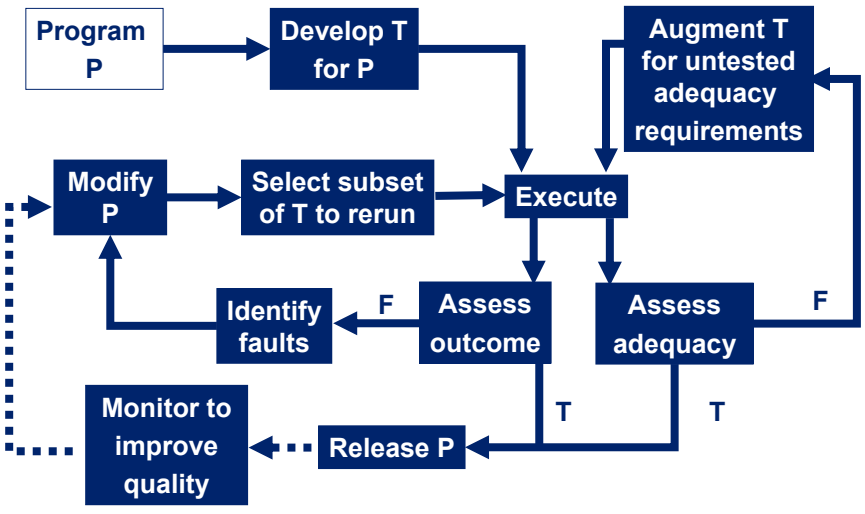
How can we test well to gain confidence in the changes in an efficient way before release of changed software?

- Reflective Corporation

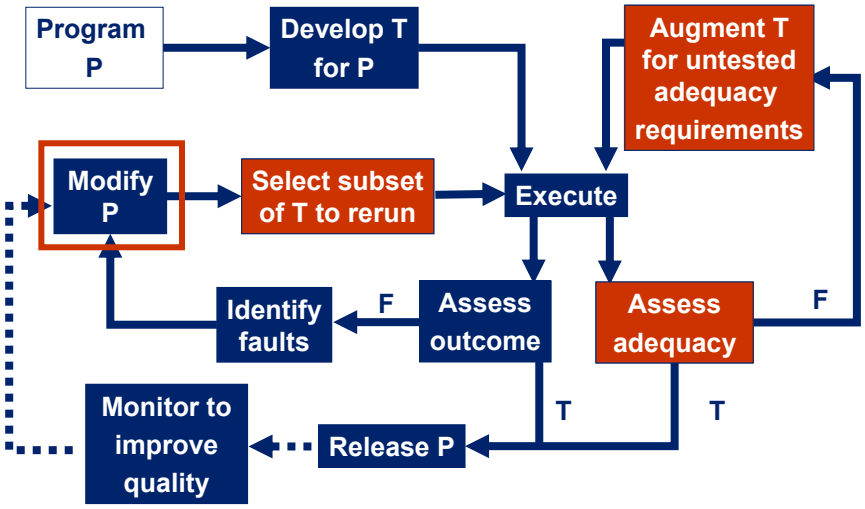
Approach

- Concentrate testing around the changes
- Automate the regression testing process

Testing Evolving Software



Testing Evolving Software



Testing Evolving Software

Select subset of T to rerun

- Present problem

Assess adequacy

- Overview current status, achievements

Augment T for untested adequacy requirements

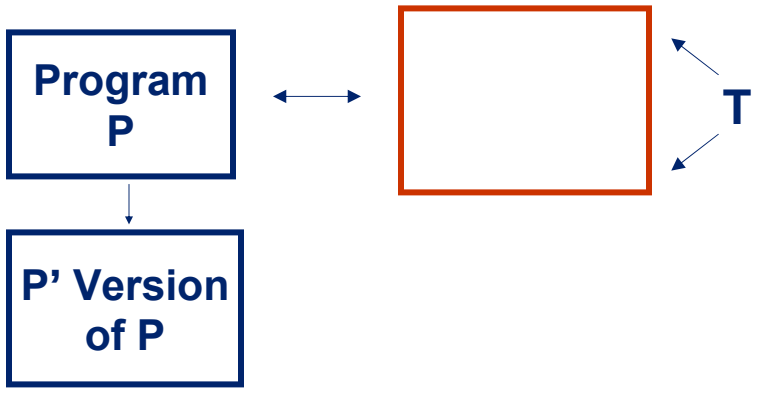
- Discuss challenges and open issues

Select Subset of T to Rerun

Assess adequacy

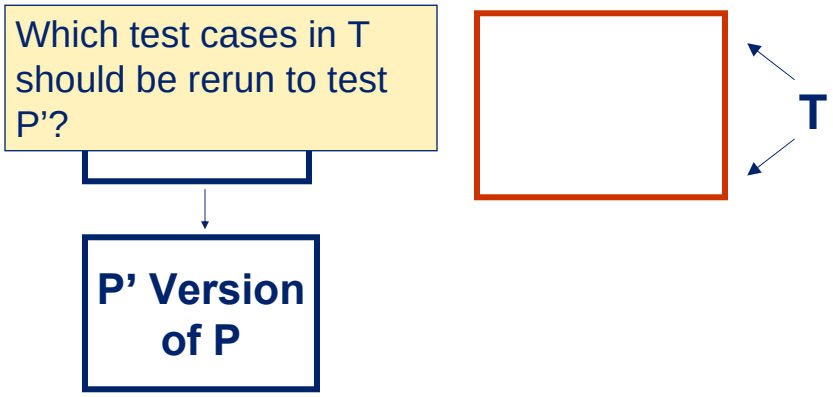
Augment T for untested adequacy requirements

Select Subset of T to Rerun



Select Subset of T to Rerun

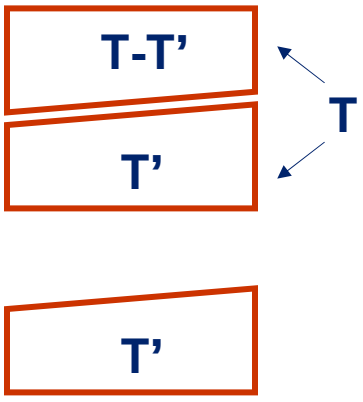
Which test cases in T should be rerun to test P'?



Select Subset of T to Rerun

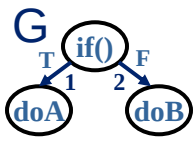
Which test cases in T should be rerun to test P'?

Solution
Partition T into two subsets
- run one on P'
- don't run the other



Select Subset of T to Rerun

1. Construct representation G for P

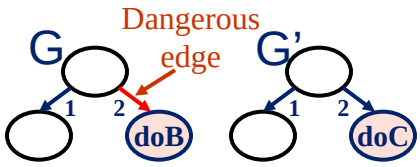


2. Associate test cases in T with entities in G

M

edge \ TC	t1	t2	t3
e1	X		
e2		X	X

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

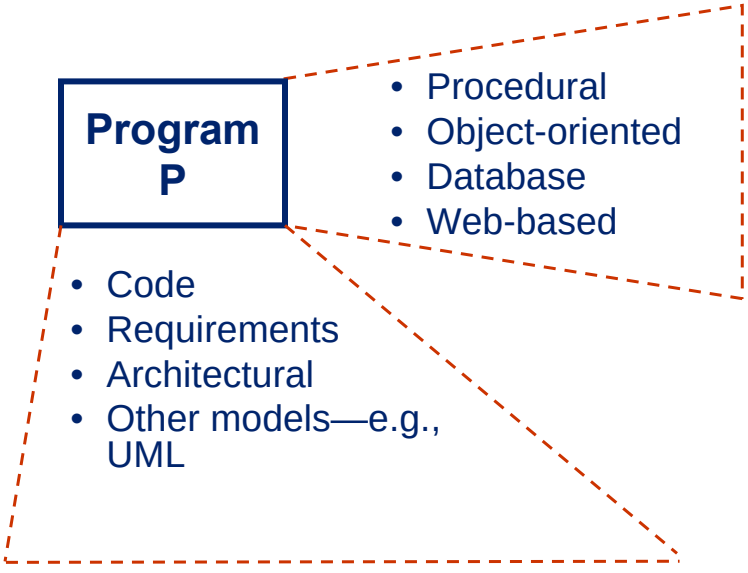


4. Select test cases based on dangerous entities

M

edge \ TC	t1	t2	t3
e1	X		
e2		X	X

Select Subset of T to Rerun



Empirical Studies

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}, measure

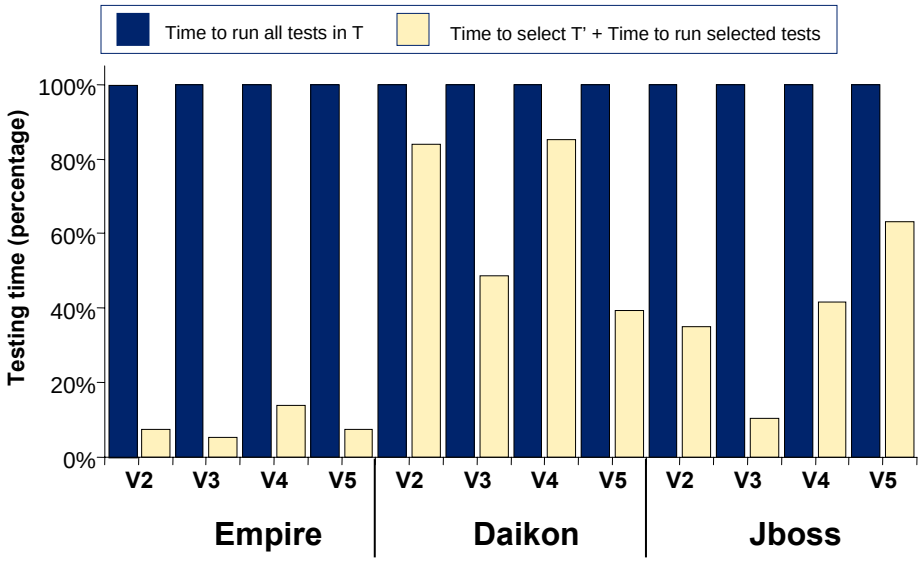
A. Time to re-run v_{i+1} on all test cases in T

B. Time to select T' + Time to run T' on v_{i+1}

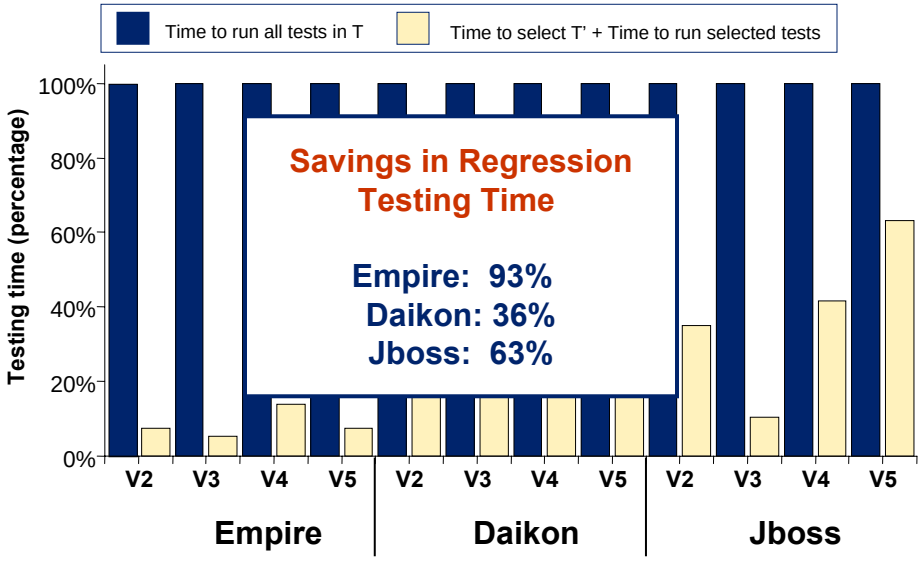
Compare times

- Save if B < A

Savings in Testing Time Using DejaVOO

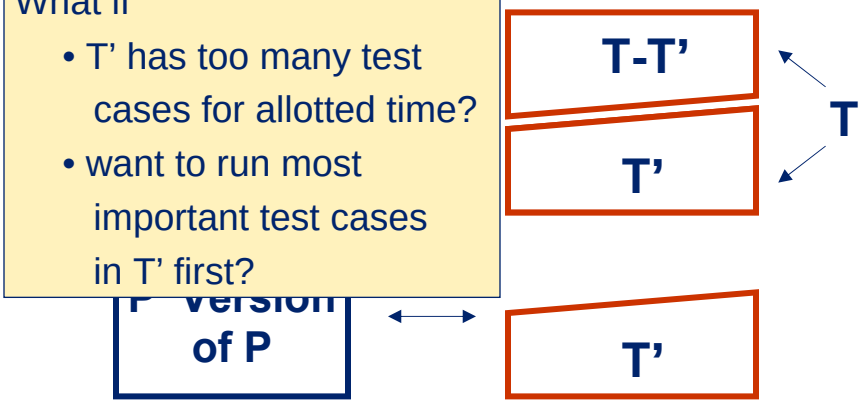


Savings in Testing Time Using DejaVOO



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?

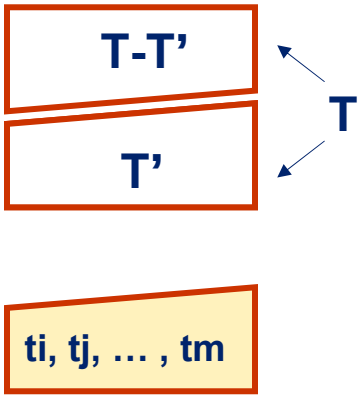


Select Subset of T to Rerun

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 different models of the system
- Empirical evidence of effectiveness on different kinds of programs written in various languages
- Evidence of the effectiveness of techniques that use simple program information

Achievements: Commercialization

Google Testar

- Selective testing tool for Java
- Works with JUnit
- Records coverage data about JUnit tests by instrumenting bytecode
- Computes, stores, and compares checksums to identify changes
- Also computes and reports coverage of methods

Challenges

Good selection/prioritization techniques

- Regression testing at the system level
- Systems with nondeterministic behavior
- Systems that are developed by distributed teams

Transfer of techniques to industry

- Automation of regression testing
- Gathering information required for selection and prioritization
- Integrating into existing testing toolsets being used in industry

Assess Adequacy

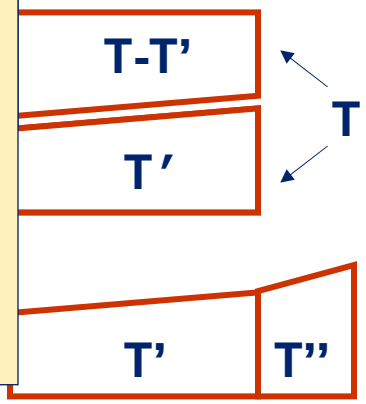
Select subset of T to rerun

Augment T for untested adequacy requirements

Assess Adequacy

How well do T, T',T'' or any test suites exercise P' with respect to changes?

Do the test cases exercise the changes so that they will affect execution?



Program and Modified Version

```
Procedure Avg
S1   count = 0
S2   fread(fptr,n)
S3   while (not EOF) do
S4     if (n<0)
S5       return(n)
       else
S6       nums[count] = n
S7       count++
       endif
S8     fread(fptr,n)
     endwhile
S9   avg = mean(nums,count)
S10  return(avg)
```

```
Procedure Avg'
S1'   count = 0
S2'   fread(fptr,n)
S3'   while (not EOF) do
S4'     if (n<=0)
S5'       return(n)
        else
S6'       nums[count] = n
S7'
       endif
S8'    fread(fptr,n)
     endwhile
S9'   avg = mean(nums,count)
S10'  return(avg)
```

Program and Modified Version

Criteria for change-impact propagation

- **Execution of the change**

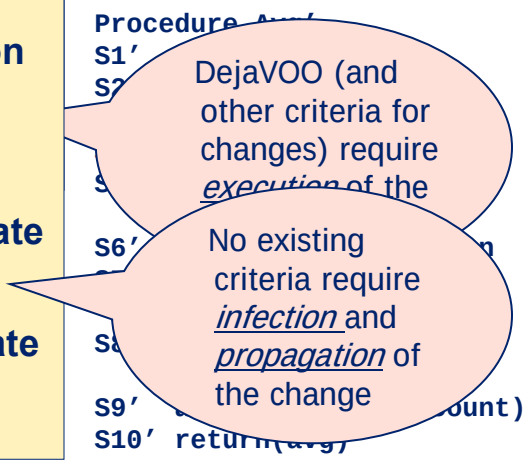
DejaVOO (and other criteria for changes) require *execution* of the change

```
Procedure Avg'
S1'
S2'
S3'
       else
S6'       nums[count] = n
S7'
       endif
S8'    fread(fptr,n)
     endwhile
S9'   avg = mean(nums,count)
S10'  return(avg)
```

Program and Modified Version

Criteria for change-impact propagation

- **E**xecution of the change
- **I**nfection of the state after change
- **P**ropagation of state to output where it can be observed



Program and Modified Version

Criteria for change-impact propagation

- **E**xecution of the change
- **I**nfection of the state after change
- **P**ropagation of state to output where it can be observed

Our new technique aims to add these requirements to the change criteria

Compute Change Testing Requirements

```
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)  if (n<=0)
S5       return(n)
      else
S6       nums[count] = n
S7       count++
      endif
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

Compute Change Testing Requirements

```
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)  if (n<=0)
S5       return(n)
      else
S6       nums[count] = n
S7       count++
      endif
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

S4

Infection: Path condition in Avg after S4 and path condition in Avg' after S4' differ

Condition for infection:

(n<=0) and not (n<0)

→ n must be 0 after S4'

Compute Change Testing Requirements

```
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)  if (n<=0)
S5       return(n)
      else
S6       nums[count] = n
S7       count++
      endif
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

S7

Infection: Value of state after execution of S7 in Avg and S7' in Avg' must differ

Condition for infection:

After

- S7 in Avg, count=count+1
- corresponding location in Avg', count=count

→ count≠count+1,

→ any value of count

Compute Change Testing Requirements

```
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)  if (n<=0)
S5       return(n)
      else
S6       nums[count] = n
S7       count++
      endif
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

Infection

Avg

PC	SS(n)
true	N_0

Avg'

PC'	SS'(n)
true	N_0

PC—path condition
SS—symbolic state

Perform symbolic execution from before change to get conditions

Compute Change Testing Requirements

```
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)  if (n<=0)
S5       return(n)
      else
S6       nums[count] = n
S7       count++
      endif
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

Infection

Avg

PC	SS(n)
true	N_0
(N_0<0) or (N_0>=0)	N_0

Avg'

PC'	SS'(n)
true	N_0
(N_0<=0) or (N_0>0)	N_0

PC—path condition
SS—symbolic state

Perform symbolic execution from before change to get conditions

Compute Change Testing Requirements

```
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)  if (n<=0)
S5       return(n)
      else
S6       nums[count] = n
S7       count++
      endif
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

Propagation

Avg

PC	SS(n)
true	N_0
(N_0<0) or (N_0>=0)	N_0

Avg'

PC'	SS'(n)
true	N_0
(N_0<=0) or (N_0>0)	N_0

Compute Change Testing Requirements

```
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)  if (n<=0)
S5       return(n)
      else
S6       nums[count] = n
S7       count++
      endif
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

Propagation

Avg

PC	SS(n)
true	N_0
(N_0<0) or (N_0>=0)	N_0
(N_0<0)	temp = 5/N_0

Avg'

PC'	SS'(n)
true	N_0
(N_0<=0) or (N_0>0)	N_0
(N_0<=0)	temp = 5/N_0

Compute Change Testing Requirements

```
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)  if (n<=0)
S5       return(n)
      else
S6       nums[count] = n
S7       count++
      endif
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

Propagation

Avg

PC	SS(n)
true	N_0
(N_0<0) or (N_0>=0)	N_0
(N_0<0)	temp = 5/N_0

Avg'

PC'	SS'(n)
true	N_0
(N_0<=0) or (N_0>0)	N_0
(N_0<=0)	temp = 5/N_0

Compute Change Testing Requirements

```
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)  if (n<=0)
S5       return(n)
      else
S6       nums[count] = n
S7       count++
      endif
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

Propagation

Avg

PC	SS(n)
true	N_0
(N_0<0) or (N_0>=0)	N_0
(N_0<0)	temp = 5/N_0
(N_0>=0)	N_0
and so on	

Avg'

PC'	SS'(n)
true	N_0
(N_0<=0) or (N_0>0)	N_0
(N_0<=0)	temp = 5/N_0
(N_0>0)	N_0
and so on	

Compute Change Testing Requirements

But

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

Our technique has two ways to improve efficiency

Avg'

PC'	SS'(n)
true	N_0
(N_0<=0) or (N_0>0)	N_0
(N_0<=0)	temp = 5/N_0
(N_0>0)	N_0
and so on	

```
S9  avg = mean(nums,count)
S10 return(avg)
```

Compute Change Testing Requirements

1. **Perform partial symbolic execution (PSE) beginning immediately before the change**
   - **computes conditions in terms of variables immediately before change**
   - **avoids symbolic execution from beginning of program to change**

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

```
S8    fread(fptr,n)
    endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

	SS'(n)
	N_0
	N_0
	temp = N_0
(N_0>0)	N_0
and so on	

	SS'(n)
	N_0
	N_0
	temp = N_0
(N_0>=0)	N_0
and so on	

Compute Change Testing Requirements

2. **Perform PSE for some specified distance (user selected) 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**

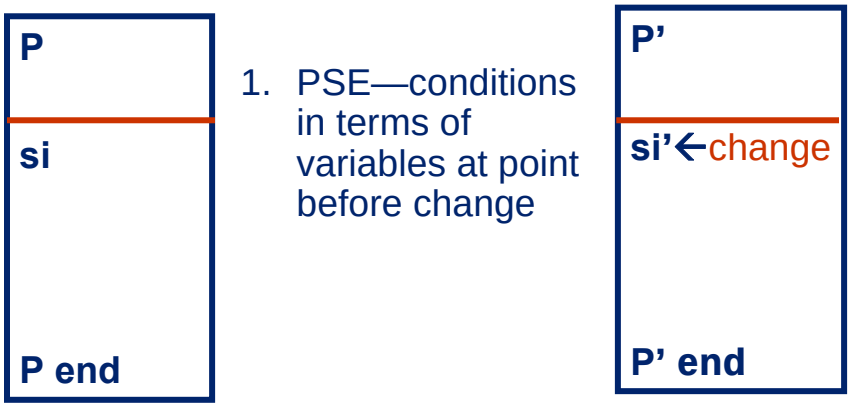
   Greater distances improve confidence in propagation to output

```
S9  avg = mean(nums,count)
S10 return(avg)
```

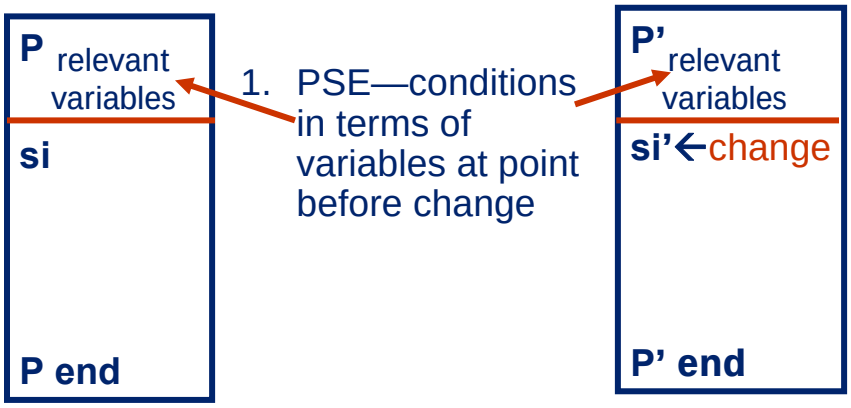
	SS'(n)
	N_0
	N_0
	temp = $5/N_0$
	N_0
and so on	

	SS'(n)
	N_0
	N_0
	temp = $5/N_0$
	N_0
and so on	

Compute Change Testing Requirements



Compute Change Testing Requirements

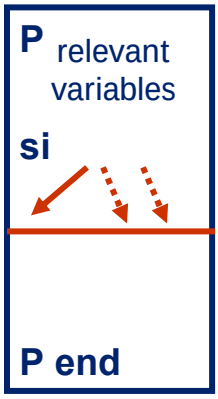


Compute Change Testing Requirements

1. PSE—conditions in terms of variables at point before change
2. Distance is N data and control dependences from change (slicing like)

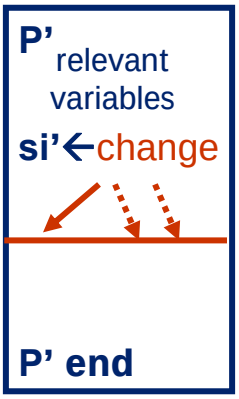
Distance is 0

Compute Change Testing Requirements

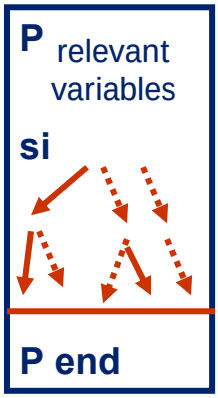


1. PSE—conditions in terms of variables at point before change
2. Distance is N data and control dependences from change (slicing like)

Distance is 1

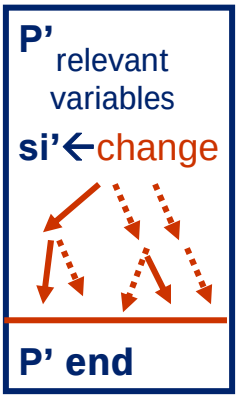


Compute Change Testing Requirements



1. PSE—conditions in terms of variables at point before change
2. Distance is N data and control dependences from change (slicing like)

Distance is 2



Insert Probes to Record Coverage

To record adequacy (coverage of conditions)

Instrument modified program so that probes check for satisfaction of condition before change (e.g., before S4' and before S7')

```
Procedure Avg'
S1'   count = 0
S2'   fread(fptr,n)
S3'   while (not EOF) do
S4'     if (n<=0)
S5'       return(n)
        else
S6'       nums[count] = n
S7'
        endif
S8'     fread(fptr,n)
      endwhile
S9'   avg = mean(nums,count)
S10'  return(avg)
```

Empirical Study

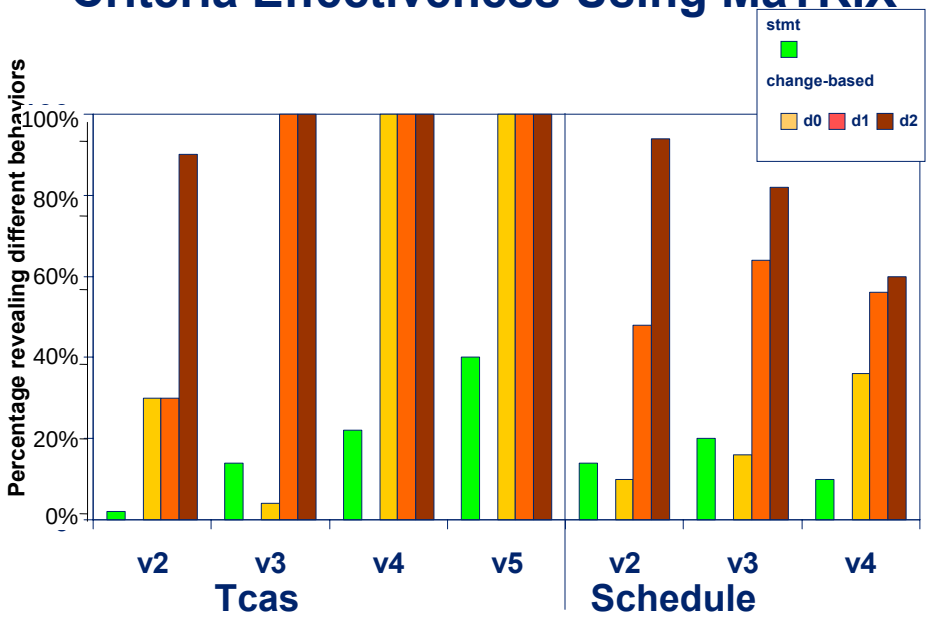
Goal: To compare the effectiveness of our changed-based criterion with statement-based criterion

Subjects: Tcas (4 versions) and Schedule (3 versions) (each version has one fault)

Procedure:

- Randomly generated 50 test suites per criterion
- Recorded the number of test suites that produce different outputs

Criteria Effectiveness Using MaTRIX



Achievements: Research

- New change test criteria
  - show promise for assessing adequacy of test suite around changes
  - can be performed on large programs since only area around change being evaluated
  - empirical evaluation encouraging

Challenges

Computing change impact criteria still in *very early* stages

- Handling multiple and interacting changes
- Implementing PSE efficiently
- Determining good distance (empirically)
- Tracking impact dynamically to check for propagation to end of program
- Creating strategy to handle false positives
- Others??

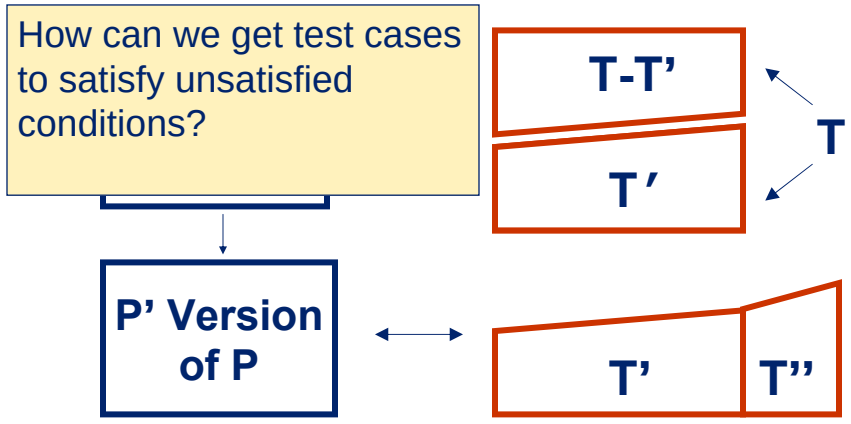
Augment T for … Requirements

Select subset of T to rerun

Assess adequacy

Augment T for … Requirements

How can we get test cases to satisfy unsatisfied conditions?



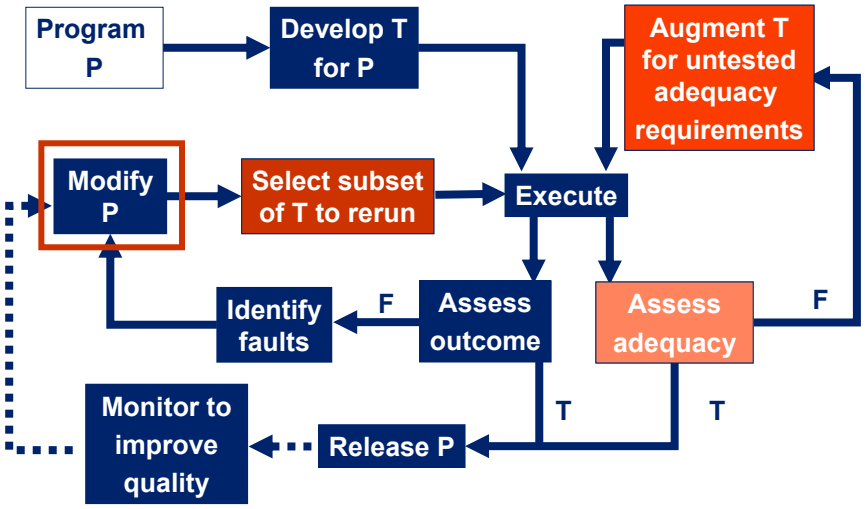
Augment T for … Requirements

- Unsatisfied conditions can be used by developers to create new test cases
- Automatically generate test cases to satisfy conditions
  - Use of regression test suite to generate new inputs to satisfy change test requirements
  - Dynamic information provides concrete values to guide symbolic execution
  - Extension of existing work in concolic test data generation

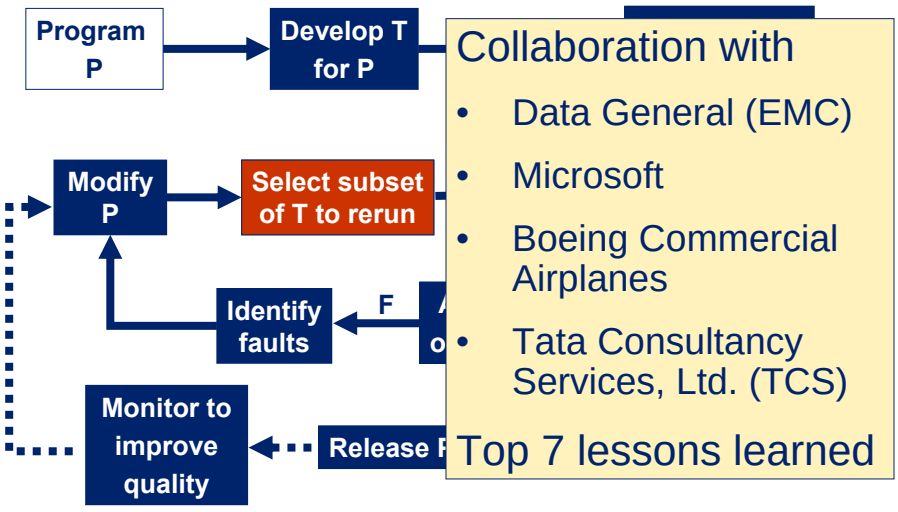
Challenges

- Generating test cases to satisfy conditions
  - may use regression test suite and apply existing techniques
  - ensure that the techniques are efficient for large programs

Testing Evolving Software



Industry-Academic Collaboration



Collaboration with

- Data General (EMC)
- Microsoft
- Boeing Commercial Airplanes
- Tata Consultancy Services, Ltd. (TCS)

Top 7 lessons learned

Top 7 Lessons Learned

1. To solve real problems, need to interact with industry on a regular basis
2. To incorporate new testing technology often requires extensive change in process, so difficult to achieve
3. To show effectiveness, techniques must be evaluated on real systems
4. To evaluate, prototype must integrate into industrial environment

Top 7 Lessons Learned

5. To get developers' help in evaluation, prototype must be usable for them
6. To get the evaluation done, need internal champion and internal developer needed for experimentation
7. To discover important problems, many ideas for interesting research emerge from collaborations

Summary

- **Regression testing approach**
  - Concentrate testing around the changes
  - Automate as much as possible
- **Components**
  - Regression test selection and prioritization
  - Test-suite adequacy assessment
  - Test-suite augmentation to satisfy adequacy requirements

Promise for Testing Evolving Software

We're making progress on automating the regression testing process

By continuing to work on these and related problems, and collaborating with industry, we will be able to automate regression testing

Thereby providing a way to test well around changes efficiently, provide confidence in the software, and eliminate many defects

Questions?