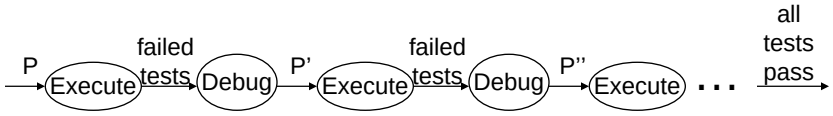
Class 20

- Fault localization (cont'd)
- Test-data generation
- Exam review: Nov 3, after class to 7:30
  - Responsible for all material up through Nov 3 (through test-data generation)
  - Send questions beforehand so all can prepare
- Exam: Nov 10
- Final project presentations: Dec 1, 3; 4:35-6:45
- Assign (see Schedule for links)
  - Problem Set 9 discuss
  - Readings

Fault Localization Using Tarantula

- What information does Tarantula use to compute suspicious (and ranking) of statements in the program?
- How is this information used?
- Are there other ways to compute the suspiciousness using this information?
- What information other than statement coverage could be used for fault localization?
- Do you think statement coverage would have worked for **tritype**?
- How could we use fault localization to identify which changes are most suspicious after a build?

Improving Fault-localization Efficiency



- Are all failing tests caused by the same fault?
- Can we associate groups of tests with different faults?
- Can we reduce debugging effort by considering these groups individually?
- Can we reduce debugging effort by considering these groups simultaneously?

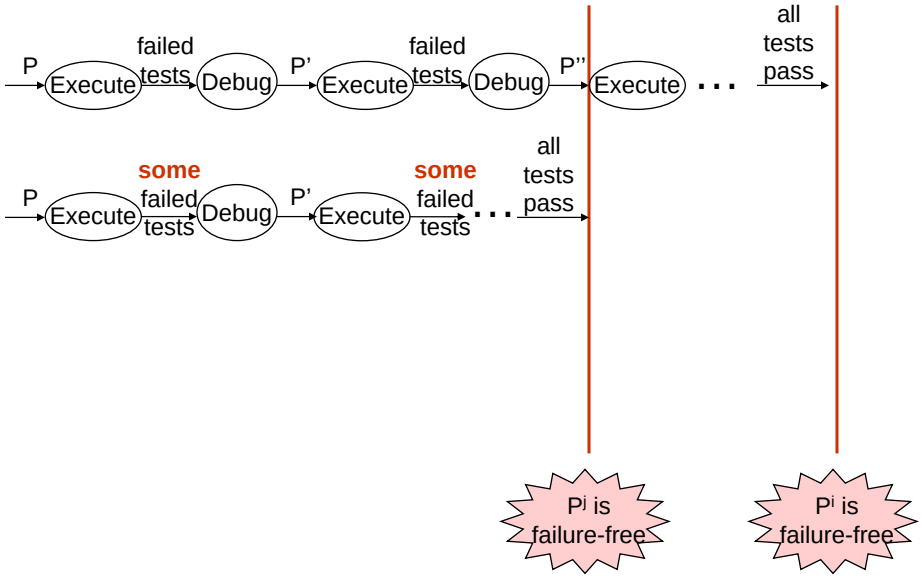
Improving Fault-localization Efficiency

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
`mid() {` `int x,y,z,m;`	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•
`2:m = z;`	•	•	•	•	•	•	•	•	•	•
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•
`4: if (x<y)`	•	•			•	•		•		•
`5: m = y;`		•								
`6: else if (x<z)`	•				•	•		•		•
`7: m = y;`	•				•			•		•
`8:else`			•	•			•		•	
`9: if (x>y)`			•	•			•		•	
`10: m = z;`			•				•		•	
`11: else if (x>z)`				•						
`12: m = x;`										
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F

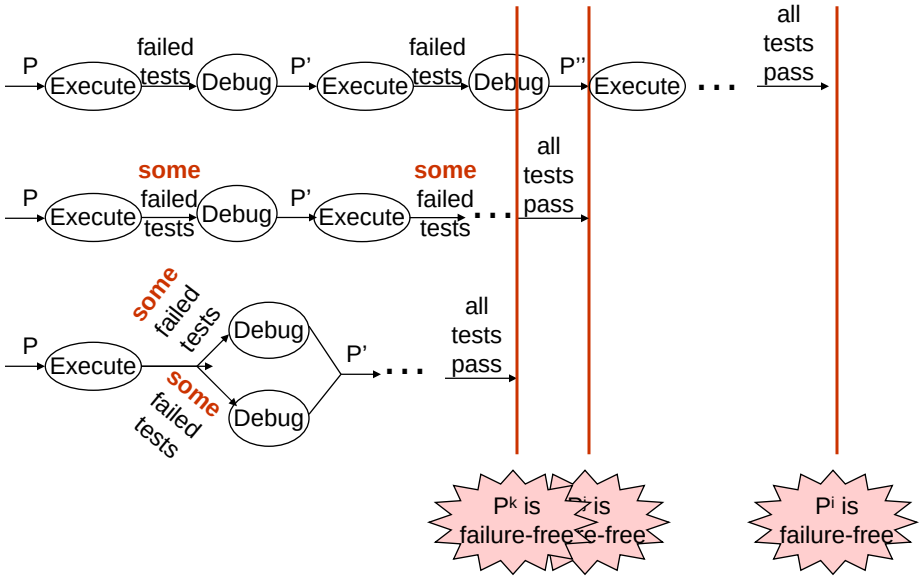
Improving Fault-localization Efficiency

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6
`mid() {`										
`  int x,y,z,m;`										
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•
`2:m = z;`	•	•	•	•	•	•	•	•	•	•
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•
`4:  if (x<y)`	•	•			•	•		•		•
`5:  m = y;`		•								
`6:  else if (x<z)`	•				•	•		•		•
`7:  m = y;`	•				•			•		•
`8:else`			•	•			•		•	
`9:  if (x>y)`			•	•			•		•	
`10:  m = z;`			•				•		•	
`11:  else if (x>z)`				•						
`12:  m = x;`										
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F

Debugging Process

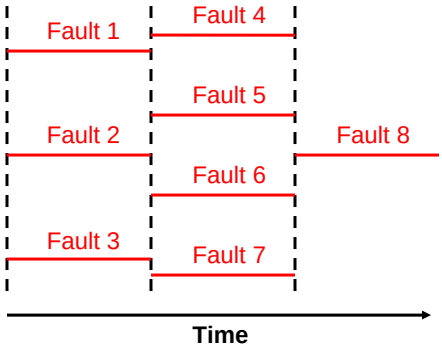


Debugging Process

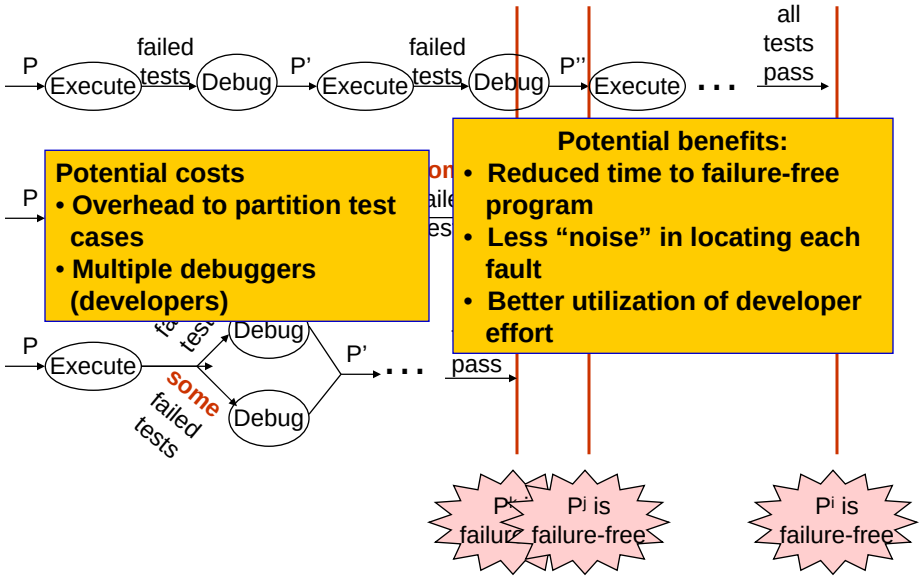


Hierarchy of Bugs

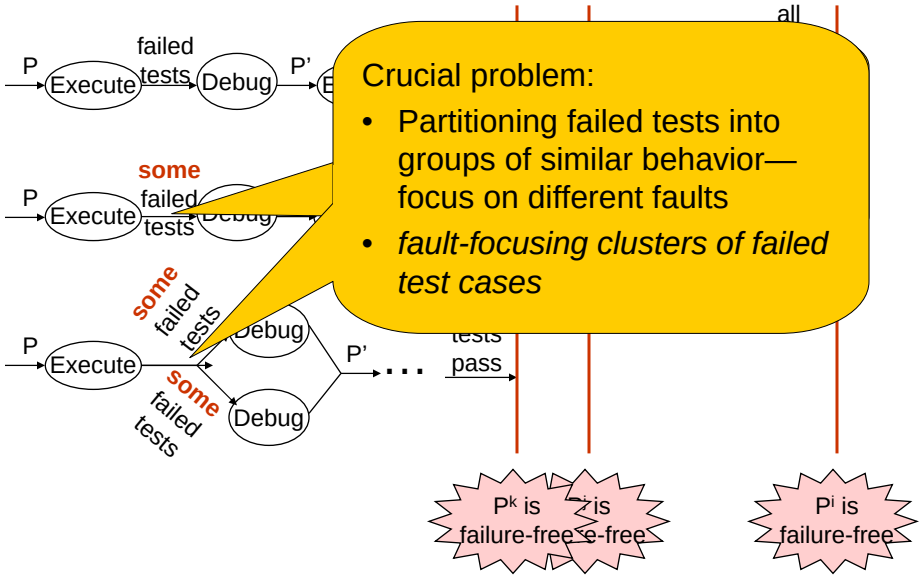
- Faults often dominate each other
- Failing test cases are first caused by a set of initial faults
- Once initial faults are fixed, other faults manifest themselves



Debugging Process



Debugging Process



Fault-focusing Clusters—Overview

Test Cases

t01 t02 t03 t04

t05 t06 t07 t08

t09 t10

t07

t09

t08

t10

Fault-focusing clusters:

- Clusters of failing test cases
- Clusters failing in similar way
- Each cluster targeting a different fault

Specialized Test Suites

Specialized test suites: Fault-focusing clusters combined with passing test cases

t07

t09

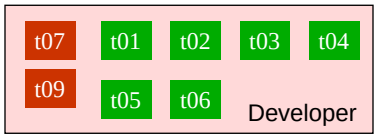
t08

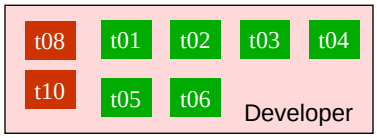
t10

Specialized Test Suites

Specialized test suites: Fault-focusing clusters combined with passing test cases

- Find faults one at a using specialized test suites

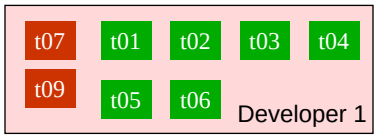


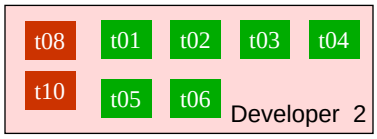


Specialized Test Suites

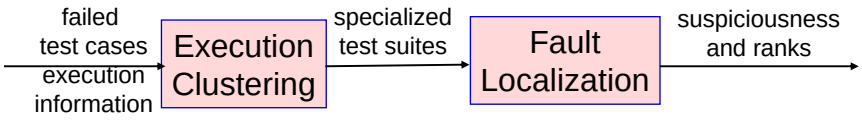
Specialized test suites: Fault-focusing clusters combined with passing test cases

- Find faults one at a using specialized test suites
- Find faults at the same time (in parallel) using specialized test suites

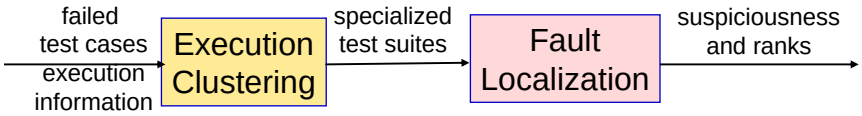




Fault-focusing Clusters



Fault-focusing Clusters



Clustering by behavior models

Dynamic information

- **profiles (branch, method-method, …)**
- **only failed tests**

Statistical analysis, machine learning

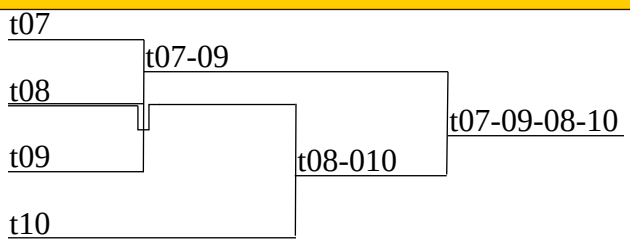
- **generate models for each execution**
- **cluster models**

Fault-localization for stopping point

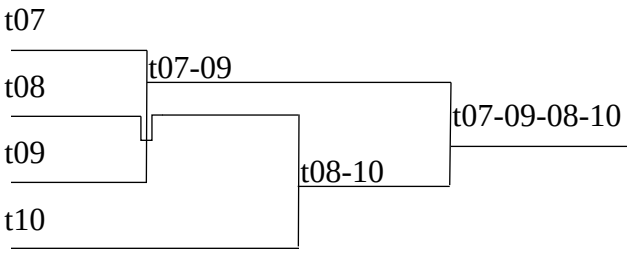
Clustering Behavior Models

- Most difficult problem of clustering is determining a good stopping criterion?
-

What is a good stopping point for the clustering for fault-focused clusters?

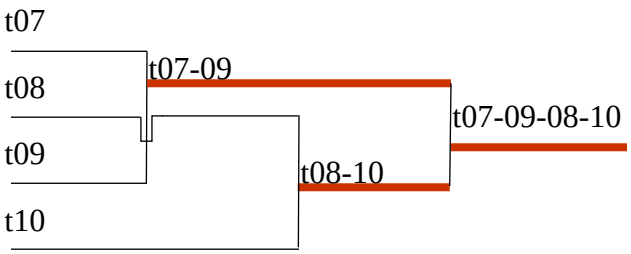


Fault Localization for Stopping Point



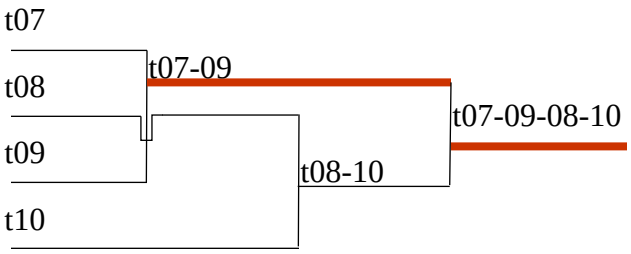
Fault Localization for Stopping Point

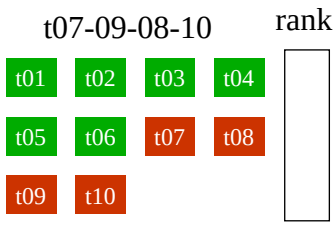
$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$



Fault Localization for Stopping Point

$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$



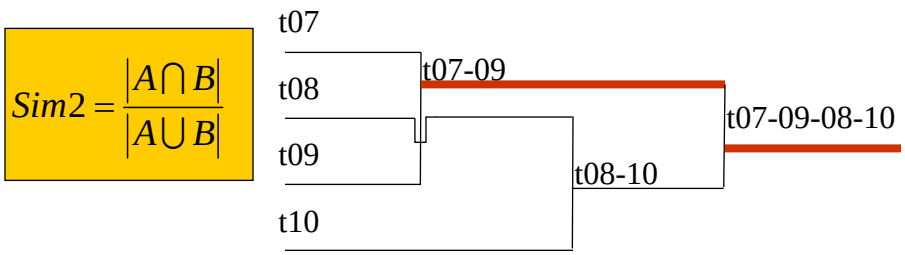


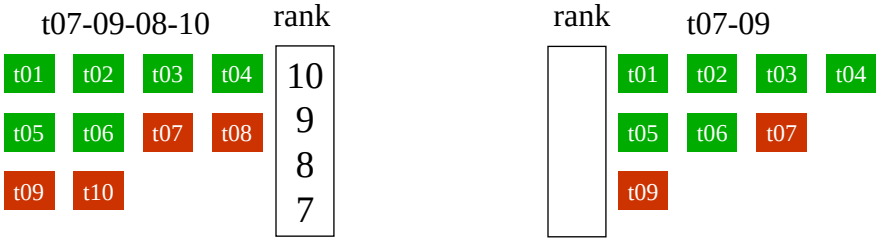
Tarantula: Fault Localization

	t1 3,3,5	t2 1,2,3	t3 3,2,2	t4 5,5,5	t5 1,1,4	t6 5,3,4	t7 3,2,1	t8 2,1,3	t9 5,4,2	t10 5,2,6	suspiciousness
`mid() {`											
`  int x,y,z,m;`											
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•	0.50
`2:m = z;`	•	•	•	•	•	•	•	•	•	•	0.50
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•	0.50
`4:  if (x<y)`	•	•			•	•		•		•	0.43
`5:  m = y;`		•									0.00
`6:  else if (x<z)`	•				•	•		•		•	0.50
`7:  m = y;  //bug`	•				•			•		•	0.60
`8:else`			•	•			•		•		0.60
`9:  if (x>y)`			•	•			•		•		0.60
`10:  m = z;  //bug`			•				•		•		0.75
`11:  else if (x>z)`				•							0.00
`12:  m = x;`											0.00
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	0.50
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F	

Fault Localization for Stopping Point

$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$



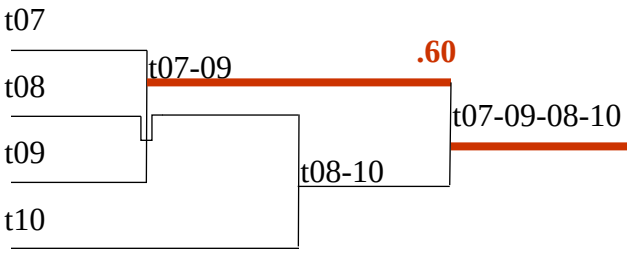


Fault-focusing Cluster 1

	t1 3,3,5	t2 1,2,3	t3 3,2,2	t4 5,5,5	t5 1,1,4	t6 5,3,4	t7 3,2,1	t8	t9 5,4,2	t10	suspiciousness
`mid() {`											
`  int x,y,z,m;`											
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•		•		0.50
`2:m = z;`	•	•	•	•	•	•	•		•		0.50
`3:if (y<z)`	•	•	•	•	•	•	•		•		0.50
`4:  if (x<y)`	•	•			•	•	•				0.00
`5:  m = y;`		•									0.00
`6:  else if (x<z)`	•				•	•					0.00
`7:  m = y;  //bug`	•				•						0.00
`8:else`			•	•			•		•		0.75
`9:  if (x>y)`			•	•			•		•		0.75
`10:  m = z;  //bug`			•				•		•		0.86
`11:  else if (x>z)`				•							0.00
`12:  m = x;`											0.00
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•		•		0.50
`}` Pass/fail Status	P	P	P	P	P	P	F		F		

Fault Localization for Stopping Point

$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$



t07-09-08-10

t01	t02	t03	t04
t05	t06	t07	t08
t09	t10		

rank

10
9
8
7

$$Sim2 = \frac{3}{5}$$

rank

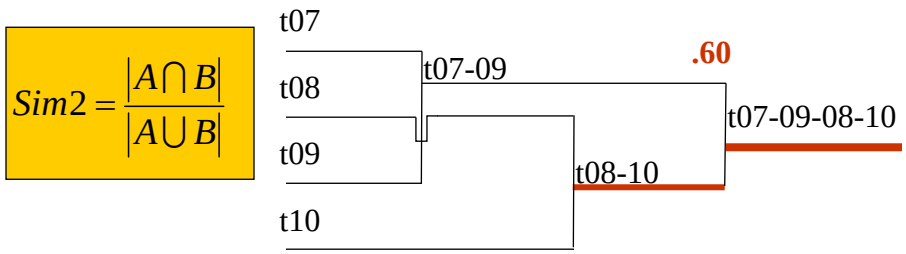
10
9
8
1

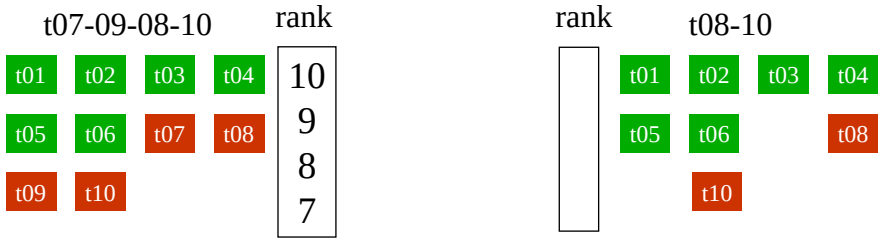
t07-09

t01	t02	t03	t04
t05	t06	t07	
t09			

Fault Localization for Stopping Point

$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$



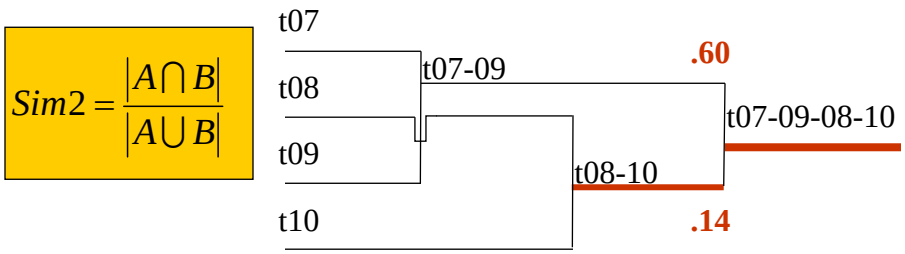


Fault-focusing Cluster 2

	t1 3,3,5	t2 1,2,3	t3 3,2,2	t4 5,5,5	t5 1,1,4	t6 5,3,4	t7	t8 2,1,3	t9	t10 5,2,6	suspiciousness
`mid() {`											
`  int x,y,z,m;`											
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•		•		•	0.50
`2:m = z;`	•	•	•	•	•	•		•		•	0.50
`3:if (y<z)`	•	•	•	•	•	•		•		•	0.50
`4:  if (x<y)`	•	•			•	•		•		•	0.60
`5:  m = y;`		•									0.00
`6:  else if (x<z)`	•				•	•		•		•	0.67
`7:  m = y;  //bug`	•				•			•		•	0.75
`8:else`			•	•							0.00
`9:  if (x>y)`			•	•							0.00
`10:  m = z;  //bug`			•								0.00
`11:  else if (x>z)`				•							0.00
`12:  m = x;`											0.00
`13:print("Middle number is:", m);`	•	•	•	•	•	•		•		•	0.50
`}` Pass/fail Status	P	P	P	P	P	P		F		F	

Fault Localization for Stopping Point

$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$



t07-09-08-10

rank

10
9
8
7

$$Sim2 = \frac{1}{7}$$

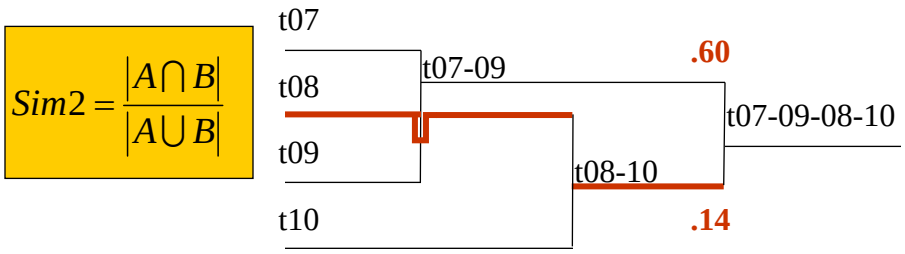
rank

7
6
4
1

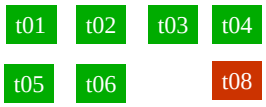
t08-10

Fault Localization for Stopping Point

$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$



t08



rank

7
6
4
1

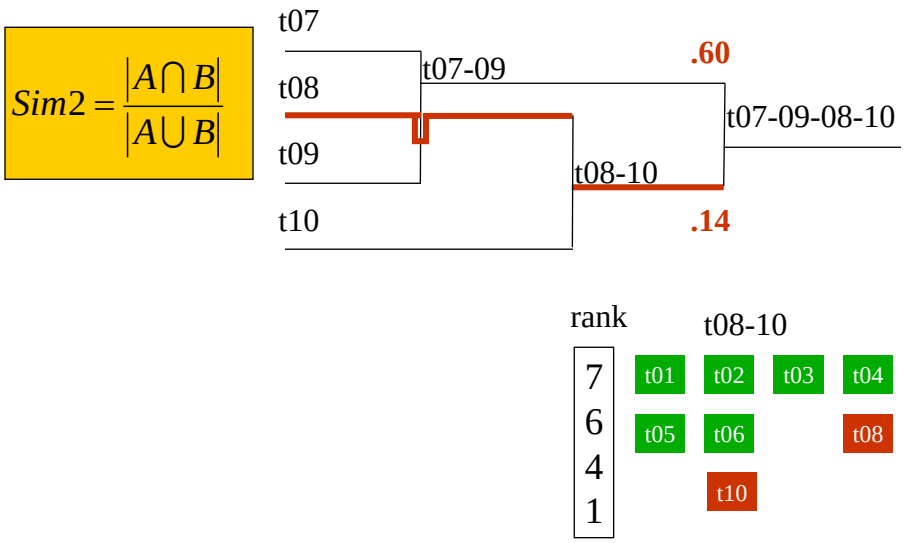
rank

7
6
4
1

t08-10

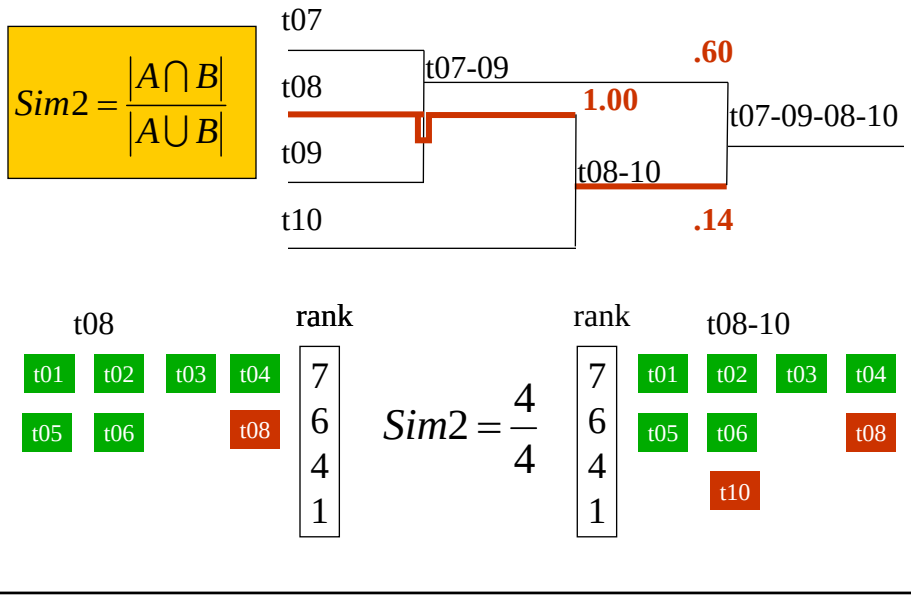
Fault Localization for Stopping Point

$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$



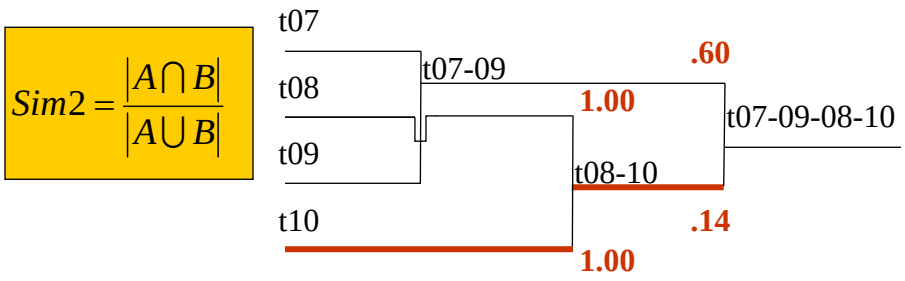
Fault Localization for Stopping Point

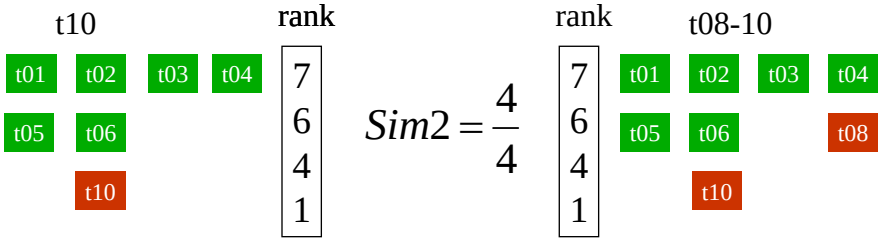
$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$



Fault Localization for Stopping Point

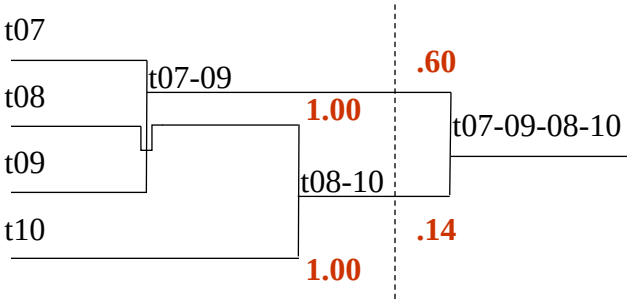
$$Sim2 = \frac{|A \cap B|}{|A \cup B|}$$





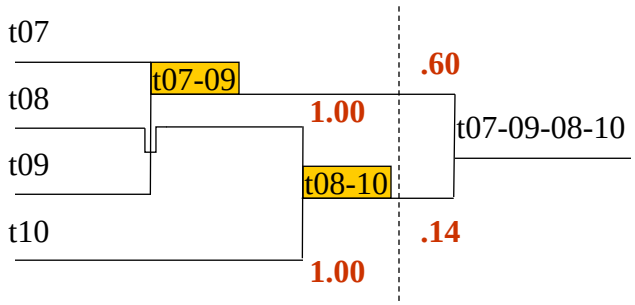
$$Sim2 = \frac{4}{4}$$

Fault Localization for Stopping Point



- Composite is similar (above threshold) to both of its constituents so clustering stops at this level
- Result is two clusters: {t07, t09}, {t08, t10}

Fault Localization for Stopping Point



- Composite is similar (above threshold) to both of its constituents so clustering stops at this level
- Result is two clusters: {t07, t09}, {t08, t10}

Fault-focusing Cluster 1

	t1 3,3,5	t2 1,2,3	t3 3,2,2	t4 5,5,5	t5 1,1,4	t6 5,3,4	t7 3,2,1	t8	t9 5,4,2	t10	suspiciousness
`mid() {`											
`  int x,y,z,m;`											
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•		•		0.50
`2:m = z;`	•	•	•	•	•	•	•		•		0.50
`3:if (y<z)`	•	•	•	•	•	•	•		•		0.50
`4:  if (x<y)`	•	•			•	•					0.00
`5:  m = y;`		•									0.00
`6:  else if (x<z)`	•				•	•					0.00
`7:  m = y;  //bug`	•				•						0.00
`8:else`			•	•			•		•		0.75
`9:  if (x>y)`			•	•			•		•		0.75
`10:  m = z;  //bug`			•				•		•		0.86
`11:  else if (x>z)`				•							0.00
`12:  m = x;`											0.00
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•		•		0.50
`}` Pass/fail Status	P	P	P	P	P	P	F		F		

Fault-focusing Cluster 2

	t1 3,3,5	t2 1,2,3	t3 3,2,2	t4 5,5,5	t5 1,1,4	t6 5,3,4	t7	t8 2,1,3	t9	t10 5,2,6	suspiciousness
`mid() {`											
`  int x,y,z,m;`											
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•		•		•	0.50
`2:m = z;`	•	•	•	•	•	•		•		•	0.50
`3:if (y<z)`	•	•	•	•	•	•		•		•	0.50
`4:  if (x<y)`	•	•			•	•		•		•	0.60
`5:  m = y;`		•									0.00
`6:  else if (x<z)`	•				•	•		•		•	0.67
`7:  m = y;  //bug`	•				•			•		•	0.75
`8:else`			•	•							0.00
`9:  if (x>y)`			•	•							0.00
`10:  m = z;  //bug`			•								0.00
`11:  else if (x>z)`				•							0.00
`12:  m = x;`											0.00
`13:print("Middle number is:", m);`	•	•	•	•	•	•		•		•	0.50
`}` Pass/fail Status	P	P	P	P	P	P		F		F	

Visualization of Specialized Test Suites

Cluster 1

```
mid() {
  int x,y,z,m;
1:read("Enter 3 integers:"…
2:m = z;
3:if (y<z)
4:   if (x<y)
5:      m = y;
6:   else if (x<z)
7:      m = y;        //bug
8:else
9:   if (x>y)
10:     m = z;        //bug
11:  else if (x>z)
12:     m = x;
13:print("Middle number is:"…
  }
```

Cluster 2

```
mid() {
  int x,y,z,m;
1:read("Enter 3 integers:"…
2:m = z;
3:if (y<z)
4:   if (x<y)
5:      m = y;
6:   else if (x<z)
7:      m = y;        //bug
8:else
9:   if (x>y)
10:     m = z;        //bug
11:  else if (x>z)
12:     m = x;
13:print("Middle number is:"…
  }
```

Empirical Study

Variables

NSTS: Finding faults using non-specialized test suites
STS-S: Finding faults using specialized test suites
STS-P: Finding faults using specialized test suites in parallel

Measures

D: total developer effort
FF: total effort to failure-free program

Subject

SPACE

- 6000 LOC
- 100 8-fault versions; > 1000 derivative versions)

Method

For each of 100 8-fault versions, debug until failure-free, using

- Non specialized test suite
- Specialized test suite both sequential and parallel

D: Total Developer Effort

- Using specialized test suites based on fault-focusing cluster is less expensive, on average, than not using specialized test suites
- Benefit holds when performing
  - fault localization sequentially (one developer)
  - fault localization in parallel (multiple developers)

FF: Total Effort to Failure-free

Sample Source	Sample mean	Sample standard deviation	99% confidence interval lower bound	99% confidence interval upper bound
FF_{NSTS}	36.26	22.86	30.83	41.69
FF_{STS-S}	26.16	22.58	20.80	31.53
FF_{STS-P}	18.29	14.00	14.96	21.62

Using specialized test suites and performing the fault localization sequential or in parallel can provide significant savings over using non specialized test suites

Summary of Results

For SPACE

- Using specialized test suites is usually less expensive than using non-specialized test suites
  - Total developer effort is reduced for both sequential and parallel modes
  - Time to a failure-free program is reduced without negatively affecting the total developer effort

Automatic Test Data Generation

Test Data Generation

Ferguson and Korel described three categories of test-data generation

What are they?

Test Data Generation

Ferguson and Korel described three categories of test-data generation

1. **Random**—randomly select from universe of inputs
2. **Goal-oriented**—select test data to execute a given entity (e.g., statement, branch, def-use pair) irrespective of the path taken
3. **Path-oriented**—select a program path and generate test data that will execute that path; path can be selected automatically or selected by the user