CSE 331
SOFTWARE DESIGN & IMPLEMENTATION
REGRESSION TESTING

Autumn 2011

Regression Testing

First slides edited from: Ammann & Offutt

- *The process of re-testing software that has been modified*
- Most software today has relatively little new development
  - Correcting, perfecting, adapting, or preventing problems with existing software
  - Composing new programs from existing components
  - Applying existing software to new situations
- Because of the deep interconnections among software components, changes in one method can cause problems in methods that seem to be unrelated
- Regression testing is intended to reduce the chance that existing properties are harmed by a change
- Large regression test suites may accumulate as programs age

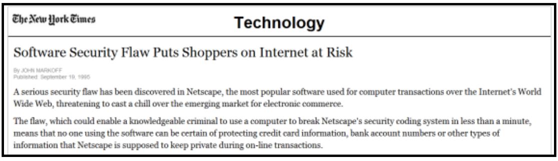
Automation and Tool Support

- Too many tests to be run by hand
- Tests must be run and evaluated quickly
  - often overnight, or more frequently for web applications
- Testers do not have time to view the results by inspection
- Types of tools include
  - Capture / Replay – Capture values entered into a GUI and replay those values on new versions
  - Version control – Keeps track of collections of tests, expected results, where the tests came from, the criterion used, and their past effectiveness
  - Scripting software – Manages the process of obtaining test inputs, executing the software, obtaining the outputs, comparing the results, and generating test reports
- Tools are plentiful and inexpensive (often free)

Managing Tests in a Regression Suite

- Test suites accumulate new tests over time
- Test suites are usually run in a fixed, short, period of time – often overnight, sometimes more frequently, sometimes less
- At some point, the number of tests can become unmanageable
  - We cannot finish running the tests in the time allotted
- We can always add more computer hardware
- But is it worth it? Does it solve the problem? How many of these tests really need to be run ?

Another related situation



The New York Times **Technology**

Software Security Flaw Puts Shoppers on Internet at Risk

By JOHN MARKOFF
Published: September 19, 1995

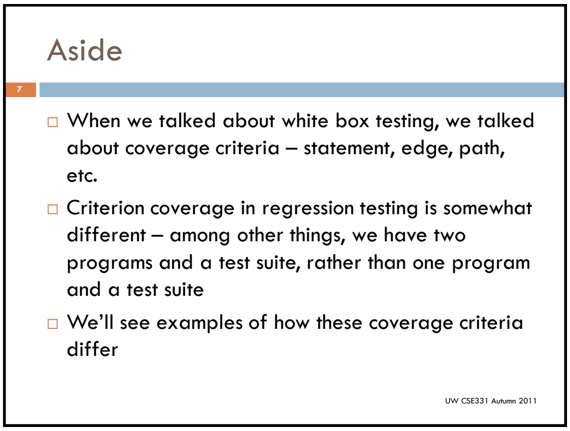
A serious security flaw has been discovered in Netscape, the most popular software used for computer transactions over the Internet's World Wide Web, threatening to cast a chill over the emerging market for electronic commerce.

The flaw, which could enable a knowledgeable criminal to use a computer to break Netscape's security coding system in less than a minute, means that no one using the software can be certain of protecting credit card information, bank account numbers or other types of information that Netscape is supposed to keep private during on-line transactions.

- When security flaws are made public, companies are under immense pressure to provide a fix very quickly
- Often full regression tests cannot be run in the given time – but running no regression tests isn't reasonable either

Policies for Updating Test Suites

- Which tests to keep can be based on several policies
  - Add a new test for every problem report
  - Ensure that a coverage criterion is always satisfied
- Sometimes harder to choose tests to remove
  - Remove tests that do not contribute to satisfying coverage
  - Remove tests that have never found a fault (risky !)
  - Remove tests that have found the same fault as other tests (also risky !)
- Reordering strategies
  - If a suite of N tests satisfies a coverage criterion, the tests can often be reordered so that the first N-k tests satisfies the criterion – so the remaining tests can be removed
  - This is often called test selection
  - If the criterion is approximated, not guaranteed, this is often called test prioritization

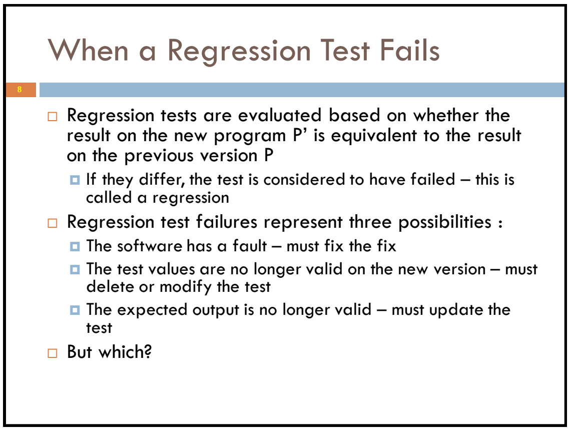


Aside

7

- When we talked about white box testing, we talked about coverage criteria – statement, edge, path, etc.
- Criterion coverage in regression testing is somewhat different – among other things, we have two programs and a test suite, rather than one program and a test suite
- We'll see examples of how these coverage criteria differ

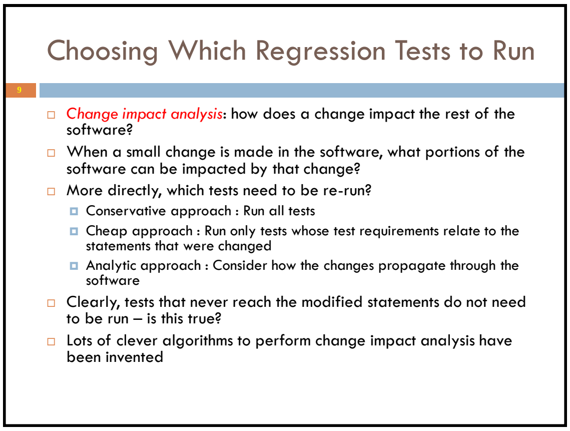
UW CSE331 Autumn 2011



When a Regression Test Fails

8

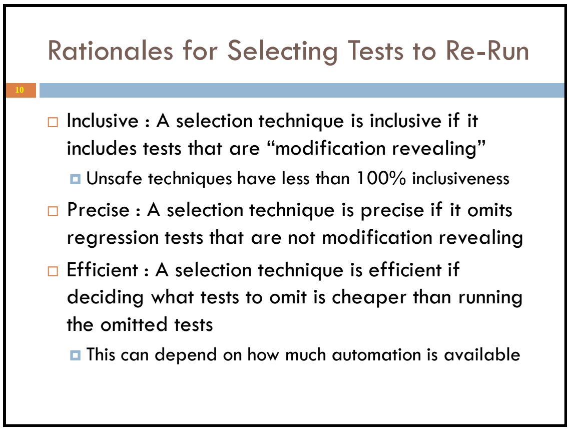
- Regression tests are evaluated based on whether the result on the new program P' is equivalent to the result on the previous version P
  - If they differ, the test is considered to have failed – this is called a regression
- Regression test failures represent three possibilities :
  - The software has a fault – must fix the fix
  - The test values are no longer valid on the new version – must delete or modify the test
  - The expected output is no longer valid – must update the test
- But which?



Choosing Which Regression Tests to Run

9

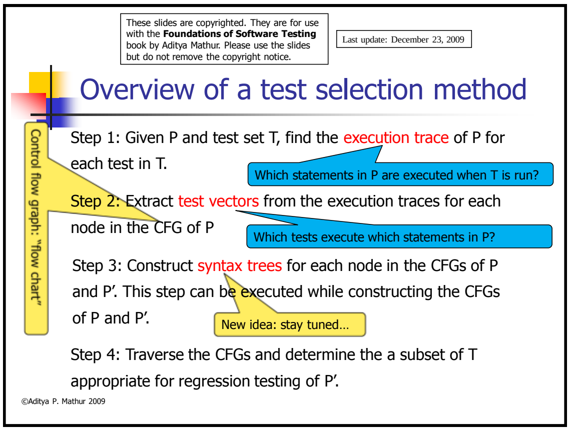
- *Change impact analysis*: how does a change impact the rest of the software?
- When a small change is made in the software, what portions of the software can be impacted by that change?
- More directly, which tests need to be re-run?
  - Conservative approach : Run all tests
  - Cheap approach : Run only tests whose test requirements relate to the statements that were changed
  - Analytic approach : Consider how the changes propagate through the software
- Clearly, tests that never reach the modified statements do not need to be run – is this true?
- Lots of clever algorithms to perform change impact analysis have been invented



Rationales for Selecting Tests to Re-Run

10

- Inclusive : A selection technique is inclusive if it includes tests that are "modification revealing"
  - Unsafe techniques have less than 100% inclusiveness
- Precise : A selection technique is precise if it omits regression tests that are not modification revealing
- Efficient : A selection technique is efficient if deciding what tests to omit is cheaper than running the omitted tests
  - This can depend on how much automation is available



These slides are copyrighted. They are for use with the **Foundations of Software Testing** book by Aditya Mathur. Please use the slides but do not remove the copyright notice.

Last update: December 23, 2009

Overview of a test selection method

Control flow graph: "flow chart"

Step 1: Given P and test set T, find the execution trace of P for each test in T.

Which statements in P are executed when T is run?

Step 2: Extract test vectors from the execution traces for each node in the CFG of P

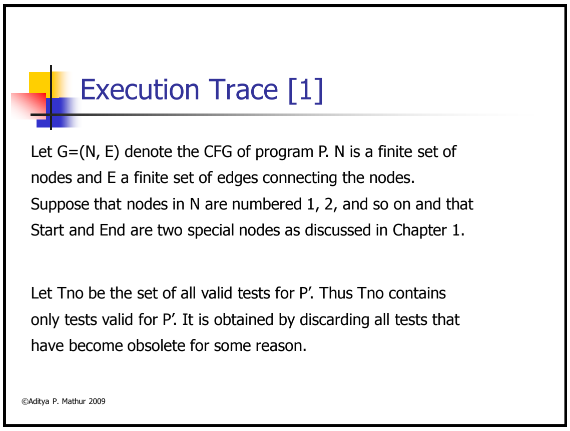
Which tests execute which statements in P?

Step 3: Construct syntax trees for each node in the CFGs of P and P'. This step can be executed while constructing the CFGs of P and P'.

New idea: stay tuned...

Step 4: Traverse the CFGs and determine the a subset of T appropriate for regression testing of P'.

©Aditya P. Mathur 2009



Execution Trace [1]

Let G=(N, E) denote the CFG of program P. N is a finite set of nodes and E a finite set of edges connecting the nodes. Suppose that nodes in N are numbered 1, 2, and so on and that Start and End are two special nodes as discussed in Chapter 1.

Let Tno be the set of all valid tests for P'. Thus Tno contains only tests valid for P'. It is obtained by discarding all tests that have become obsolete for some reason.

©Aditya P. Mathur 2009

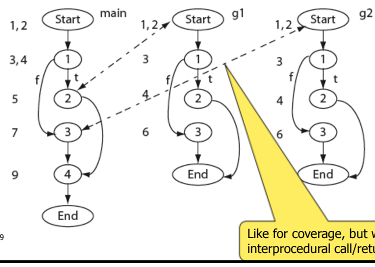
Execution Trace [2]

An execution trace of program P for some test t in Tno is the sequence of nodes in G traversed when P is executed against t. As an example, consider the following program.

```
 1  main(){         1  int g1(int a, b){ 1  int g2 (int a, b){
 2  int x,y,p;      2  int a,b;          2  int a,b;
 3  input (x,y);  3  if(a+ 1==b)       3  if(a==(b+1))
 4  if (x<y)        4    return(a*a);    4    return(b*b);
 5    p=g1(x,y);  5  else              5  else
 6  else            6    return(b*b);    6    return(a*a);
 7    p=g2(x,y);  7  }                 7  }
 8  endif
 9  output (p);
10  end
11  }
```

Execution Trace [3]

Here is a CFG for our example program.



Like for coverage, but with interprocedural call/return included

Execution Trace [4]

Now consider the following set of three tests and the corresponding trace.

$$T = \left\{ \begin{array}{l} t_1 :< x = 1, y = 3 > \\ t_2 :< x = 2, y = 1 > \\ t_3 :< x = 3, y = 1 > \end{array} \right\}$$

Test (t)	Execution trace ($trace(t)$)
t_1	main.Start, main.1, main.2, g1.Start, g1.1, g1.3, g1.End, main.2, main.4, main.End.
t_2	main.Start, main.1, main.3, g2.Start, g2.1, g2.2, g2.End, main.3, main.4, main.End.
t_3	main.Start, main.1, main.2, g1.Start, g1.1, g1.2, g1.End, main.2, main.4, main.End.

Test vector

A test vector for node n, denoted by test(n), is the set of tests that traverse node n in the CFG. For program P we obtain the following test vectors.

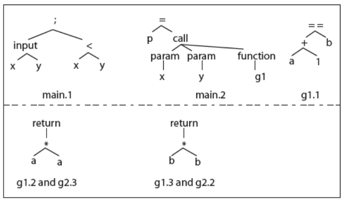
	Test vector ($test(n)$) for node n			
Function	1	2	3	4
main	t_1, t_2, t_3	t_1, t_3	t_2	t_1, t_2 t_3
g1	t_1, t_3	t_3	t_1	–
g2	t_2	t_2	None	–

test 1 and test 3 execute main

Syntax trees

Standard compiler representation: represent the program as a tree – produced by parsing the source code

A syntax tree is constructed for each node of CFG(P) and CFG(P'). Recall that each node represents a basic block. Here sample syntax trees for the example program.



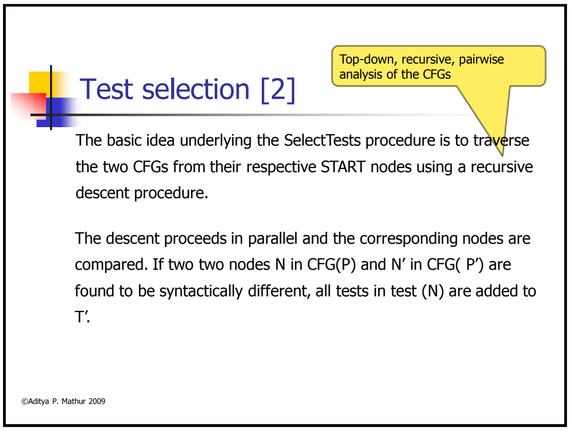
Test selection [1]

Given the execution traces and the CFGs for P and P', the following three steps are executed to obtain a subset T' of T for regression testing of P'.

Step 1 Set $T' = \emptyset$. Unmark all nodes in G and in its child CFGs.

Step 2 Call procedure `SelectTests` (G. Start, G'.Start'), where G.Start and G'.Start' are, respectively, the start nodes in G and G'.

Step 3 T' is the desired test set for regression testing P'.

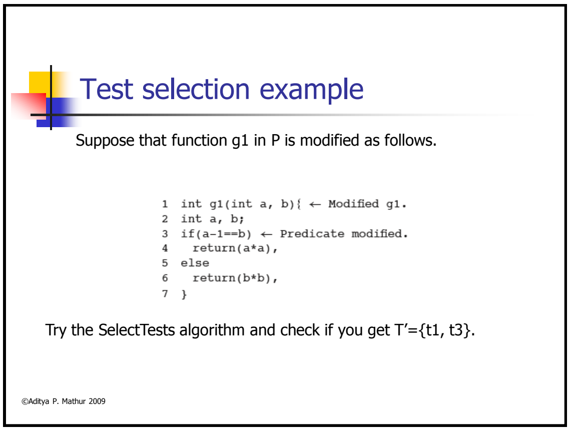


Test selection [2]

Top-down, recursive, pairwise analysis of the CFGs

The basic idea underlying the SelectTests procedure is to traverse the two CFGs from their respective START nodes using a recursive descent procedure.

The descent proceeds in parallel and the corresponding nodes are compared. If two two nodes N in CFG(P) and N′ in CFG(P′) are found to be syntactically different, all tests in test (N) are added to T′.

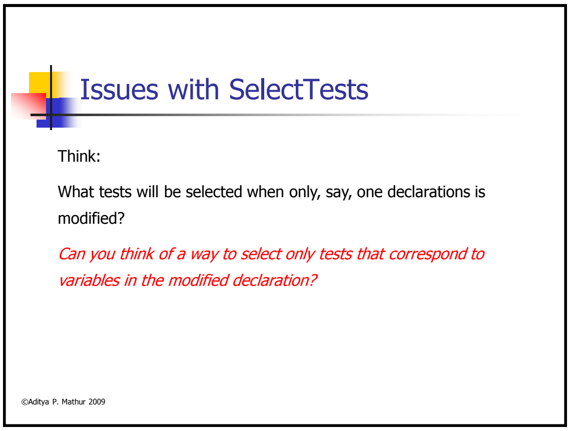


Test selection example

Suppose that function g1 in P is modified as follows.

```
int g1(int a, b){ ← Modified g1.
int a, b;
if(a-1==b) ← Predicate modified.
  return(a*a),
else
  return(b*b),
}
```

Try the SelectTests algorithm and check if you get T′={t1, t3}.

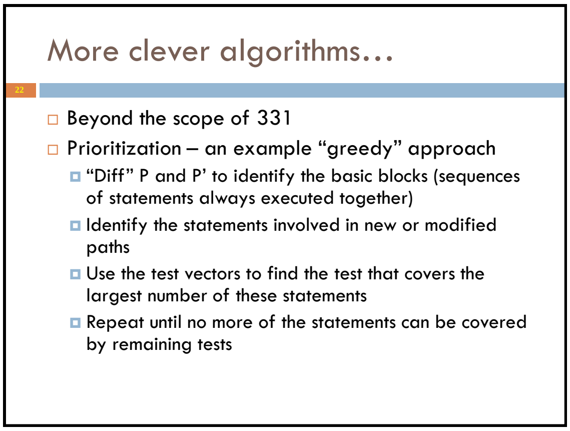


Issues with SelectTests

Think:

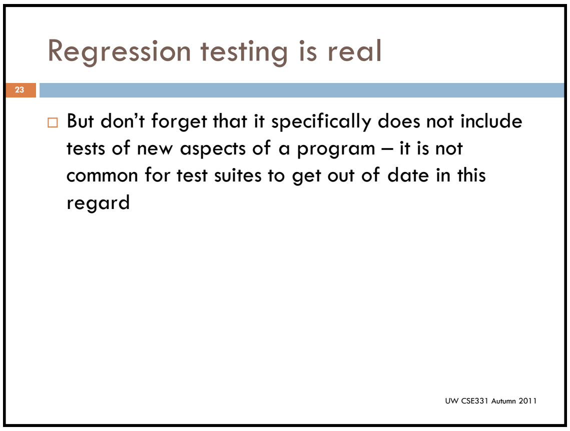
What tests will be selected when only, say, one declarations is modified?

Can you think of a way to select only tests that correspond to variables in the modified declaration?



More clever algorithms…

- Beyond the scope of 331
- Prioritization – an example "greedy" approach
  - "Diff" P and P' to identify the basic blocks (sequences of statements always executed together)
  - Identify the statements involved in new or modified paths
  - Use the test vectors to find the test that covers the largest number of these statements
  - Repeat until no more of the statements can be covered by remaining tests



Regression testing is real

- But don't forget that it specifically does not include tests of new aspects of a program – it is not common for test suites to get out of date in this regard