

# Software Testing: An Evolution-Centric Perspective

Gregg Rothermel

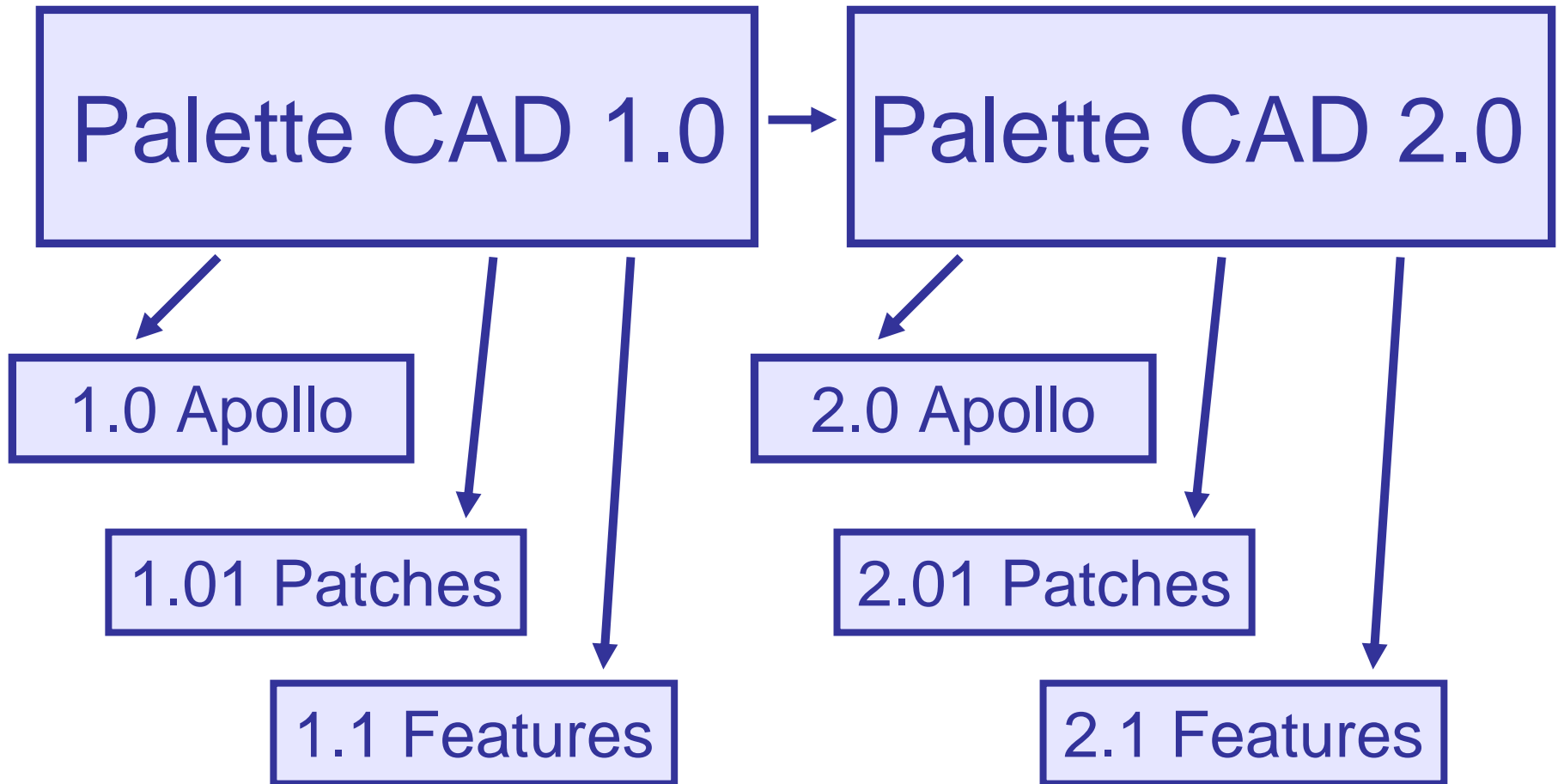


Dept. of Computer Science and Engineering  
University of Nebraska - Lincoln

Supported by the National Science Foundation,  
Microsoft, Lockheed Martin,  
and Boeing Commercial Aircraft Group



# Evolving Software



# An Evolution-Centric Perspective on Software Testing

- Focus on evolution first
- Harness evolution
- Design for regression testability

# Overview of Presentation

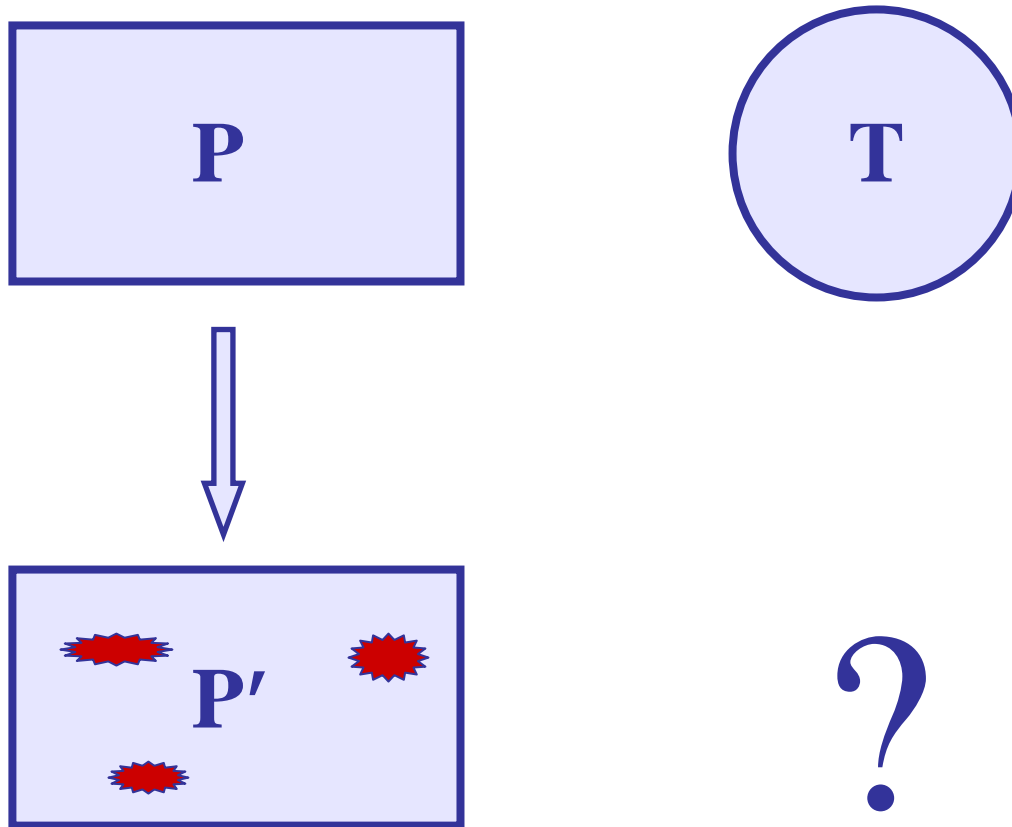
- Testing evolving software
- Regression test selection
  - Dejavu algorithm
  - Analytical and empirical evaluation
- Test case prioritization
  - Prioritization measures and techniques
  - Empirical results
- Ongoing and Future Work

# Overview of Presentation

- Testing evolving software
- Regression test selection
  - Dejavu algorithm
  - Analytical and empirical evaluation
- Test case prioritization
  - Prioritization measures and techniques
  - Empirical results
- Ongoing and Future Work

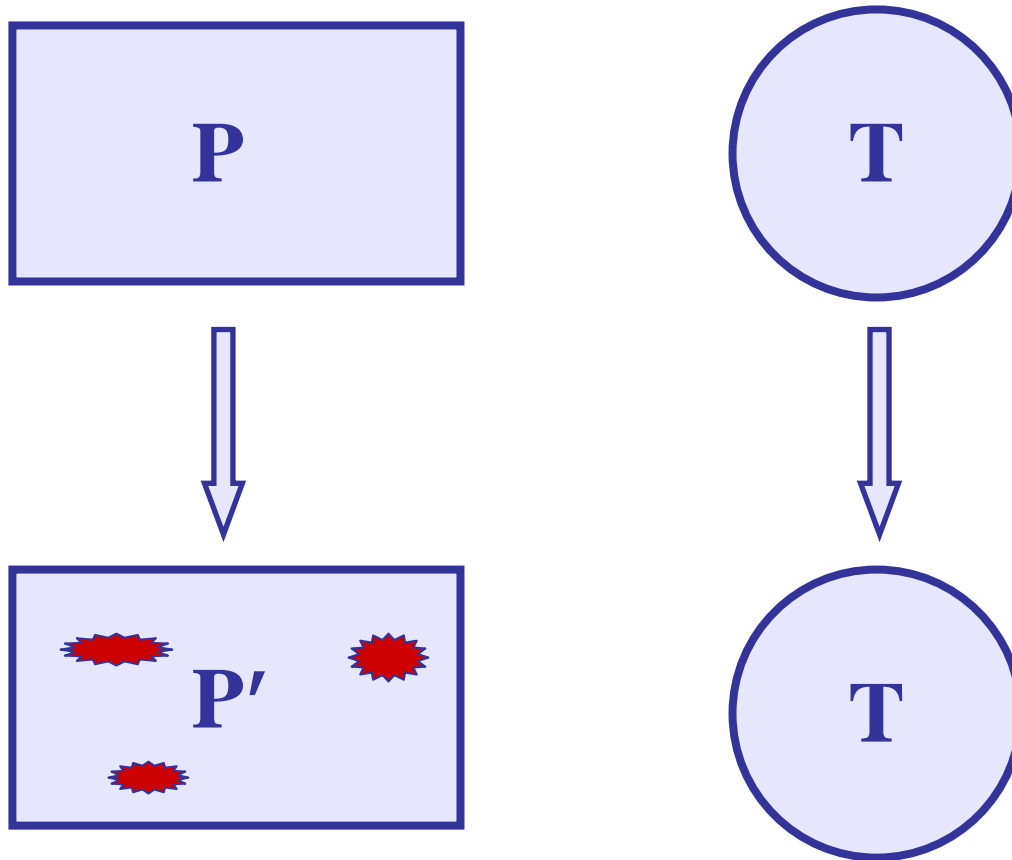
# Testing Evolving Software

## Regression testing



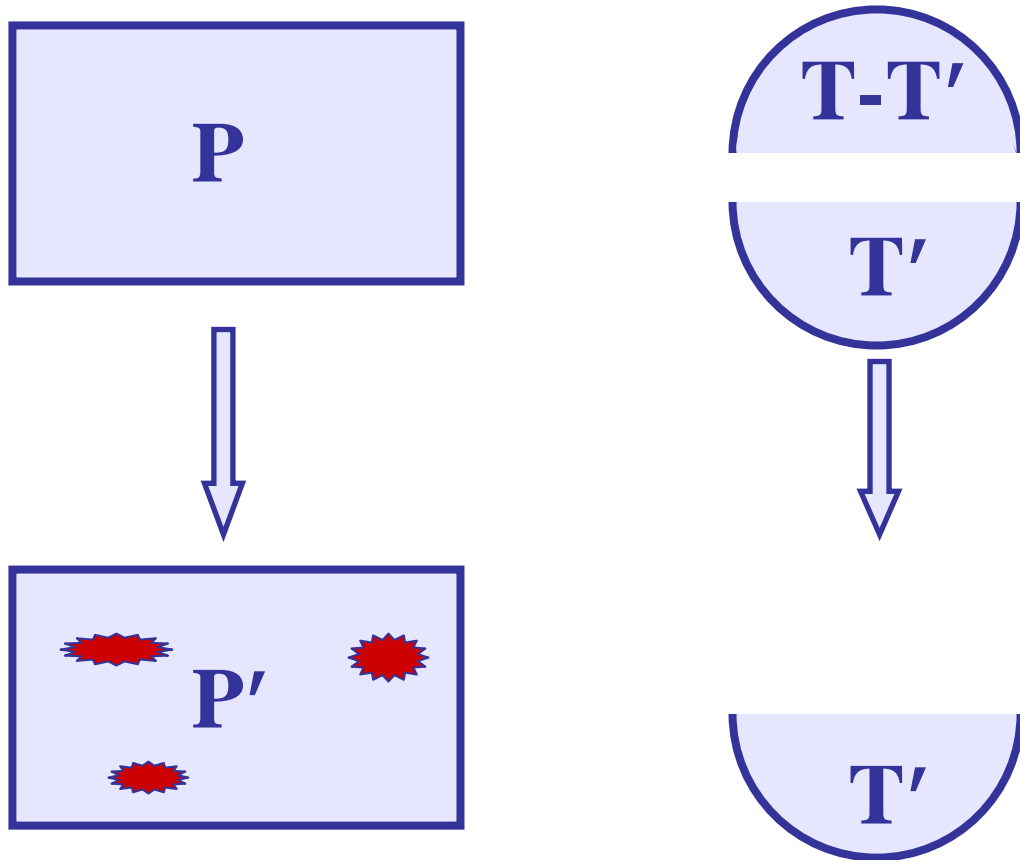
# Testing Evolving Software

Retest-all



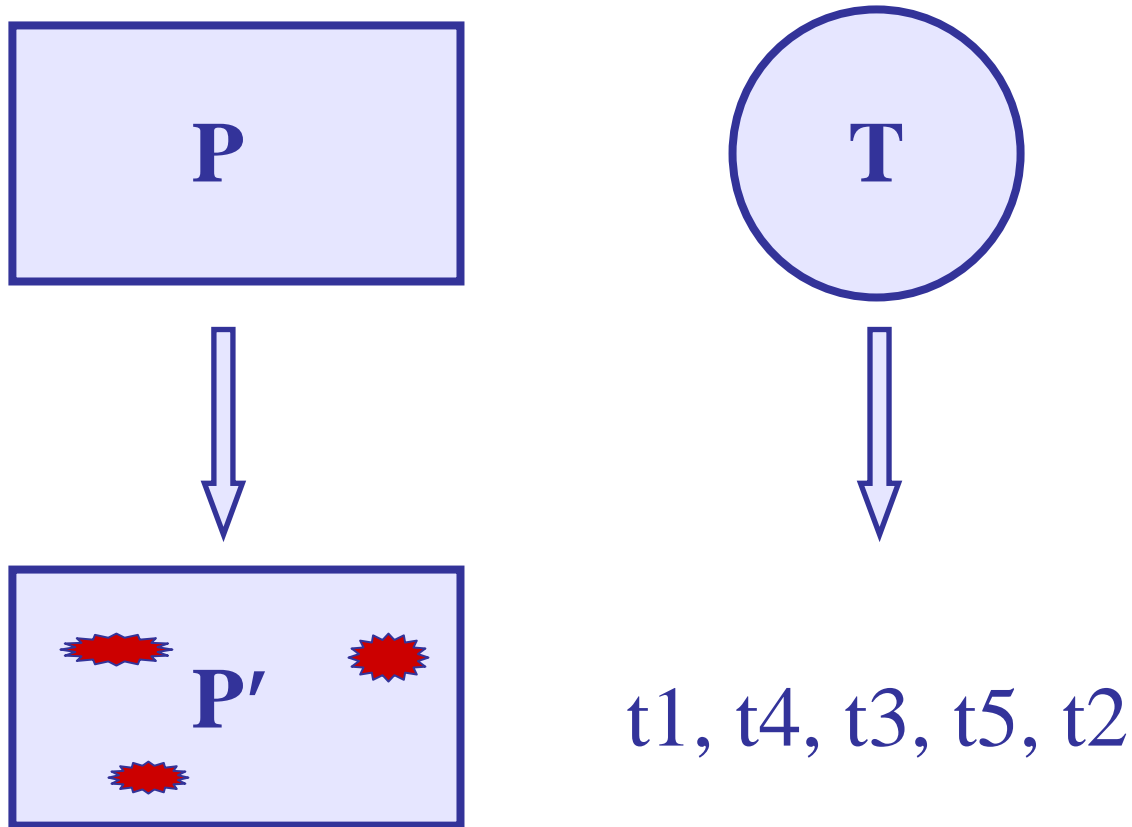
# Testing Evolving Software

## Regression test selection



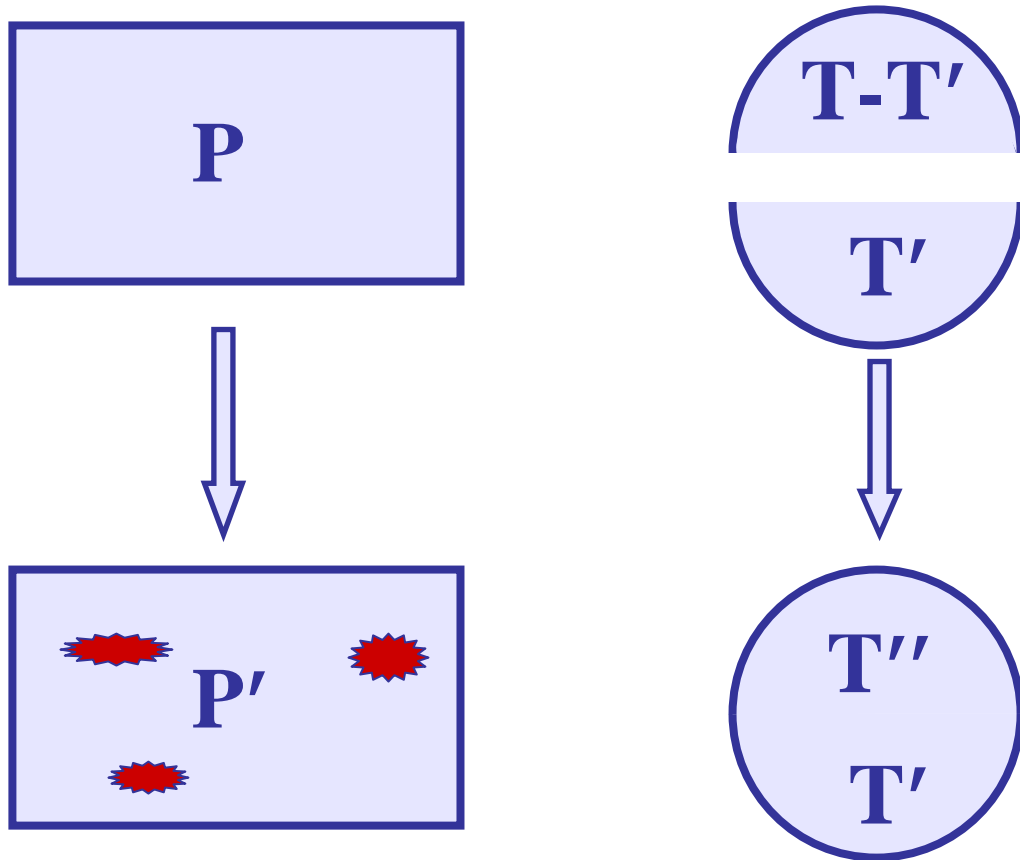
# Testing Evolving Software

## Test case prioritization



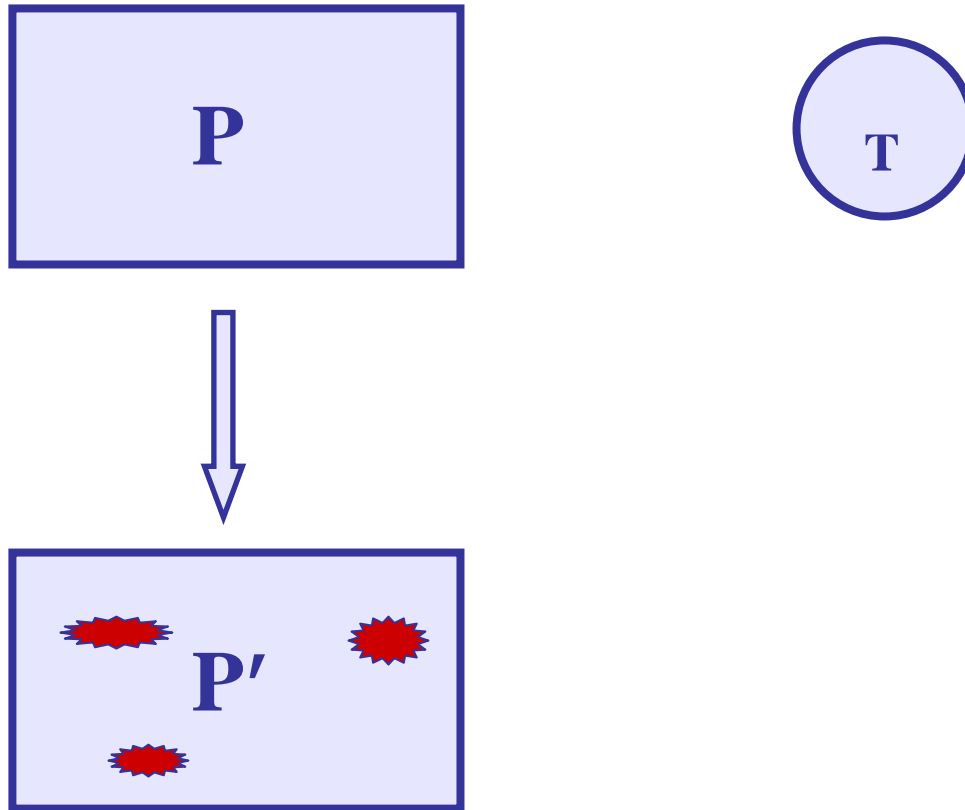
# Testing Evolving Software

Test suite augmentation and impact analysis



# Testing Evolving Software

## Test suite reduction

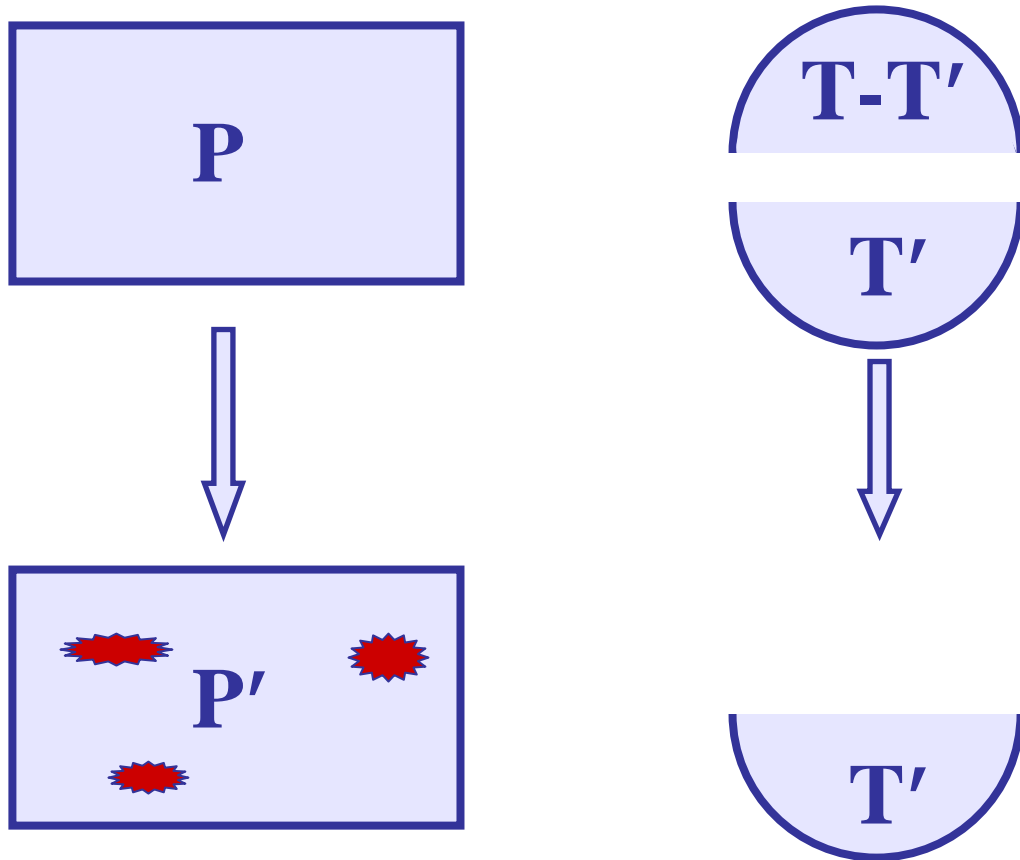


# Overview of Presentation

- Testing evolving software
- Regression test selection
  - Dejavu algorithm
  - Analytical and empirical evaluation
- Test case prioritization
  - Prioritization measures and techniques
  - Empirical results
- Ongoing and Future Work

# Testing Evolving Software

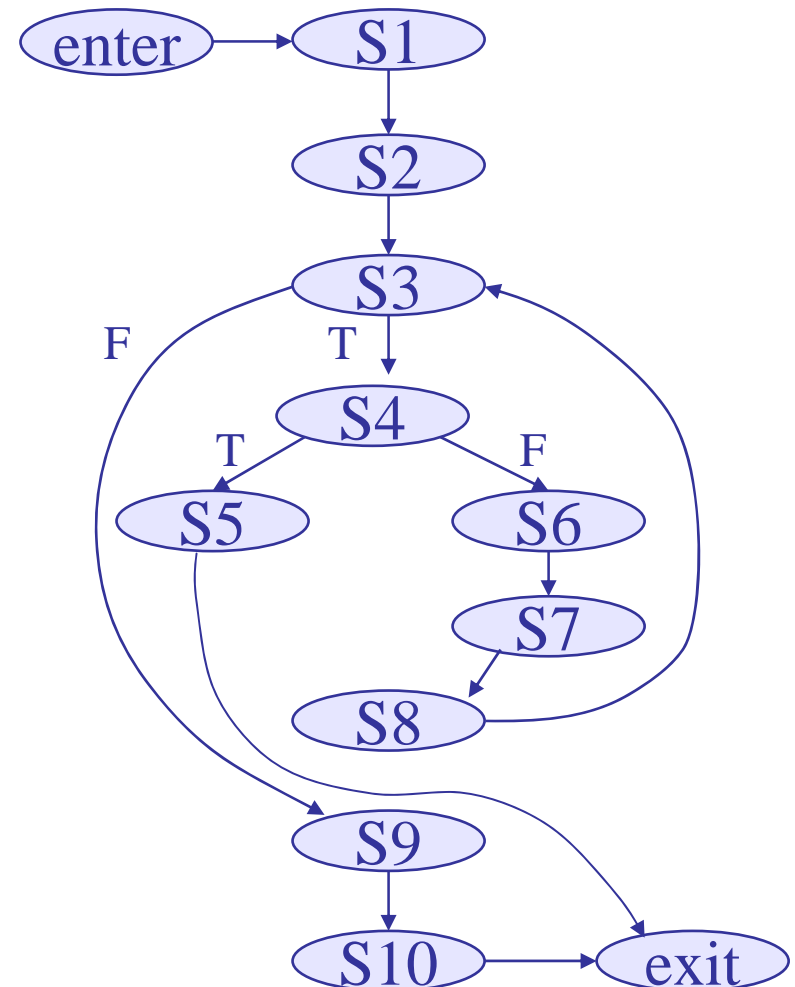
## Regression test selection



# Control Flow Graphs

## Procedure Avg

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



# Execution Traces

## Procedure Avg

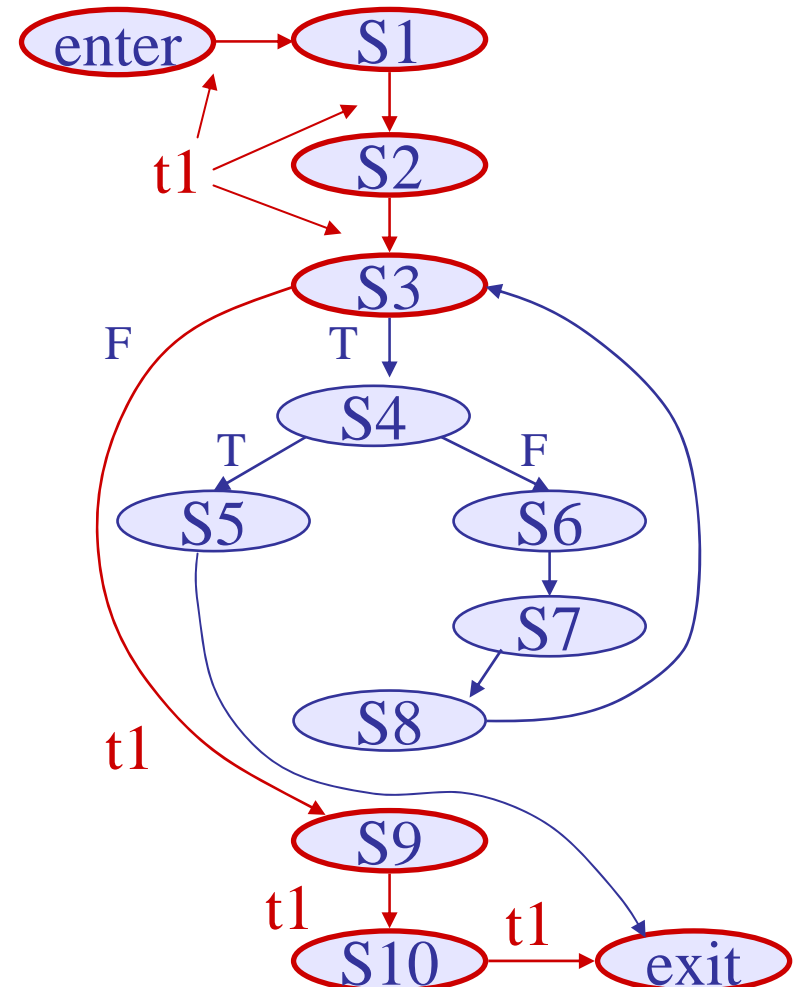
```
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)
S5      return(error)
      else
S6        nums[count] = n
S7        count++
      endif
S8    fread(fptr,n)
      endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

test	input	output
t1	empty file	0

# Execution Traces

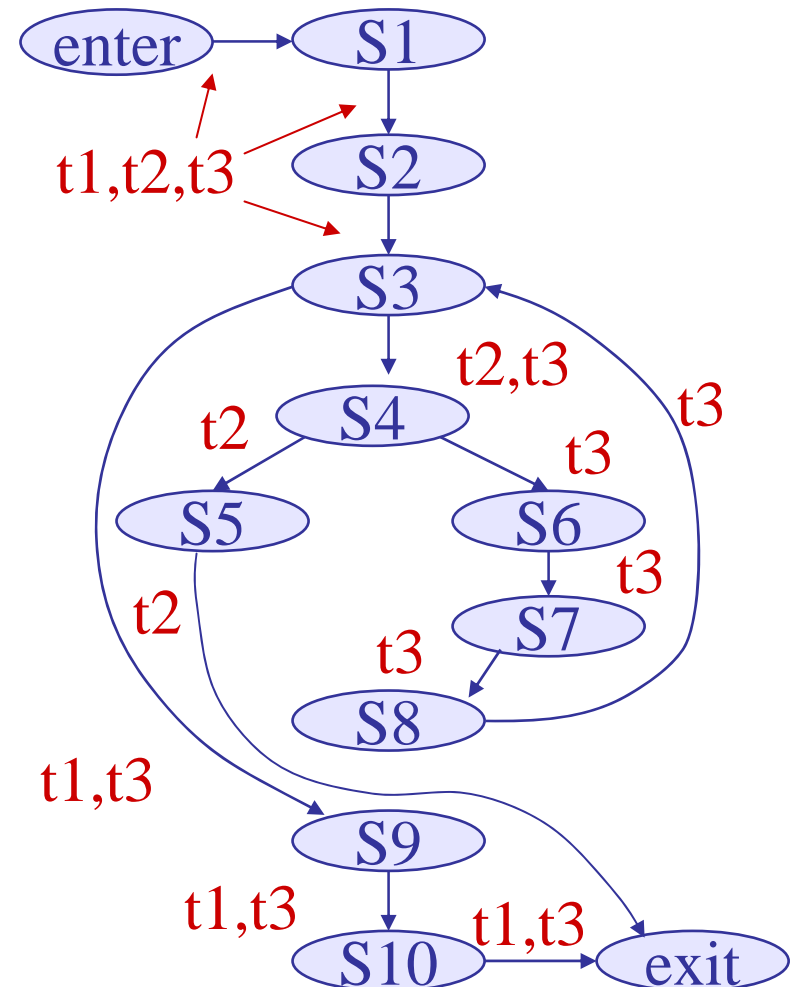
## Procedure Avg

```
S1 count = 0
S2 fread(fptr,n)
S3 while (not EOF) do
S4   if (n<0)
S5     return(error)
S6   else
S7     nums[count] = n
S8     count++
S9   endif
S10  fread(fptr,n)
S11 endwhile
S12 avg = mean(nums,count)
S13 return(avg)
```



# Test History Information

test	input	output
t1	empty file	0
t2	-1	error
t3	1 2 3	2




# Program and Modified Version

## Procedure Avg

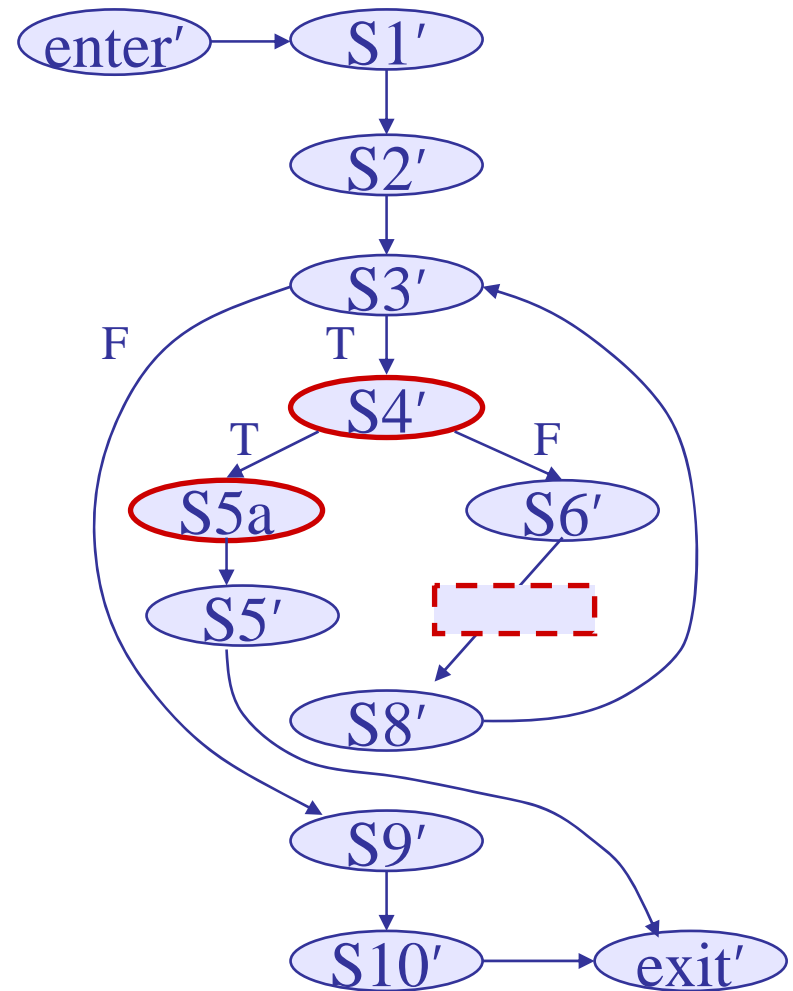
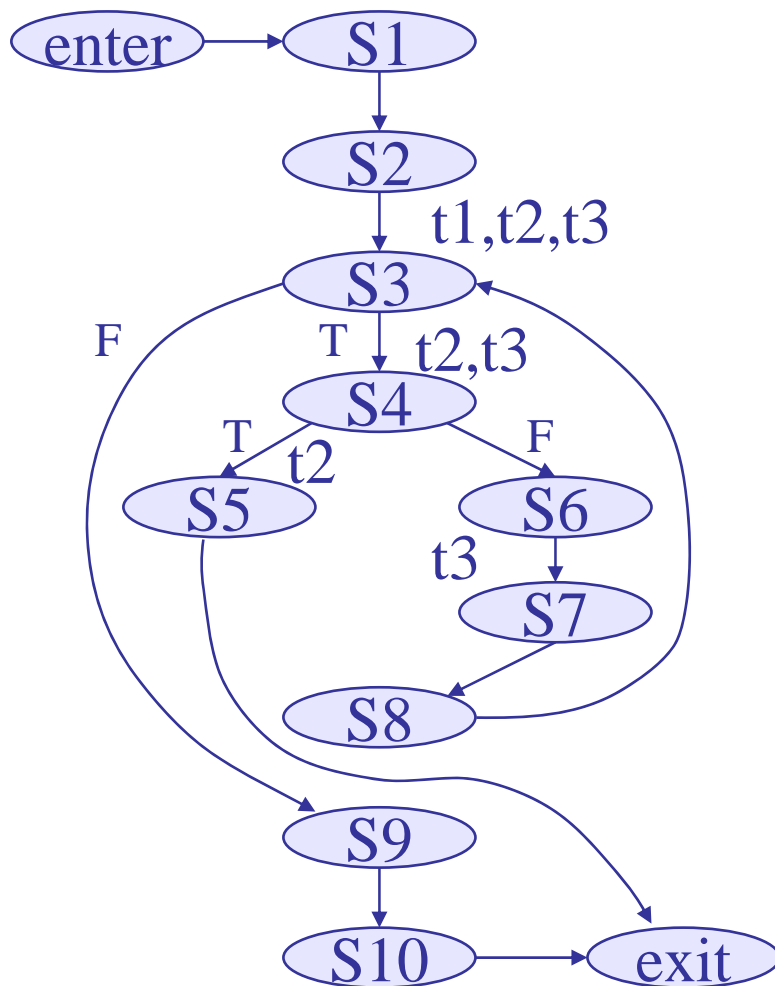
```
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4    if (n<0)

S5      return(error)
      else
S6      nums[count] = n
S7      count++
      endif
S8  fread(fptr,n)
      endwhile
S9  avg = mean(nums,count)
S10 return(avg)
```

## Procedure Avg'

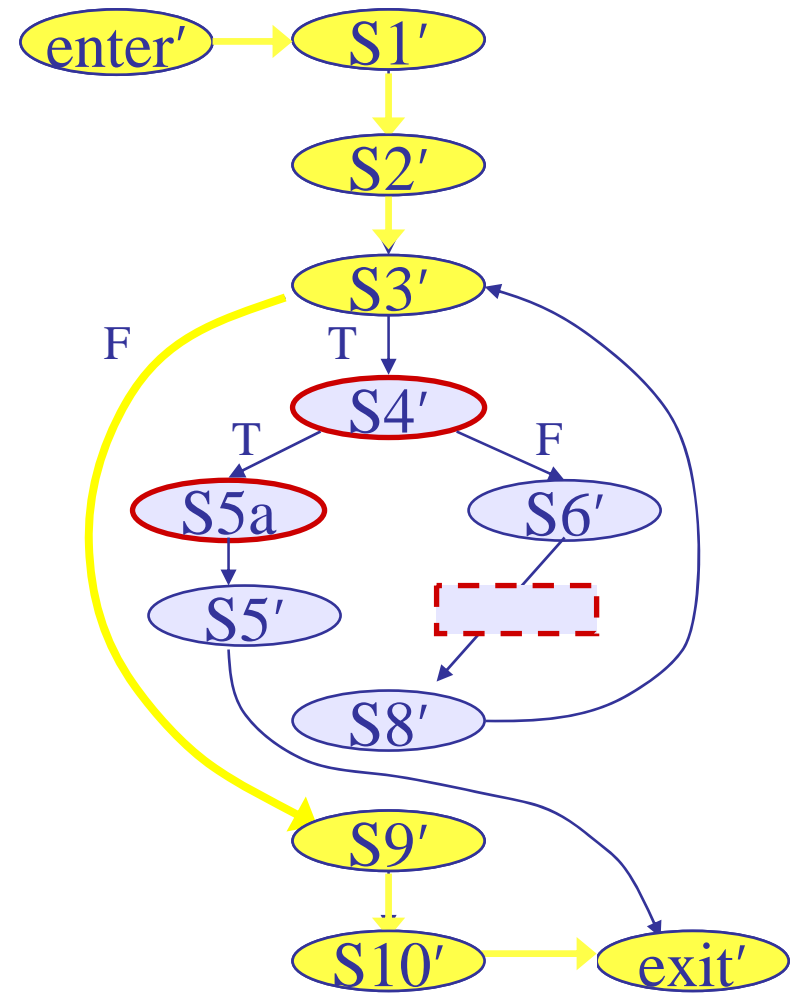
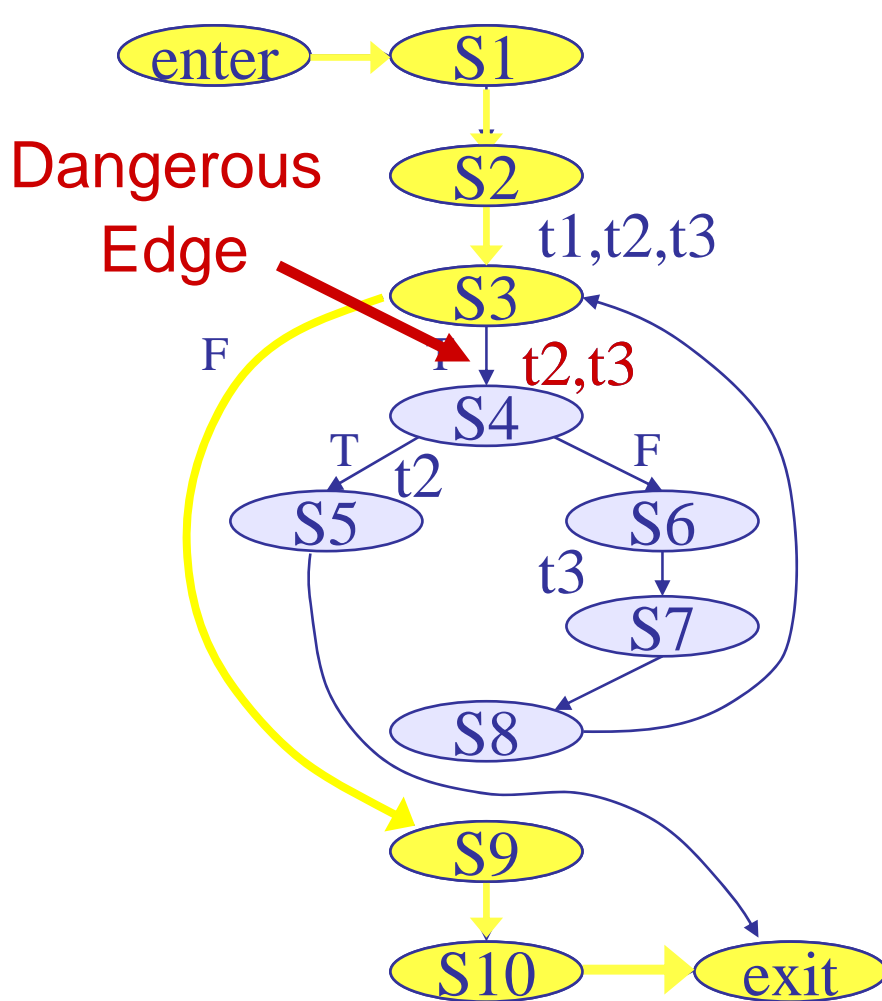
```
S1'  count = 0
S2'  fread(fptr,n)
S3'  while (not EOF) do
S4'   if (n<=0)
S5a   print("input error")
S5'   return(error)
      else
S6'   nums[count] = n
      
S7'   endif
S8'  fread(fptr,n)
      endwhile
S9'  avg = mean(nums,count)
S10' return(avg)
```

# CFG and Modified CFG



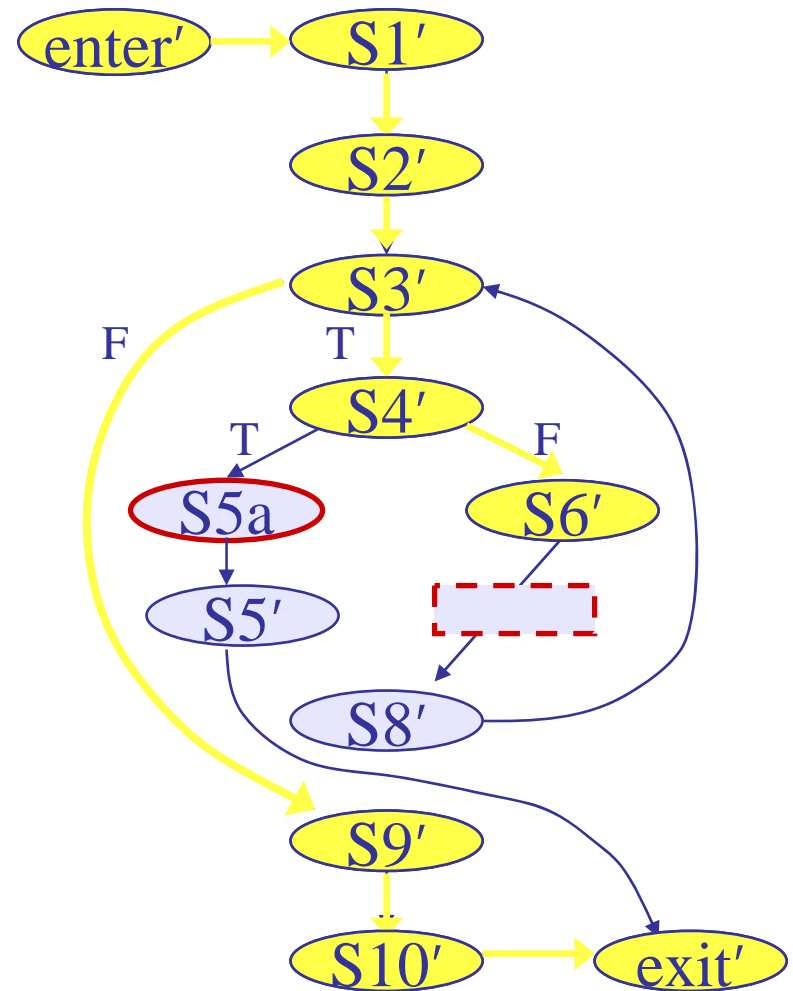
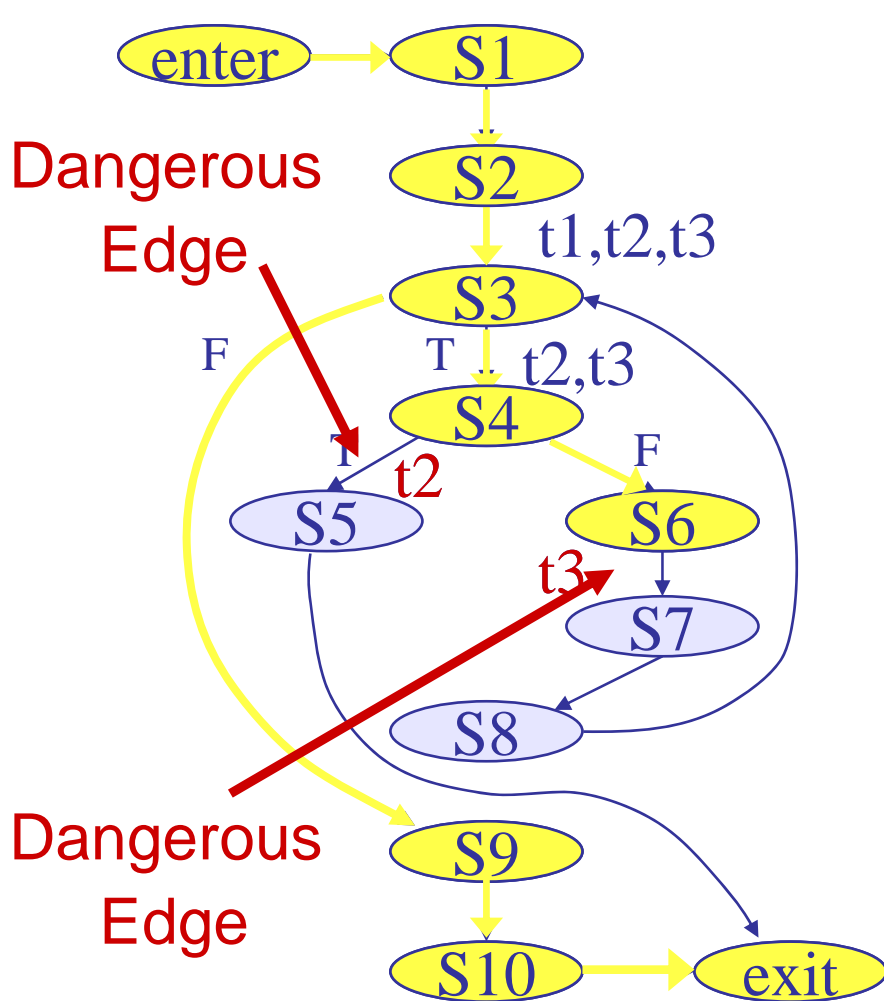
# Example 1

$$T' = \{t2, t3\}$$



# Example 2

$$T' = \{t2, t3\}$$



# Algorithm Dejavu

Input:  $P, P', T$       Output:  $T'$

1. Build CFGs  $G$  and  $G'$  for  $P$  and  $P'$
2. Compare( $G$ .EntryNode, $G'$ .EntryNode)
3. Compare( $N, N'$ )
4.     mark  $N$  “ $N'$ -visited”
5.     for each pair of successors  $C$  and  $C'$  of  $N$  and  $N'$
6.     on equivalently labeled edges do
7.         if  $C$  is not marked “ $C'$ -visited”
8.             if  $C$  and  $C'$  are not lexically identical
9.                  $T' = T' \cup \text{TestsOnEdge}(N, C, T)$
10.             else
11.                 Compare( $C, C'$ )

# Interprocedural Methodologies

1. Compare all pairs of procedures
2. Create & walk interprocedural representation
3. Compare all pairs of procedures identified by configuration management system

# Overview of Presentation

- Testing evolving software
- Regression test selection
  - Dejavu algorithm
  - Analytical and empirical evaluation
- Test case prioritization
  - Prioritization measures and techniques
  - Empirical results
- Ongoing and Future Work

# Algorithm Efficiency

CFG construction: linear in program size

Graph walk (graph sizes  $n, n'$ ; test set size  $t$ ):

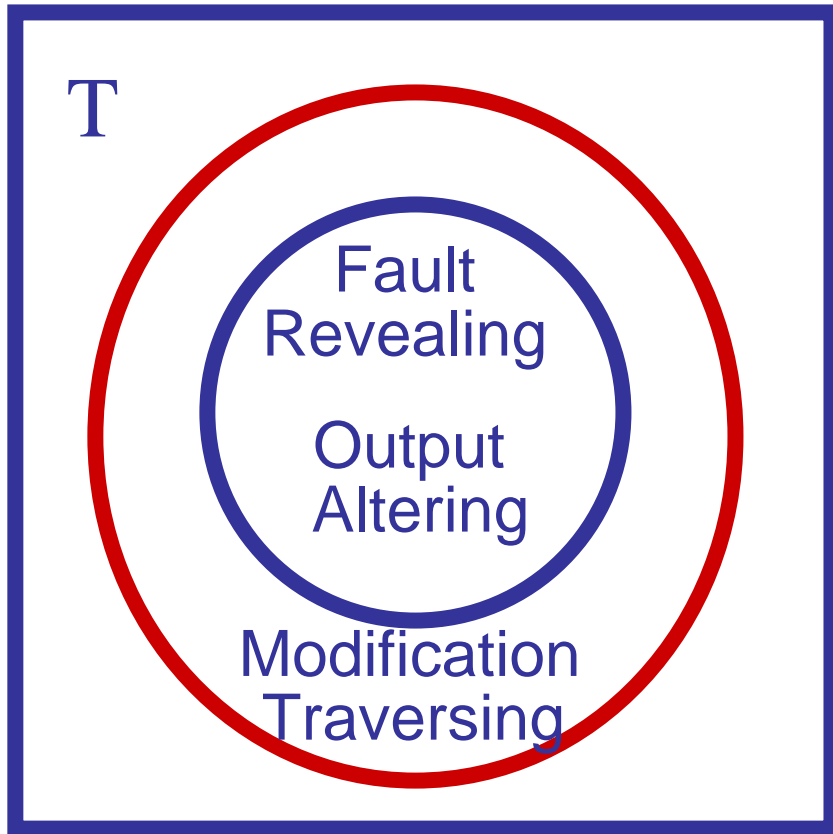
$$O ( t * n * n' )$$

(with *multiply-visited nodes*)

$$O ( t * \min(n, n') )$$

(with no *multiply-visited nodes*)

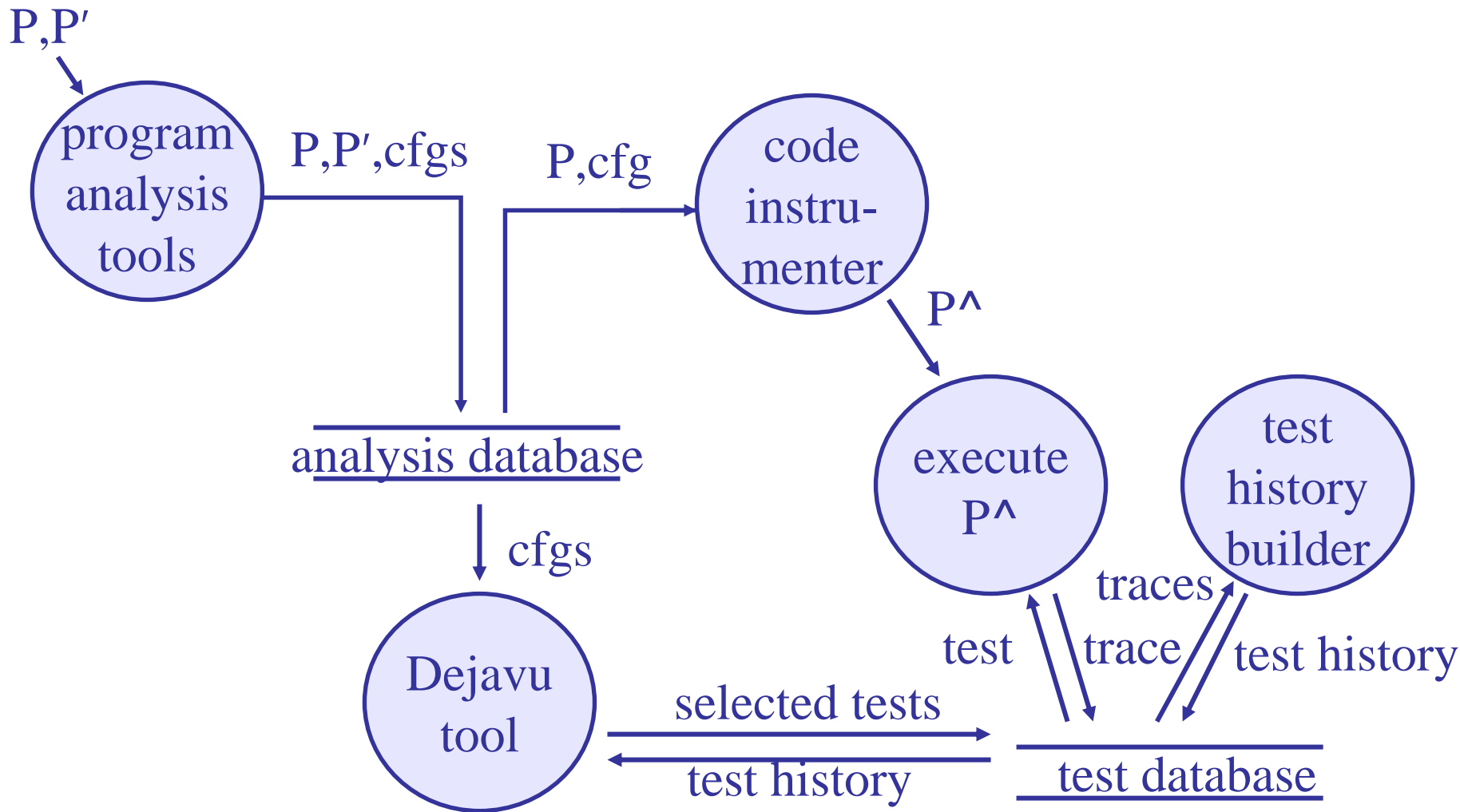
# Precision and Safety



Conditions:

1. P was correct for all tests in T
2. T contains no obsolete tests
3. Controlled regression testing

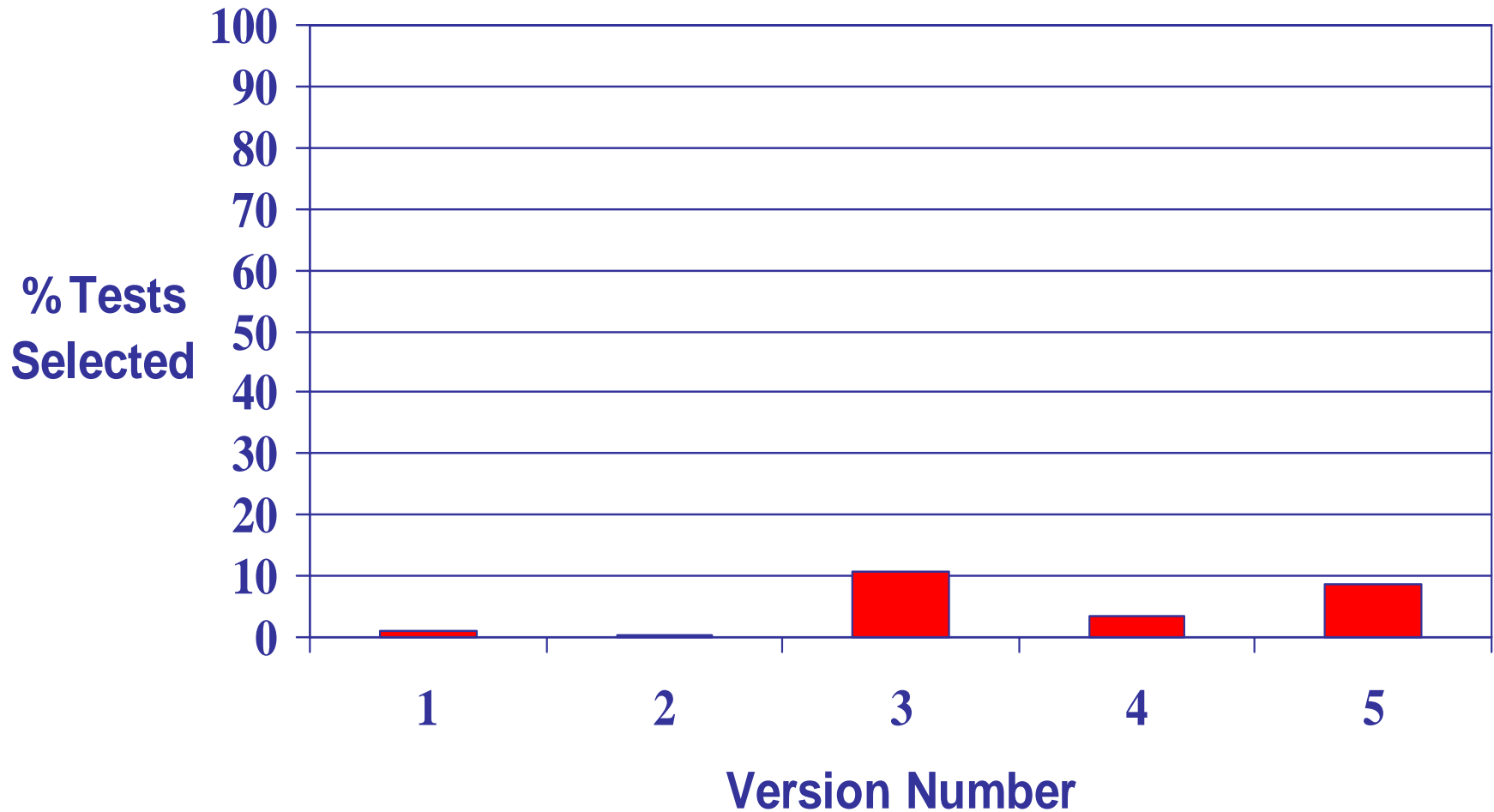
# Regression Test Selection System



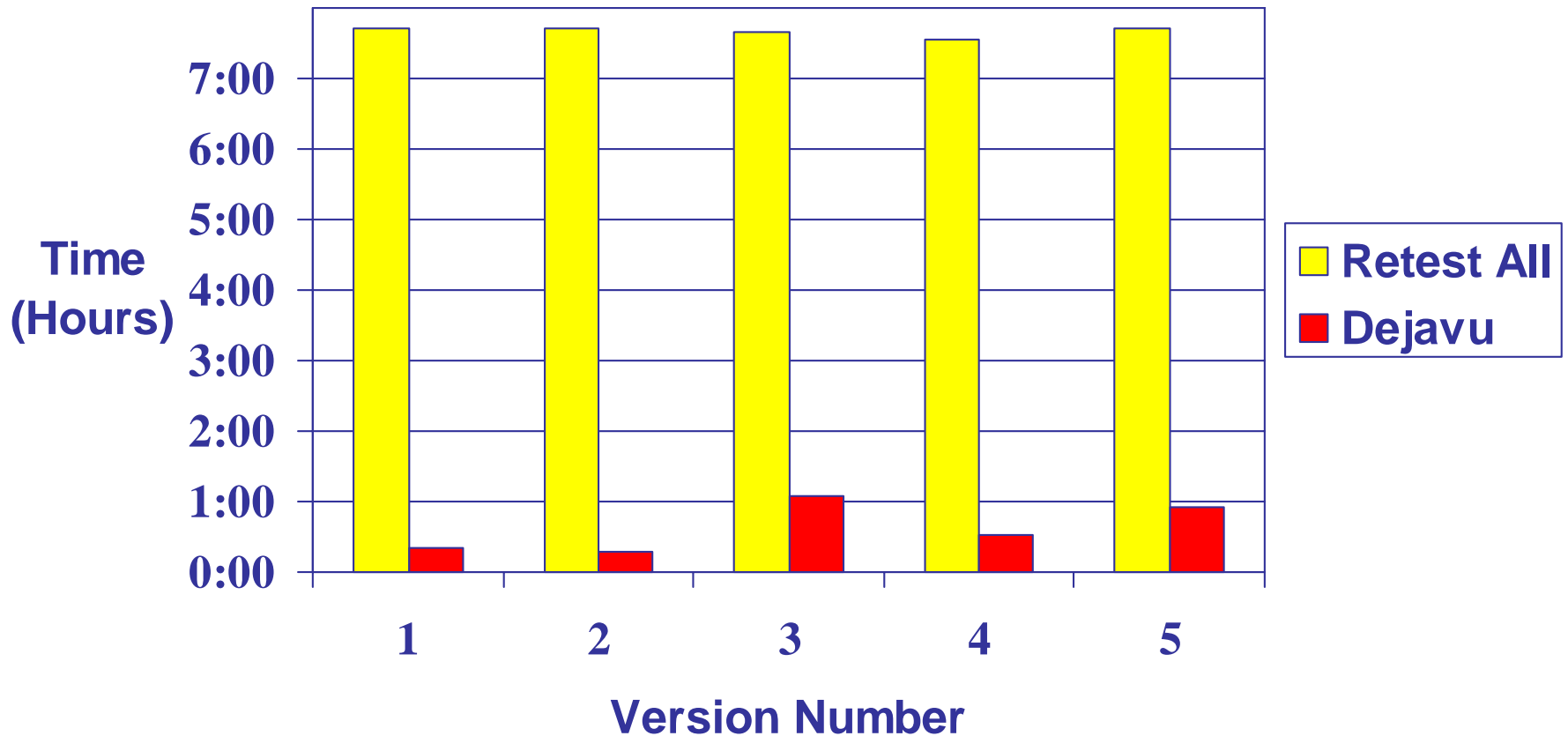
# Study 1: Empire

<b>Program</b>	<b>Procs</b>	<b>LOC</b>	<b>Vers</b>	<b>Tests</b>
server	766	49316	5	1033
<b>Version</b>		<b>Functions Modified</b>	<b>LOC Modified</b>	
1		3	114	
2		2	55	
3		11	726	
4		11	62	
5		42	221	

# Study 1: Test Selection Percentages



# Study 1: Cost Effectiveness



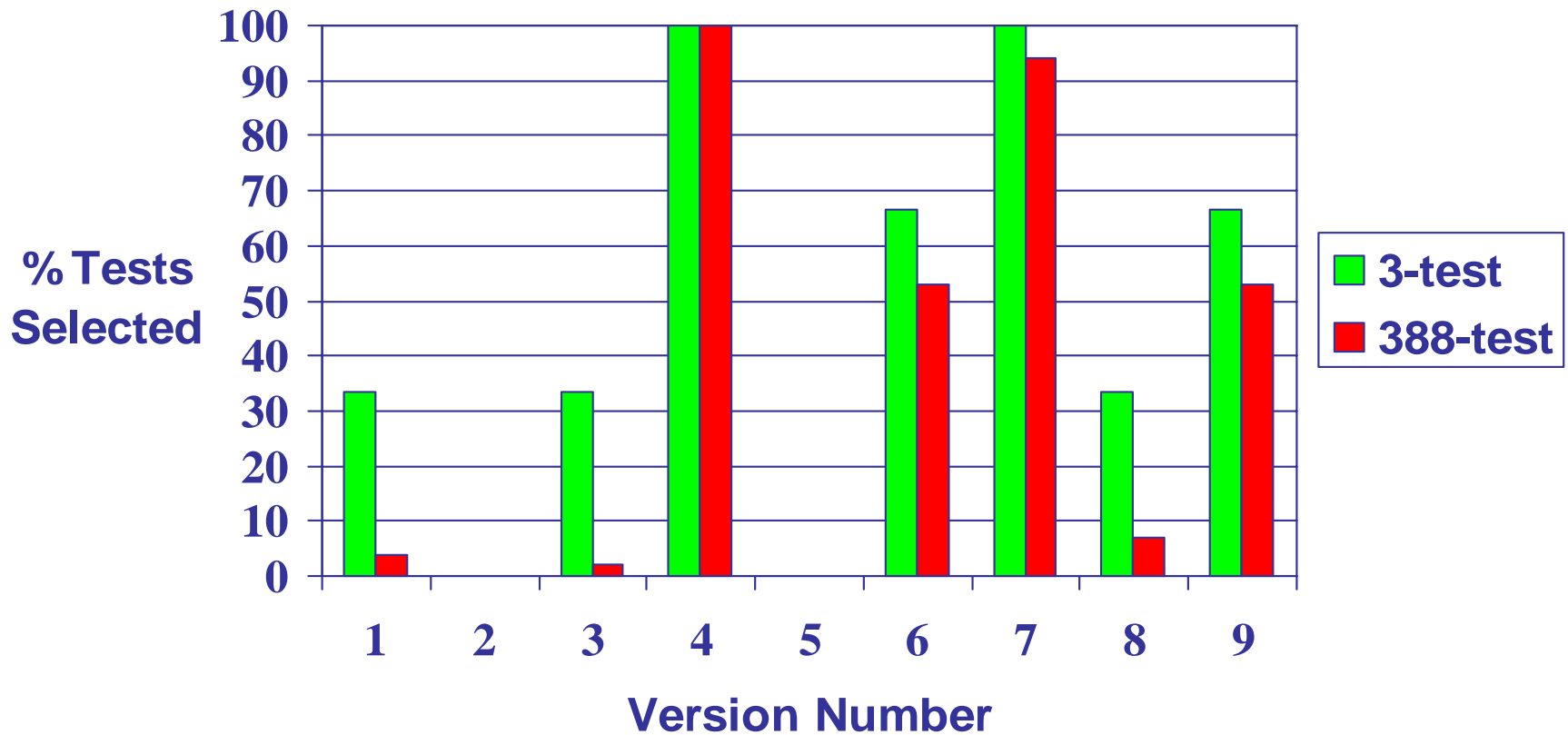
# Study 2:

## Windows NT Calculator

<b>Program</b>	<b>Funcs</b>	<b>LOCs</b>	<b>Vers</b>	<b>Tests</b>
calculator	27	2145	9	3/388

<b>Version</b>	<b>Functions Modified</b>	<b>LOCs Modified</b>
1	1	1
2	2	12
3	1	1
4	12	264
5	1	3
6	3	4
7	3	245
8	2	15
9	2	44

# Study 2: Test Selection Percentages

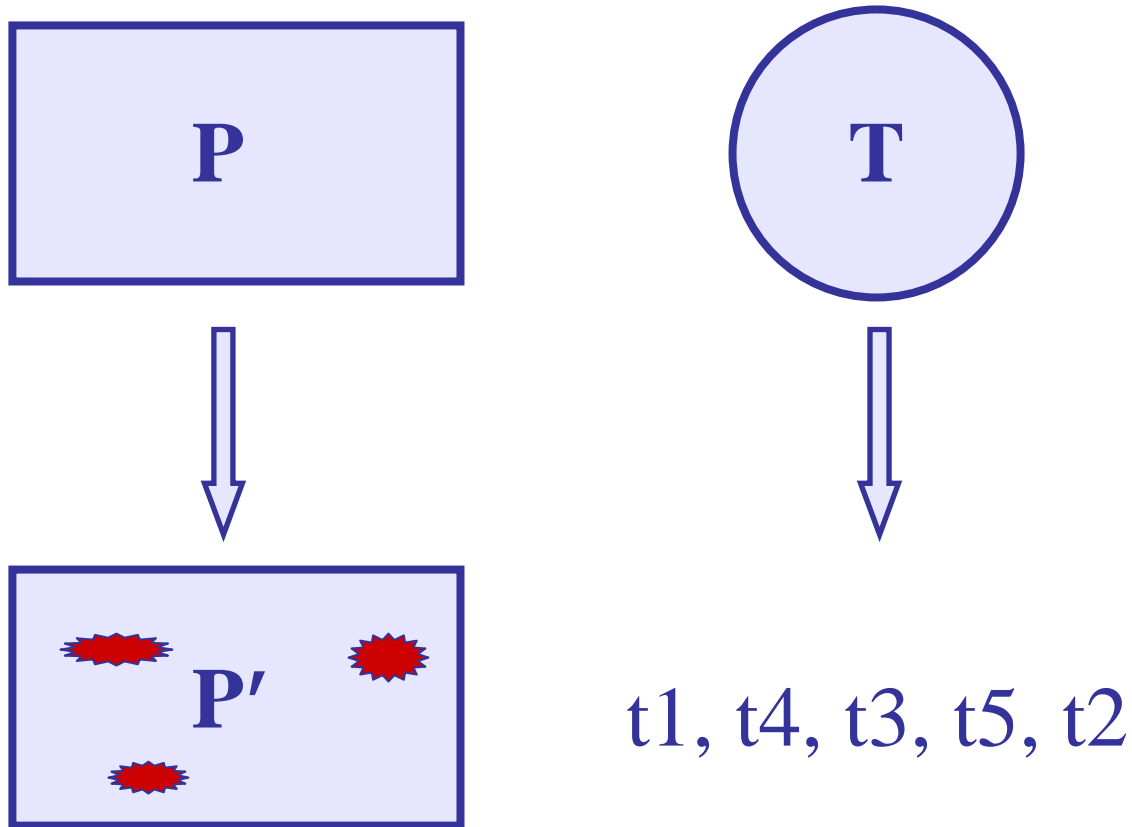


# Overview of Presentation

- Testing evolving software
- Regression test selection
  - Dejavu algorithm
  - Analytical and empirical evaluation
- **Test case prioritization**
  - **Prioritization measures and techniques**
  - Empirical results
- Ongoing and Future Work

# Testing Evolving Software

## Test case prioritization



# Addressing the Problem Requires:

- (1) Objective function  $f$
- (2) Algorithms/heuristics for maximizing  $f$

# Addressing the Problem Requires:

- (1) Objective function  $f$
- (2) Algorithms/heuristics for maximizing  $f$

# Prioritization Objectives

- Cover system components more quickly
- Build reliability estimates more quickly
- Reveal faults earlier in testing
- Reveal regression faults earlier in testing
- Reveal critical faults earlier in testing

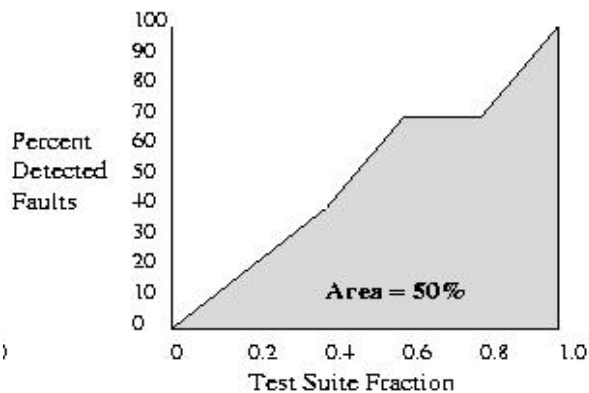
# Prioritization Objectives

- Cover system components more quickly
- Build reliability estimates more quickly
- Reveal faults earlier in testing
- Reveal regression faults earlier in testing
- Reveal critical faults earlier in testing

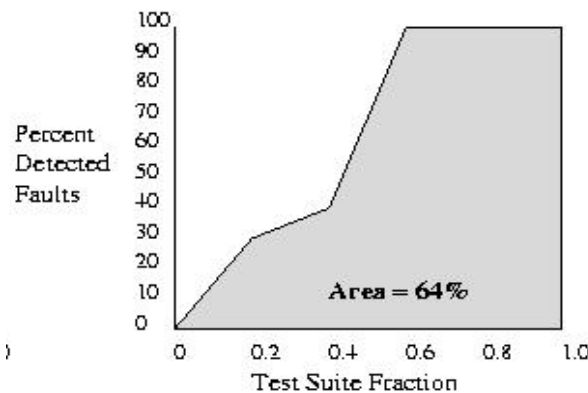
# Rate of Fault Detection – APFD

TESTS	Faults									
	1	2	3	4	5	6	7	8	9	10
A	X				X					
B				X		X				
C	X	X	X	X	X	X	X			
D					X					
E								X	X	X

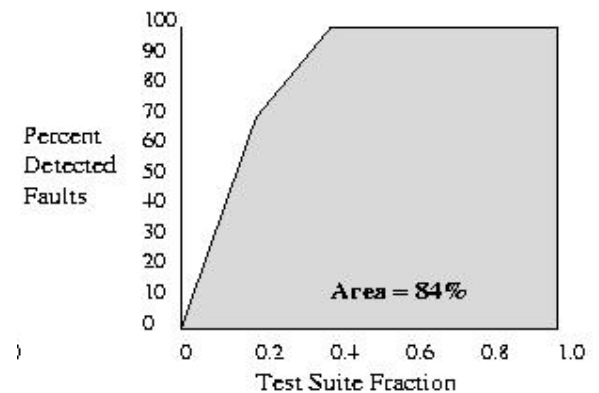
T1: A-B-C-D-E



T2: E-D-C-B-A



T3: C-E-B-A-D



# Addressing the Problem Requires:

(1) Objective function  $f$

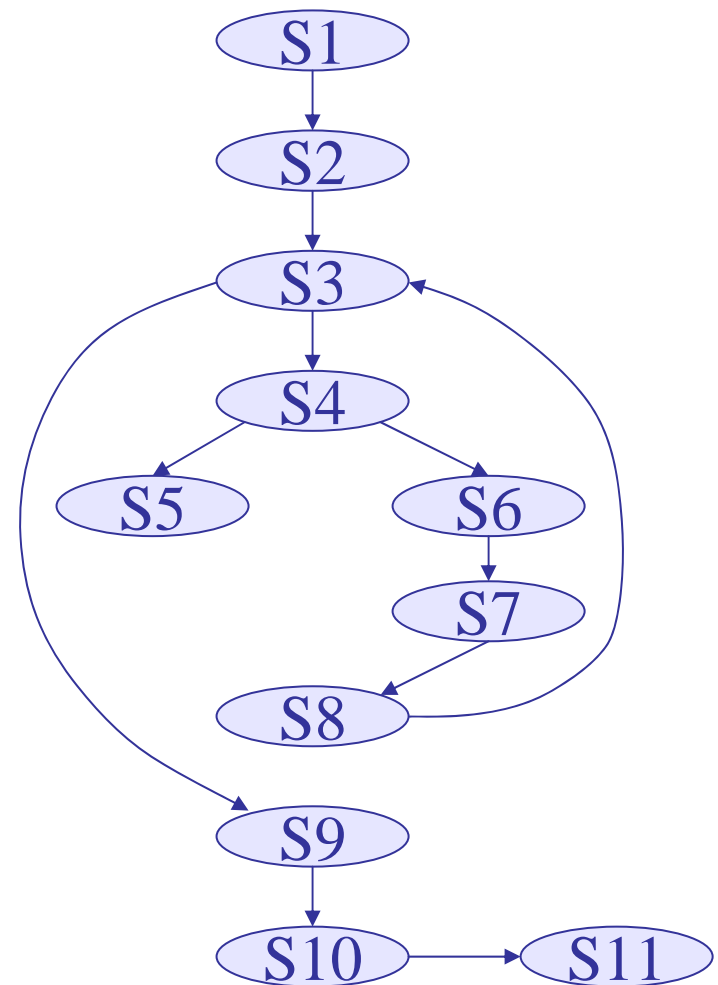
(2) Algorithms (or heuristics) for maximizing  $f$

# A Simple Technique:

## *Total Statement Coverage Prioritization*

test	stmts covered
t1	s1,s2,s3,s9,s10,s11
t2	s1,s2,s3,s4,s5
t3	s1,s2,s3,s4,s6,s7 s8,s9,s10,s11

st-total: **t3, t1, t2**



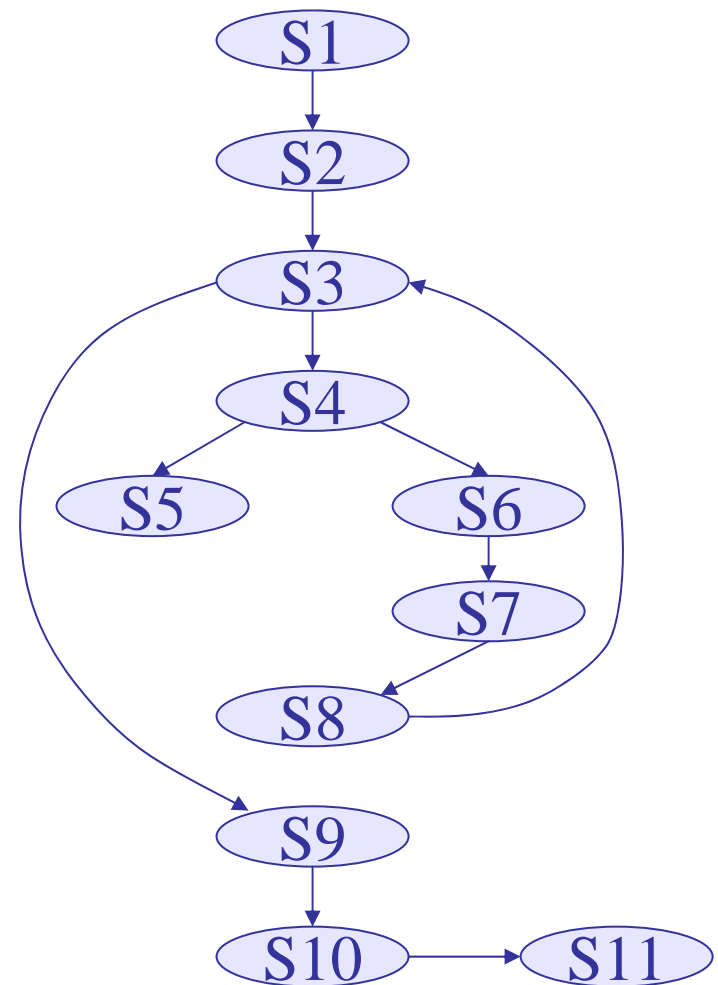
# Using Feedback:

## *Additional Statement Coverage Prioritization*

test	stmts covered
t1	s1,s2,s3,s9,s10,s11
t2	s1,s2,s3,s4,s5
t3	s1,s2,s3,s4,s6,s7 s8,s9,s10,s11

st-total: t3, t1, t2

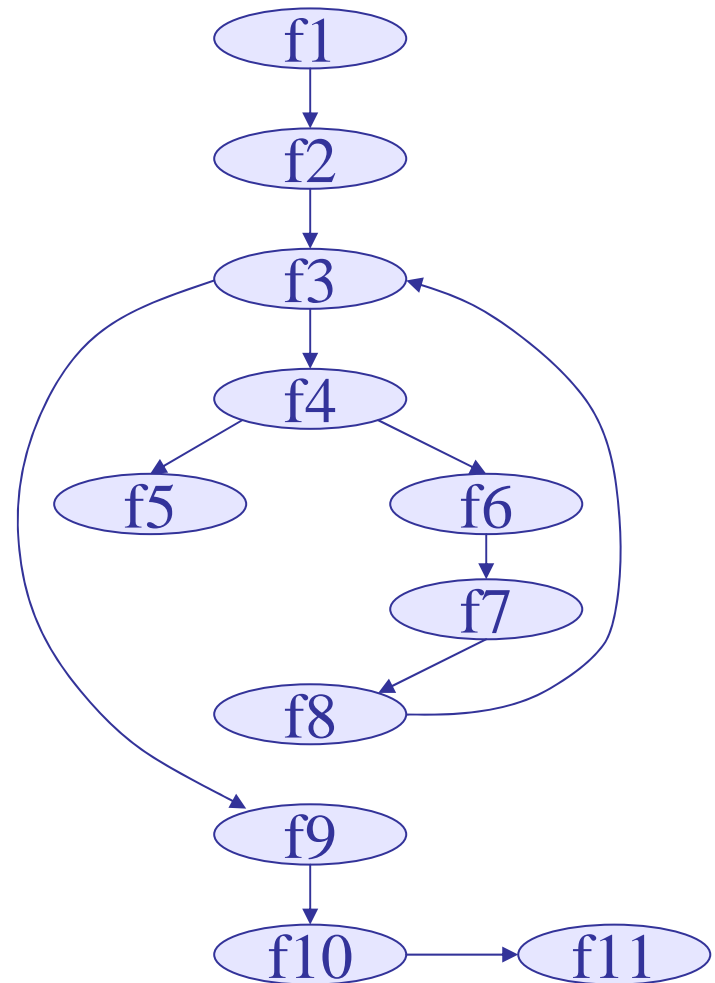
st-addtl: t3, t2, t1



# Using Feedback:

## *Total/Addt'l Function Coverage Prioritization*

test	functions covered
t1	f1,f2,f3,f9,f10,f11
t2	f1,f2,f3,f4,f5
t3	f1,f2,f3,f4,f6,f7 f8,f9,f10,f11



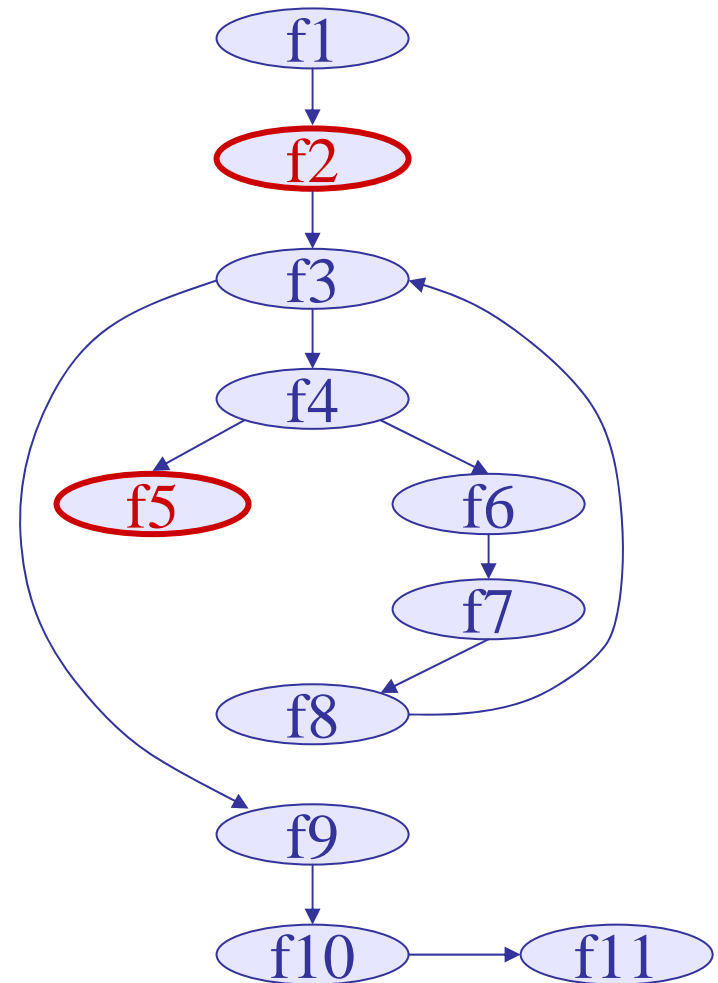
func-total: t3, t1, t2

func-addtl: t3, t2, t1

# Incorporating Modification Info:

*Total/Addt'l Modified Function Coverage Prio.*

test	functs covered
t1	f1, <b>f2</b> , f3, f9, f10, f11
t2	f1, <b>f2</b> , f3, f4, <b>f5</b>
t3	f1, <b>f2</b> , f3, f4, f6, f7 f8, f9, f10, f11



fn-mod-total: **t2**, t1, t3

fn-mod-addtl: **t2**, t1, t3

# Sources of Prioritization Data

- Code Coverage Data
- Modification information
- Test cost data
- Test criticality estimates
- History information

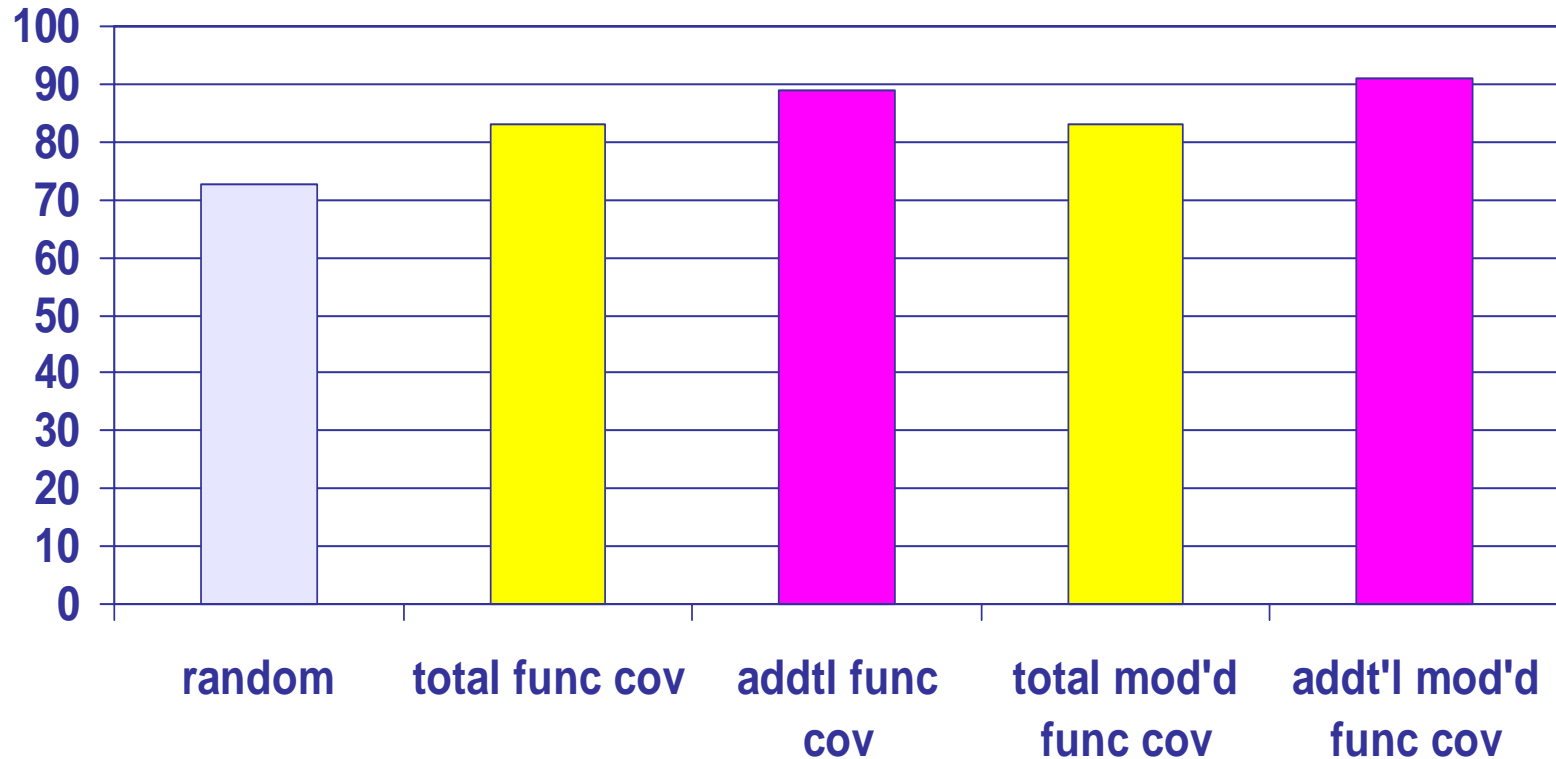
# Overview of Presentation

- Testing evolving software
- Regression test selection
  - Dejavu algorithm
  - Analytical and empirical evaluation
- Test case prioritization
  - Prioritization measures and techniques
  - Empirical results
- Ongoing and Future Work

# Case Study

- Empire program (60K + LOC)
- 11 sequential versions, several faults each
- 1 large functional test suite
- Various techniques
  - 1 control technique –random (avg of 20 runs)
  - function-level granularity
  - with/without feedback
  - with/without modification information

# Mean APFD Values for Empire



# Overview of Presentation

- Testing evolving software
- The test case prioritization problem
  - Measuring success
  - Prioritization techniques
- Empirical studies of test case prioritization
- Related work
- Ongoing work

# Ongoing Work

- Continued empirical assessment
- Investigate sources of variation in techniques
- Develop models for evaluating costs/benefits
- Develop methods for choosing techniques
- Identify implications for development and testing processes
- Develop process models and guidance mechanisms for practitioners

# An Evolution-Centric Perspective on Software Testing

- Focus on evolution first
- Harness evolution
- Design for regression testability

# Acknowledgments

- **Sponsors:**

- National Science Foundation
- Boeing Commercial Airplane Group
- Microsoft
- Lockheed Martin

- **Ph.D. Students:**

- Hyunsook Do
- Marc Fisher
- Jung-Min Kim (U. Md.)
- Jim Law
- Alexey Malishevsky
- Joe Ruthruff

- **Colleagues:**

- Sebastian Elbaum
- Mary Jean Harrold
- Alex Orso
- Adam Porter
- David Rosenblum
- Mary Lou Sofa
- Roland Untch
- Elaine Weyuker

# Software Testing: An Evolution-Centric Perspective

Gregg Rothermel



Dept. of Computer Science and Engineering  
University of Nebraska - Lincoln

Supported by the National Science Foundation,  
Microsoft, Lockheed Martin,  
and Boeing Commercial Aircraft Group

