Reduce Cost of Regression Testing

W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
ewong@utdallas.edu
http://www.utdallas.edu/~ewong

Speaker Biographical Sketch

- Professor & Director of International Outreach
  Department of Computer Science
  University of Texas at Dallas
- Guest Researcher
  Computer Security Division
  National Institute of Standards and Technology (NIST)
- Vice President, IEEE Reliability Society
- Secretary, ACM SIGAPP (Special Interest Group on Applied Computing)
- Principal Investigator, NSF TUES (Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics) Project
  - *Incorporating Software Testing into Multiple Computer Science and Software Engineering Undergraduate Courses*
- Founder & Steering Committee co-Chair for the SERE conference (*IEEE International Conference on Software Security and Reliability*) (http://paris.utdallas.edu/sere13)

Outline

- What is regression testing?
- How to select a subset of tests for regression testing?
  - Modification-based test selection
  - Coverage-based test selection
    - Test set minimization
    - Test case prioritization
  - Risk analysis-based test selection

Regression Testing (1)

Version 1	Version 2
1. Develop *P*	4. Modify *P* to *P′*
2. Test *P*	5. Test *P′* for new functionality or bug fixing
3. Release *P*	6. Perform regression testing on *P′* to ensure that the code carried over from *P* behaves correctly
	7. Release *P′*

May need to generate additional new test cases to test the enhancement

Regression Testing (2)

- *Small changes in one part of a program may have subtle undesired effects in other seemingly unrelated parts of the program.*
  - Does fixing introduce new bugs?
  - Revalidate the functionalities inherited from the previous release

- Consequences of poor regression testing
  - Thousands of 800 numbers disabled by a poorly tested software upgrade (December 1991)
  - Fault in an SS7 software patch causes extensive phone outages (June 1991)
  - Fault in a 4ESS upgrade causes massive breakdown in the AT&T network (January 1990)

AT&T Network Outage, January 1990 **(1)**

- At 2:20PM on January 15, 1990, the 75 screens displaying a giant map of the United States at the AT&T operation center in New Jersey began to suddenly display red lines stretching from one switch to another, cascading across the wall. The entire country was soon covered in a series of red lines, representing switches that were now offline.
- Only 50% of calls placed through AT&T were connected, the other half heard a prerecorded message saying, "Sorry, all circuits are busy now."
- The network remained down until a team of 100 telephone technicians discovered and corrected the problem at 11:30 that night.
- AT&T carried 70% of the nation's telephone traffic routing over 115 million telephone calls on an average day

AT&T Network Outage, January 1990 **(2)**

```
While (ring receive buffer | empty and side buffer | empty)
{
Initialize pointer to first message in side buffer or ring received buffer
Get a copy of buffer
Switch (message) {
Case incoming message: if (sending switch = out of service)
        {
        if (ring write buffer = empty)
        Send in service to states map manager;
        Else
        Break;
        }
Process incoming message, set up pointers to optional parameters
        Break;
        :
        }
Do optional parameter work
}
```

Bug! (points to line "Break;" after Else)

Outline

- What is regression testing?
- How to select a subset of tests for regression testing?
  - Modification-based test selection
  - Coverage-based test selection
    - Test set minimization
    - Test case prioritization
  - Risk analysis-based test selection

Static & Dynamic Slice

- A *static slice* for a given variable at a given statement contains all the executable statements that could possibly affect the value of this variable at the statement.
  - Example: a static slice treats an entire array as a single variable
  - Advantage: easy to implement
  - Disadvantage: can be unnecessarily large with too much code
- A *dynamic slice* can be considered as a refinement of the corresponding static slice by excluding those statements in the program that do not have an impact on the variables of interest.
  - Different types of dynamic slices
  - Example: a dynamic slice treats every array element as a separate variable
  - Advantage: size is much smaller
  - Disadvantage: construction is in general time-consuming

1: if ($a \leq 0$)
2: $x = y + 1$;
3: else
4: $x = y - 1$;

- **Static Slice: 1, 2, 4**
- **Dynamic Slice with respect to variable x at line 4 for input ($a = 1, y = 3$): 1, 4**

Execution Slice (1)

- An execution slice with respect to a given test case contains the set of code executed by this test.

- We can also represent an execution slice as a set of blocks, decisions, c-uses, or p-uses, respectively, with respect to the corresponding block, decision, c-use, or p-use coverage criterion

Execution Slice (2)

- The dynamic slice with respect to the output variables includes only those statements that are not only executed but also have an impact on the program output under that test.
- Since not all the statements executed might have an impact on the output variables, an execution slice can be a super set of the corresponding dynamic slice.
- No inclusion relationship between static and execution slices

```
int sum, min, count, average;
sum = 0;            <--
min = -1;
read(count);
for (int i = 1; i <= count; i++) {
   read(num);
   sum += num;
   if (num < min) {
      min = num;
   }
}
average = sum/count;
write(min);         <--
 write(average);
```

– The first statement, *sum = 0*, will be included in the execution slice *with respect to min* but *not* in the corresponding static slice (nor the dynamic slice) because this statement does not affect the value of *min*.

– An execution slice can be constructed very easily if we know the coverage of the test because the execution slice with respect to a test case can be obtained simply by converting the coverage data collected during the testing into another format, i.e., instead of reporting the coverage percentage, it reports which parts of the program (in terms of *basic blocks*, *decisions*, *c-uses*, and *p-uses*) are covered.

An Example (1)

Which tests should be re-executed?

Test case	Input			Output	
	a	b	c	class	area
T_1	2	2	2	equilateral	1.73
T_2	4	4	3	isosceles	5.56
T_3	5	4	3	right	6.00
T_4	6	5	4	scalene	9.92
T_5	3	3	3	equilateral	3.90
T_6	4	3	3	scalene	4.47

Failure!

Quiz: Should T_6 be selected?

An Example (2)

A patch is installed

```
read (a, b, c);
class = scalene;
if a = b || b = c
    class = isosceles;
if a*a = b*b + c*c
    class = right;
if a = b && b = c
    class = equilateral;
case class of
    right       :  area = b*c / 2;
    equilateral :  area = a*a * sqrt(3)/4;
    otherwise   :  s = (a+b+c)/2;
                   area = sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Patch Applied

An Example (3)

Execution Slice w.r.t. the Successful Test T_2 = (4 4 3)

```
read (a, b, c);
class = scalene;
if a = b || b = c
    class = isosceles;
if a*a = b*b + c*c
    class = right;
if a = b && b = c
    class = equilateral;
case class of
    right        :  area = b*c / 2;
    equilateral  :  area = a*a * sqrt(3)/4;
    otherwise    :  s = (a+b+c)/2;
                    area = sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Patch is outside the execution slice!

Quiz: Should T_2 be selected?

An Example (4)

Execution Slice w.r.t. the Successful Test T_4 = (6 5 4)

```
read (a, b, c);
class = scalene;
if a = b || b = c
    class = isosceles;
if a*a = b*b + c*c
    class = right;
if a = b && b = c
    class = equilateral;
case class of
    right       : area = b*c / 2;
    equilateral : area = a*a * sqrt(3)/4;
    otherwise   : s = (a+b+c)/2;
                  area = sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```

Patch is in the execution slice!

Quiz: Should T_4 be selected?

An Example (5)

Which tests should be re-executed? (cont'd)

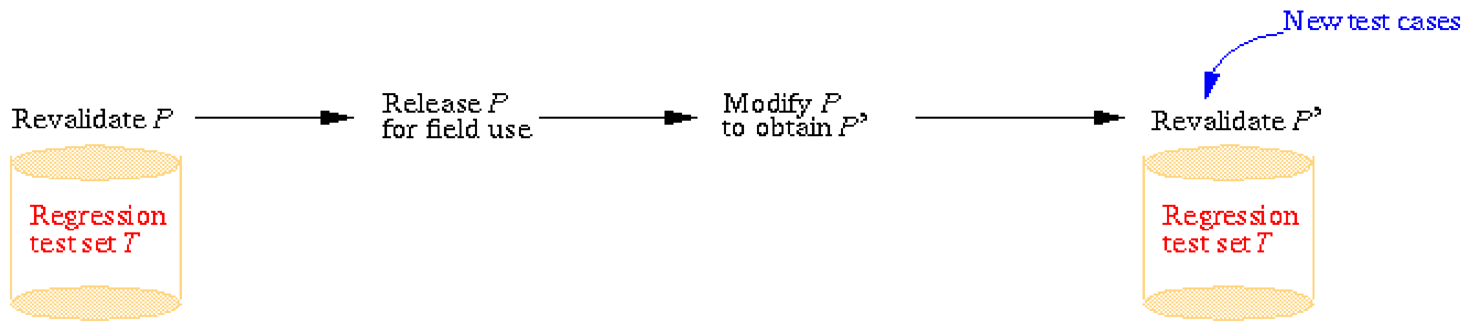
Test case	Input			Output	
	a	b	c	class	area
T_1	2	2	2	equilateral	1.73
T_2	4	4	3	isosceles	5.56
T_3	5	4	3	right	6.00
T_4	6	5	4	scalene	9.92
T_5	3	3	3	equilateral	3.90
T_6	4	3	3	isosceles	4.47

Passed!

Quiz: What if still too many tests?

How to Select Regression Tests (1)

- Traditional approach: *select all* (Too Expensive)



  – The test-all approach is good when you want to be certain that the new version works on all tests developed for the previous version.
  – What if you only have limited resources to run tests and have to meet a deadline?

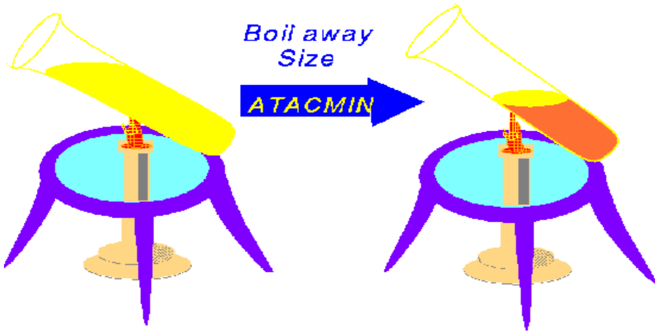
- Those on which the new and the old programs *produce different outputs* (Undecidable)

How to Select Regression Tests (2)

Select a subset (T_{sub}) of the original test set such that successful execution of the modified code (*P'*) against T_{sub} implies that all the functionality carried over from the original code to *P'* is still intact.

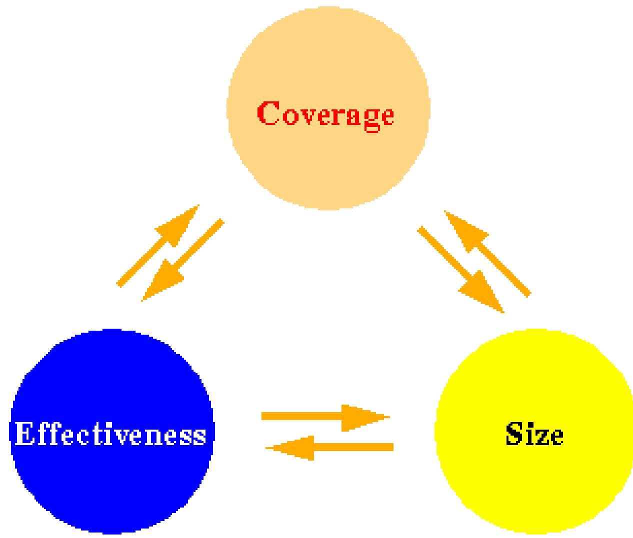
- Modification-based test selection
  - Those which *execute some modified code*
    - Still too many
    - Need to further reduce the number of regression tests

- Coverage-based test selection
  - Those selected based on *Test Set Minimization* and *Test Case Prioritization*



♣ **Coverage** ⟶ **Same**
♣ **Size** ⟶ **Reduced Significantly**

Three Attributes of a Test Set



- Is a larger test set likely to be more effective in revealing program faults than a smaller of equal coverage ?
- Is a higher coverage test set likely to be more effective than one of lower coverage but the same size ?
- Need a better understanding of the relationship among a test set's size, its code coverage, and its fault detection effectiveness

Coverage, Size, & Effectiveness

- Higher coverage → Better fault detection
- Bigger size → Better fault detection

Coverage and effectiveness are more correlated than size and effectiveness

Is Test Set Minimization Affordable in Practice?

- The minimization algorithm can be exponential in time
  - Does not occur in our experience
    - Some examples
      - an object-oriented language compiler (100 KLOC)
      - a provisioning application (353 KLOC) with 32K regression tests
      - a database application with 50 files (35 KLOC)
      - a space application (10 KLOC)
  - Stop after a pre-defined number of iterations
  - Obtain an approximate solution by using a greedy heuristic

Greedy Algorithm for Test Set Minimization

- Select each test case whose cost is *zero*
  - The complexity is order of *n* where *n* is the number of test cases

- For the remaining test cases
  - If the minimized subset has the same coverage as the original test set, STOP
  - Select the one that gives the *maximal coverage increment per unit cost*
  - Add this test case to the minimized subset
  - Go back to the beginning of this step
  - The complexity for the worst case scenario is order of n^2

Test Set Minimization (1)

Coverage & Cost per Test Case

```
$ atac  pK  main.atac wc.atac wordcount.trace
```

cost	% blocks	% decisions	% C Uses	% P Uses	test
120	69(35/51)	57(20/35)	43(39/90)	68(21/31)	wordcount.1
50	16(8/51)	11(4/35)	8(7/90)	6(2/31)	wordcount.2
20	53(27/51)	49(17/35)	23(21/90)	58(18/31)	wordcount.3
10	18(9/51)	11(4/35)	9(8/90)	13(4/31)	wordcount.4
40	31(16/51)	26(9/35)	18(16/90)	13(4/31)	wordcount.5
60	69(35/51)	60(21/35)	52(47/90)	71(22/31)	wordcount.6
80	14(7/51)	11(4/35)	7(6/90)	6(2/31)	wordcount.7
20	75(38/51)	66(23/35)	48(43/90)	68(21/31)	wordcount.8
10	75(38/51)	66(23/35)	48(43/90)	68(21/31)	wordcount.9
70	61(31/51)	60(21/35)	30(27/90)	61(19/31)	wordcount.10
50	61(31/51)	60(21/35)	30(27/90)	61(19/31)	wordcount.11
50	61(31/51)	60(21/35)	30(27/90)	61(19/31)	wordcount.12
50	27(14/51)	20(7/35)	16(14/90)	13(4/31)	wordcount.13
40	20(10/51)	14(5/35)	11(10/90)	6(2/31)	wordcount.14
60	69(35/51)	60(21/35)	41(37/90)	71(22/31)	wordcount.15
20	53(27/51)	26(9/35)	38(34/90)	32(10/31)	wordcount.16
150	69(35/51)	54(19/35)	44(40/90)	68(21/31)	wordcount.17
900	100(51)	100(35)	98(88/90)	100(31)	-- all --

coverage increment per cost = 38 blocks/10

Test Set Minimization (2)

Minimization w.r.t. Block Coverage

```
$ atac -M -mb main.atac wc.atac wordcount.trace

% blocks        test
------------- -------------
75(38/51)       wordcount.9
53(27/51)       wordcount.3
20(10/51)       wordcount.14
31(16/51)       wordcount.5
100(51)         == all ==


$ atac -M -mb -q -K main.atac wc.atac wordcount.trace

cost    % blocks        test
(cum)   (cumulative)
------ ------------- -------------
10      75(38/51)       wordcount.9
30      86(44/51)       wordcount.3
70      94(48/51)       wordcount.14
110     100(51)         wordcount.5
```

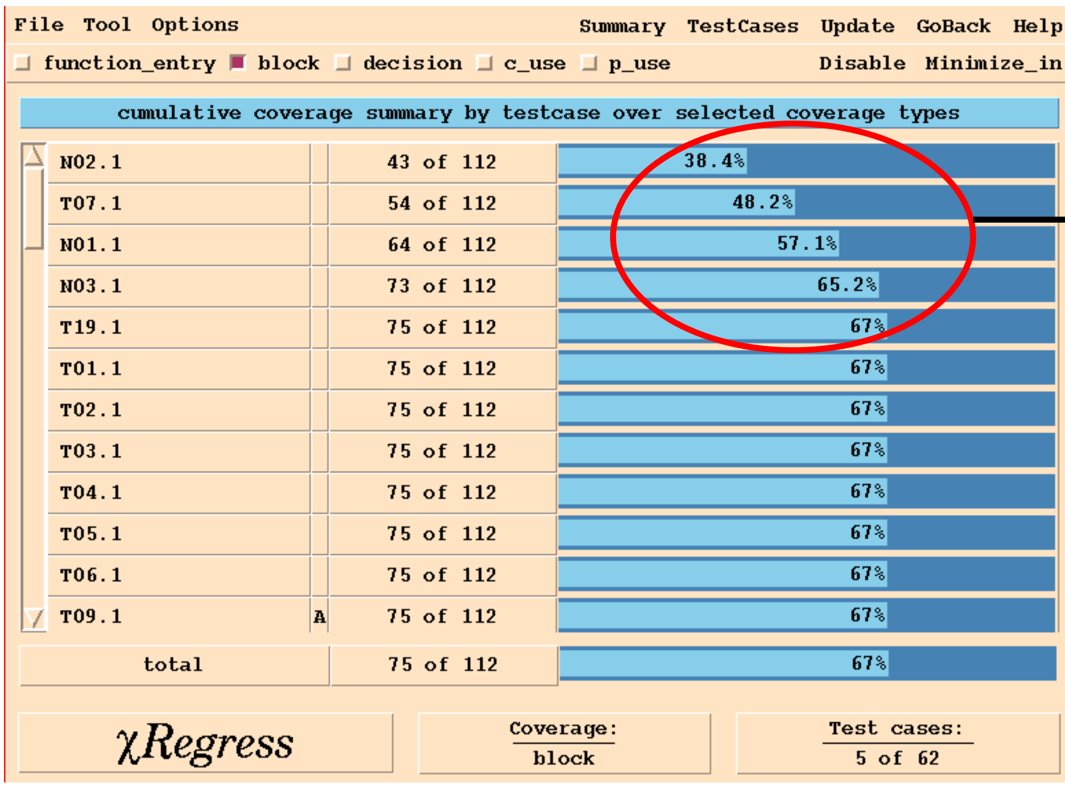
Test Set Minimization (3)

Minimization w.r.t. Block and Decision Coverage

```
$ atac -M -mbd main.atac wc.atac wordcount.trace

% blocks      % decisions   test
------------- ------------- -------------
75(38/51)     66(23/35)     wordcount.9
53(27/51)     49(17/35)     wordcount.3
20(10/51)     14(5/35)      wordcount.14
69(35/51)     60(21/35)     wordcount.15
61(31/51)     60(21/35)     wordcount.12
14(7/51)      11(4/35)      wordcount.7
100(51)       100(35)       == all ==


$ atac  M  mbd  q  K main.atac wc.atac wordcount.trace

cost   % blocks      % decisions   test
(cum)  (cumulative)  (cumulative)
------ ------------- ------------- -------------
10     75(38/51)     66(23/35)     wordcount.9
30     86(44/51)     77(27/35)     wordcount.3
70     94(48/51)     83(29/35)     wordcount.14
130    98(50/51)     91(32/35)     wordcount.15
180    100(51)       97(34/35)     wordcount.12
260    100(51)       100(35)       wordcount.7
```

Test Set Minimization (4)

- Sort test cases in order of increasing cost per additional coverage



Only 5 of the 62 test cases are included in the minimized subset which has the same block coverage as the original test set.

Test Set Minimization (5)

- How to guarantee the *inclusion* of a certain test?
  - Assign a very *low* cost to that test
- How to guarantee the *exclusion* of a certain test?
  - Assign a very *high* cost to that test
  - Some tests might become obsolete when P is modified to P'.
  - Such tests should not be included in the regression subset.

Test Set Minimization (6)

Include wordcount.10 in the Minimized Set

```
$ atactm -n wordcount.10 -c 0 wordcount.trace

$ atac -M -mb main.atac wc.atac wordcount.trace

% blocks        test
--------------  --------------
61(31/51)       wordcount.10
75(38/51)       wordcount.9
53(27/51)       wordcount.3
31(16/51)       wordcount.5
20(10/51)       wordcount.14
100(51)         == all ==

$ atac -M -q -mb main.atac wc.atac wordcount.trace

% blocks        test
(cumulative)
--------------  --------------
61(31/51)       wordcount.10
84(43/51)       wordcount.9
88(45/51)       wordcount.3
94(48/51)       wordcount.5
100(51)         wordcount.14
```

Test Set Minimization (7)

Exclude wordcount.9 in the Minimized Set

```
$ atactm -n wordcount.9 -c 1000 wordcount.trace

$ atac -M -mb main.atac wc.atac wordcount.trace

% blocks        test
-------------   -------------
75(38/51)       wordcount.8
53(27/51)       wordcount.3
31(16/51)       wordcount.5
20(10/51)       wordcount.14
100(51)         == all ==

$ atac -M -q -mb main.atac wc.atac wordcount.trace

% blocks        test
(cumulative)
-------------   -------------
75(38/51)       wordcount.8
86(44/51)       wordcount.3
94(48/51)       wordcount.5
100(51)         wordcount.14
```

Test Set Minimization (8)

- Is it reasonable to apply coverage-based criteria as a filter to reduce the size of a test set ?
  - Recall that coverage and effectiveness are more correlated than size and effectiveness
- Yes, it is
  - Test cases that do not add coverage are likely to be ineffective in revealing more program faults
  - Test set minimization can be used to reduce the cost of regression testing

Test Case Prioritization (1)

- Sort test cases in order of *increasing cost per additional coverage*
- Select the first test case
- Repeat the above two steps until *n* test cases are selected

Test Case Prioritization (2)

- Individual decision coverage and cost per test case

```
$ atac -K -md main.atac wc.atac wordcount.trace

 cost   % decisions   test
 ------ ------------- -------------
 120    57(20/35)     wordcount.1
 50     11(4/35)      wordcount.2
 20     49(17/35)     wordcount.3
 10     11(4/35)      wordcount.4
 40     71(25/35)     wordcount.5
 60     60(21/35)     wordcount.6
 80     11(4/35)      wordcount.7
 20     66(23/35)     wordcount.8
 10     66(23/35)     wordcount.9
 70     60(21/35)     wordcount.10
 50     60(21/35)     wordcount.11
 50     60(21/35)     wordcount.12
 50     20(7/35)      wordcount.13
 40     14(5/35)      wordcount.14
 60     60(21/35)     wordcount.15
 20     26(9/35)      wordcount.16
 150    54(19/35)     wordcount.17
 900    100(35)       == all ==
```

Test Case Prioritization (3)

- Cumulative decision coverage and cost per test case

```
$ atac -K -q -md main.atac wc.atac wordcount.trace

  cost    % decisions    test
  (cum)   (cumulative)
  ------  -------------  -------------
  120     57(20/35)      wordcount.1
  170     66(23/35)      wordcount.2
  190     71(25/35)      wordcount.3
  200     74(26/35)      wordcount.4
  240     86(30/35)      wordcount.5
  300     89(31/35)      wordcount.6
  380     91(32/35)      wordcount.7
  400     97(34/35)      wordcount.8
  410     100(35)        wordcount.9
  480     100(35)        wordcount.10
  530     100(35)        wordcount.11
  580     100(35)        wordcount.12
  630     100(35)        wordcount.13
  670     100(35)        wordcount.14
  730     100(35)        wordcount.15
  750     100(35)        wordcount.16
  900     100(35)        wordcount.17
```

Test Case Prioritization (4)

- Prioritized cumulative decision coverage and cost per test case

```
$ atac -Q -md main.atac wc.atac wordcount.trace

cost    % decisions    test
(cum)   (cumulative)
------  -------------  -------------
10      66(23/35)      wordcount.9
30      77(27/35)      wordcount.3
40      83(29/35)      wordcount.4
60      89(31/35)      wordcount.8
100     91(32/35)      wordcount.5
140     94(33/35)      wordcount.14
200     97(34/35)      wordcount.15
280     100(35)        wordcount.7
300     100(35)        wordcount.16
350     100(35)        wordcount.2
400     100(35)        wordcount.12
450     100(35)        wordcount.11
500     100(35)        wordcount.13
560     100(35)        wordcount.6
630     100(35)        wordcount.10
750     100(35)        wordcount.1
900     100(35)        wordcount.17
```

cost per additional coverage (increasing order)

10/23 = 0.43

(30-10)/(27-23) = 20/4 = 5.00

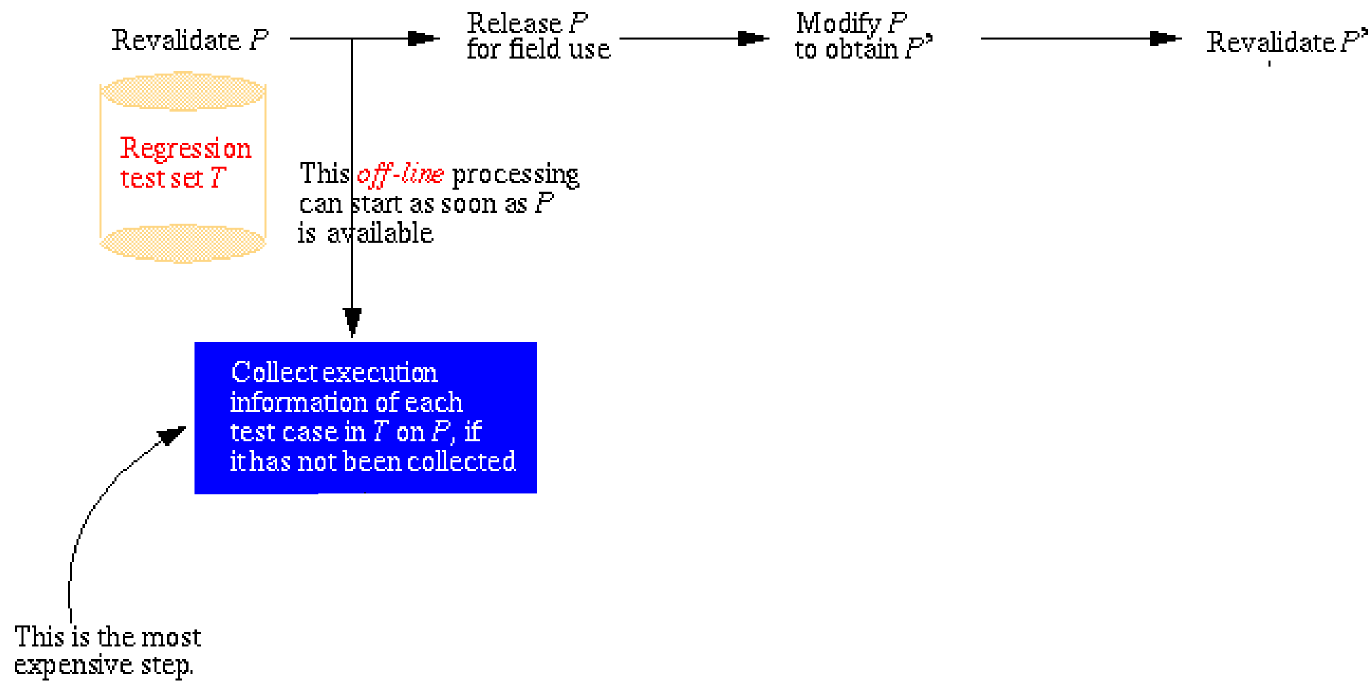
(40-30)/(29-27) = 10/2 = 5.00

(60-40)/(31-29) = 20/2 = 10.00

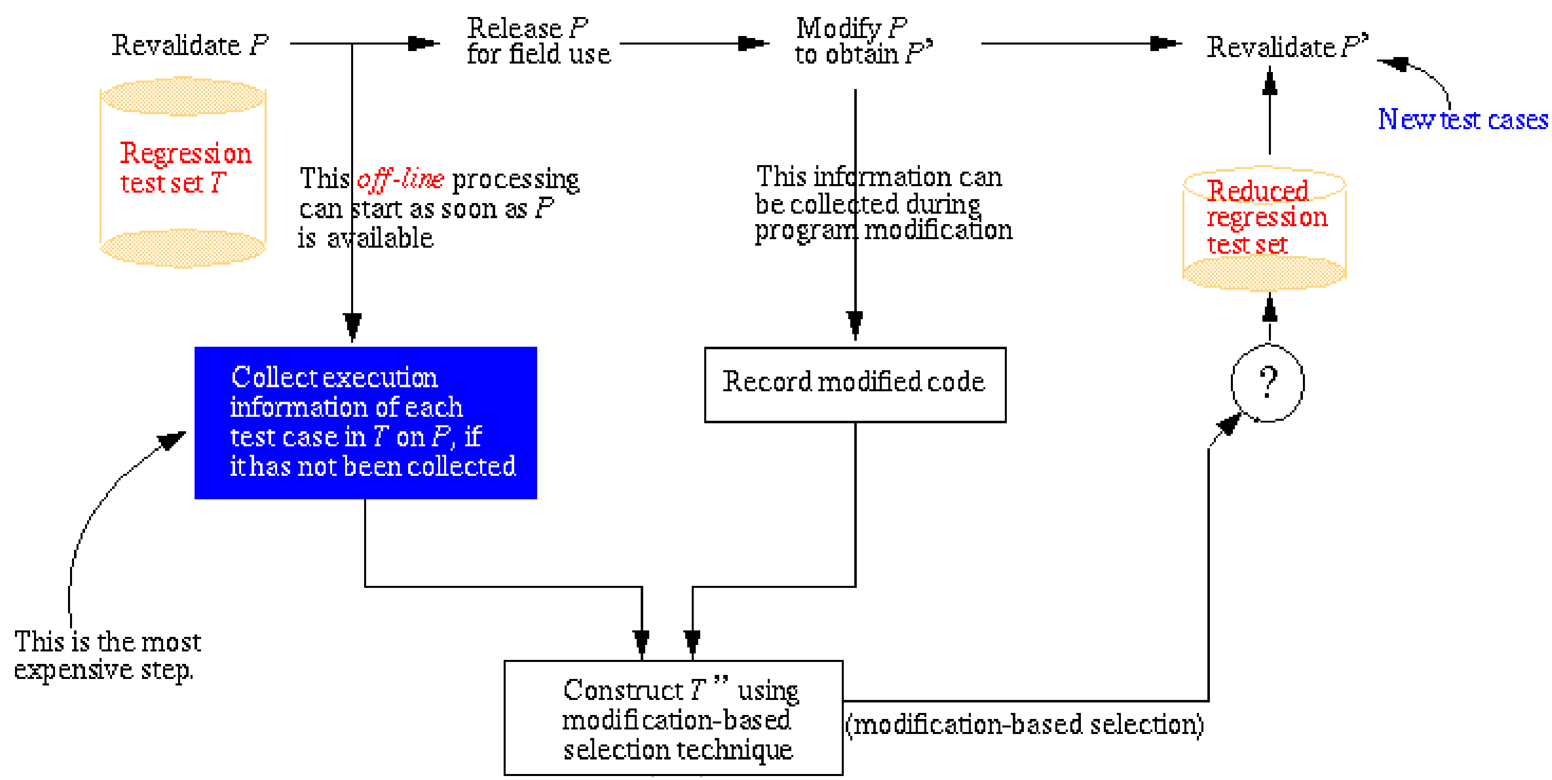
(100-60)/(32-31) = 40/1 = 40.00

Modification-based Selection followed by Test Set Minimization and/or Test Case Prioritization

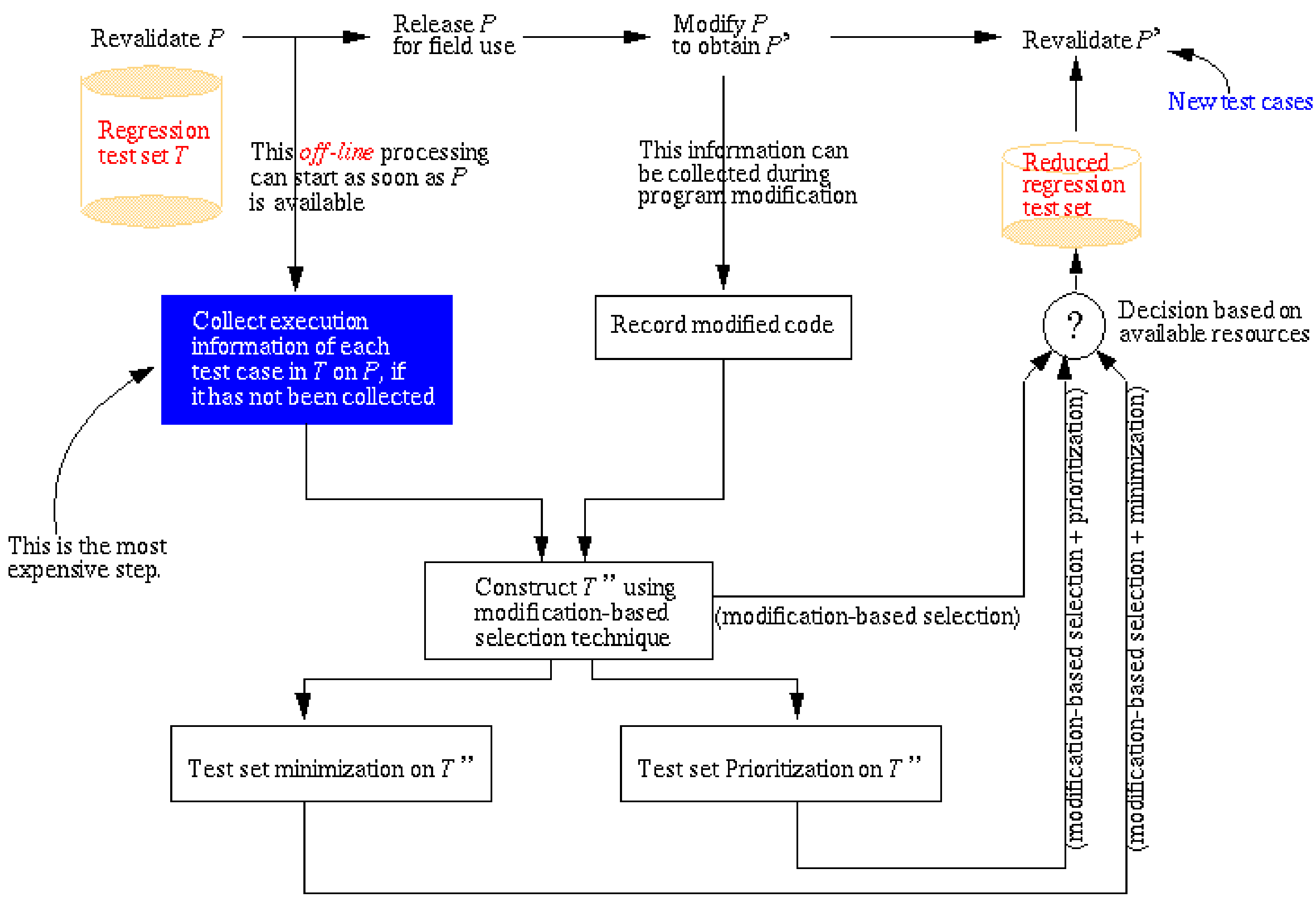
How to Select Regression Tests (3)



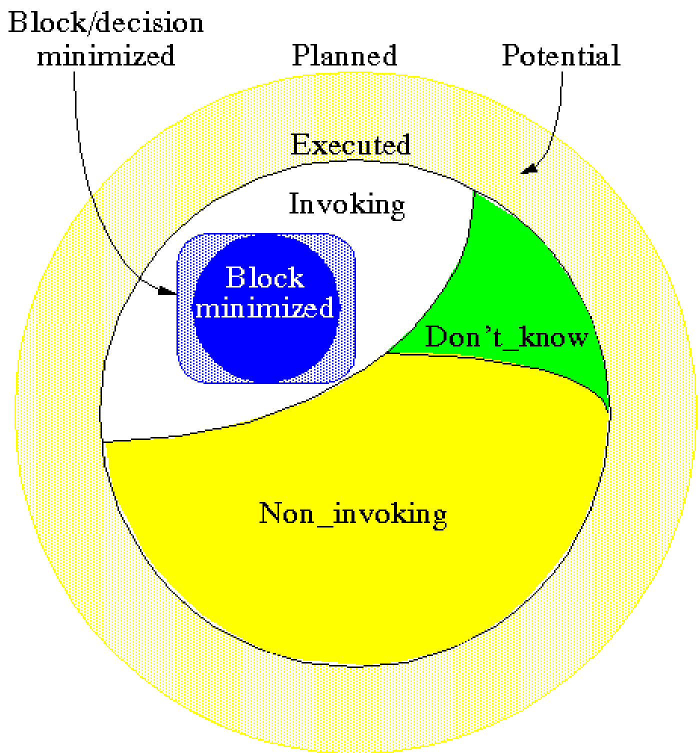
How to Select Regression Tests (4)



How to Select Regression Tests (5)



How to Select Regression Tests (6)



Executed = Invoking ∪ Non_invoking ∪ Don't_know

Potential = Planned - Executed

Possibly_invoking = Potential ∪ Invoking ∪ Don't_know

How to Select Regression Tests (7)

- A *complete* approach selects all tests in the *Planned* category
- A *conservative* approach excludes tests in the *Non-invoking* category
- An *aggressive* approach selects all tests in the *Invoking* category
- A *very aggressive* approach selects the *block/decision minimized subset* of the *Invoking* category
- An *extremely aggressive* approach selects the *block minimized subset* of the *Invoking* category

How to Select Regression Tests (8)

- We can also conduct regression test selection using dynamic slicing (instead of execution slicing).
- What are the advantages?
- What price do we have to pay for such advantages?
- It is a trade-off decision!

Risk Analysis-based Test Selection

Our Method

- Combining *dynamic testing effort* such as code coverage and execution counts with *static complexity* computed by using the internal and external metrics
  - Fault-proneness of a module with high static complexity should be appropriately calibrated based on how much effort has been spent on testing it.

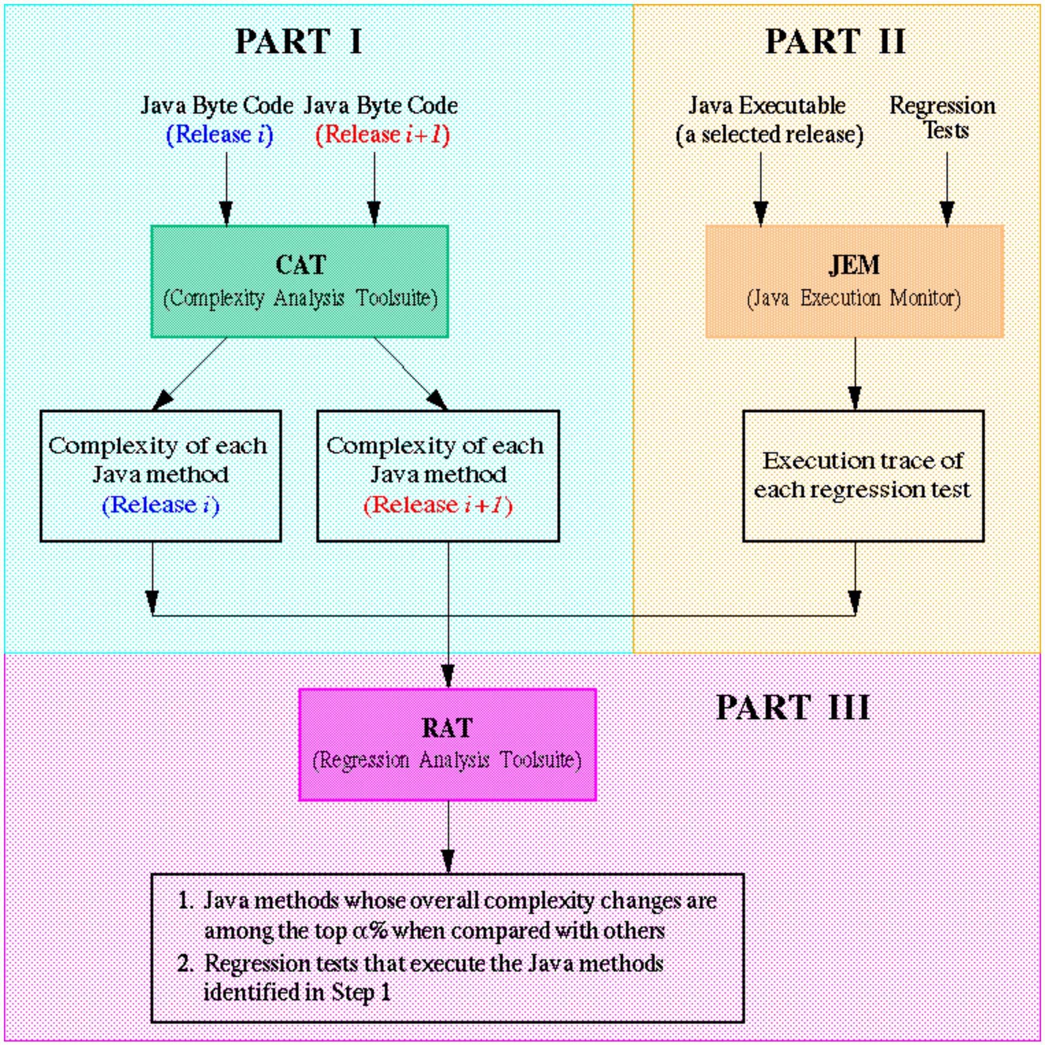
- Fault-proneness of a module =

f (inflows, outflows, fan-in, fan-out, ………
(internal/external complexity metrics)

#of decisions, # of def-uses, # of interface mutants, ……
(controlflow-/dataflow-/mutation-based testing metrics)

block coverage, decision coverage, execution counts, …...)
(dynamic testing effort)

Risk Analysis & Regression Test Selection



Parts I and III need to be performed between every two subsequent releases, but Part II only on some selected release to create an appropriate baseline.

Rules of Thumb for Regression Test Selection

- Effectiveness
- Efficiency
- Tool-Support
- State-of-Art Research versus State-of-Practice Techniques

Remember:

In testing, variation is good.

Summary

- Regression testing is an essential phase of software product development.
- In a situation where test resources are limited and deadlines are to be met, execution of all tests might not be feasible.
- One can make use of different techniques for selecting a subset of all tests to reduce the time and cost for regression testing.