Regression Testing

Professor Mark Gabel
Slides courtesy of W. Eric Wong
Department of Computer Science
University of Texas at Dallas

Outline

- What is regression testing?
  - What is a *regression*?
- How can we select a subset of tests for regression testing?
  - Modification-based test selection
  - Coverage-based test selection
    - Test set minimization
    - Test case prioritization

Regressions

- Ideally, software should *improve* over time.
- But changes can both
  - **improve** software, adding feature and fixing bugs
  - **break** software, introducing new bugs
- We call such “breaking changes” **regressions**

3

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 tests to test the enhancement

4

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)

5

TERMS AND FUNDAMENTALS

6

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.

7

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 **on all inputs**.
  - Advantage: global an universal reasoning – consider *everything*.
  - Key phrase: "could possibly affect"
  - Disadvantages: can be unnecessarily large with too much code. Undecidable to compute. Process does not scale to large code bases.
- A *dynamic slice* can be considered as a refinement of the corresponding static slice by focusing on a **specific input.**
  - Different types of dynamic slices
  - Key phrase: "what *did* affect"
  - Advantage: size is much smaller, more focused
  - 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

8

Execution Slice (2)

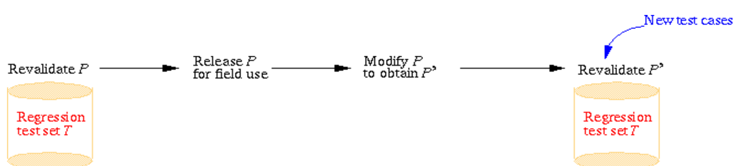
- An execution slice with respect to a given test case is the set of code executed by this test
  - The dynamic slice with respect to the output variables includes only those statements that are not only executed but also have an impact on the **program output** under that test.
  - Since not all the statements executed might have an impact on the output variables, an execution slice can be a super set of the corresponding dynamic slice.
  - There is 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 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.

How to Select Regression Tests (1)

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

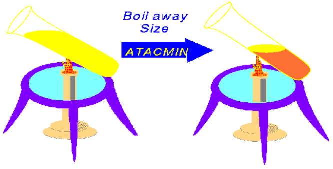


  - The test-all approach is good when you want to be certain that the new version works on all tests developed for the previous version.
  - What if you only have limited resources to run tests and have to meet a deadline?
- The perfect solution: select those on which the new and the old programs **produce different outputs**
  - Undecidable
- What do we do?
  - Heuristics and approximations

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**

An Example (1)

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!

An Example (2)

A patch is installed

Patch Applied

```
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);
```


An Example (3)

Which tests should be reexecuted?

Should T_6 be selected?

An Example (4)

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 (5)

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 (6)

Which tests should be reexecuted? (cont'd)

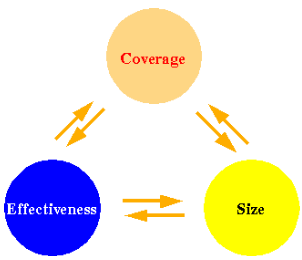
Test case	Input			Output	
	a	b	c	class	area
T_1	2	2	2	equilateral	1.73
T_2	4	4	3	isosceles	5.56
T_3	5	4	3	right	6.00
T_4	6	5	4	scalene	9.92
T_5	3	3	3	equilateral	3.90
T_6	4	3	3	isosceles	4.47

Passed!

Quiz: What if still too many tests?

17

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

18

Coverage, Size, & Effectiveness

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

Coverage and effectiveness are more correlated than size and effectiveness

Greedy Algorithm for Test Set Minimization

- 1: Rank the test cases by a **cost metric**
  - Example: how long each one took to execute on the last test run
- 2: Choose the "cheapest" test case.
- 3. 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

Test Set Minimization (1)

Coverage & Cost per Test Case

```
$ atac  pl(  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)

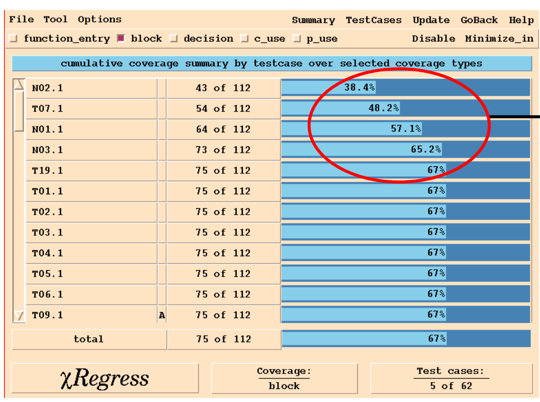
Minimization w.r.t. Block and Decision Coverage

```
$ 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)

- When using that greedy algorithm,
- How can we guarantee the *inclusion* of a certain test?
  - Assign a very *low* cost to that test
- How can we 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 top *n* test cases for re-validation

Test Case Prioritization (2)

- 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)

- 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
```

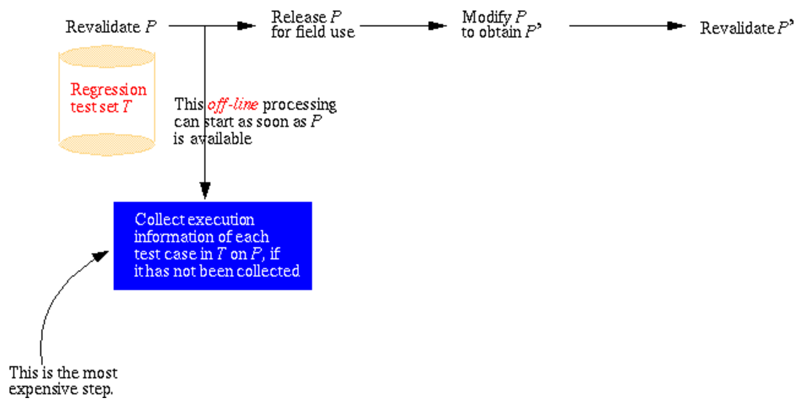
cost per additional coverage

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

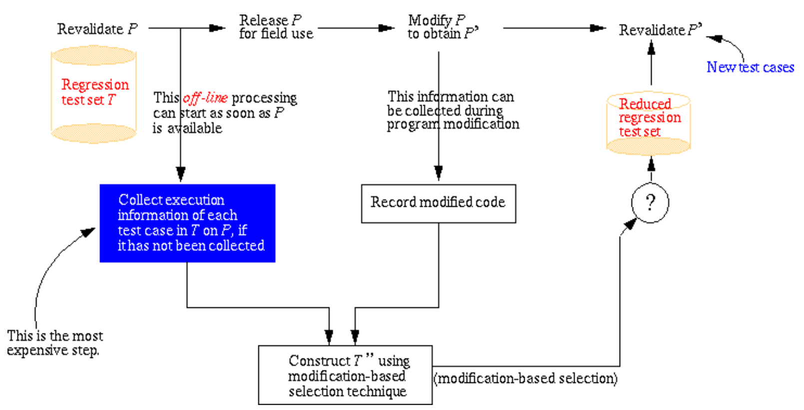
How to Select Regression Tests (3)

Modification-based selection
followed by
test set minimization
and/or
test case prioritization

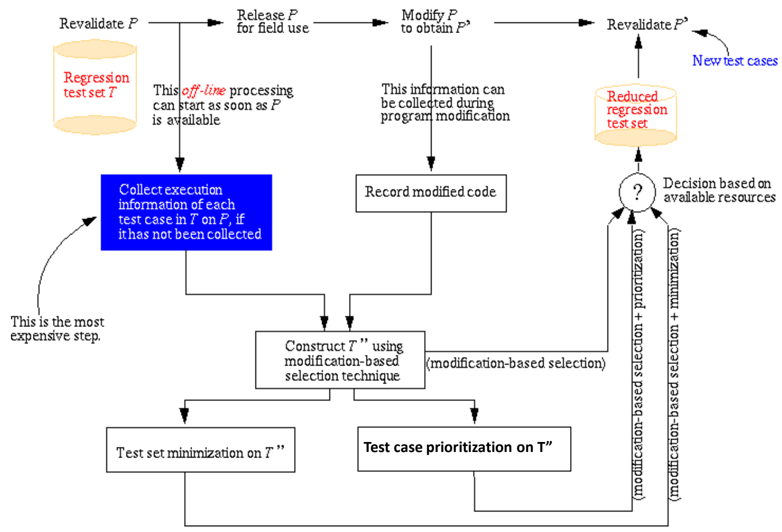
How to Select Regression Tests (4)



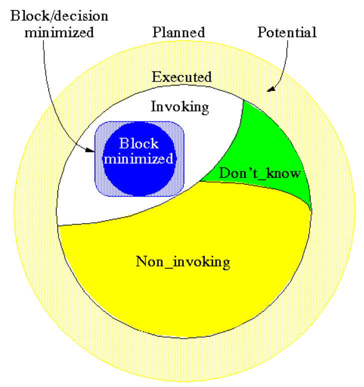
How to Select Regression Tests (5)



How to Select Regression Tests (6)



How to Select Regression Tests (7)



Executed = Invoking ∪ Non_invoking ∪ Dont't_know

Potential = Planned - Extended

Possibly_invoking = Potential ∪ Invoking ∪ Don't_know

How to Select Regression Tests (8)

- 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

37

How to Select Regression Tests (9)

- We can also conduct regression test selection using dynamic slicing (instead of execution slicing)

38

Tools for Regression Testing

- χSuds from Telcordia Technologies (formerly Bellcore) can be used for C/C++ programs to minimize and prioritize tests
- Many commercial tools for regression testing simply run the tests automatically; they do not use any of the algorithms described here for test selection. Instead they rely on the tester for test selection. Such tools can be useful when all tests are to be rerun.

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.
- In such situations one can make use of sophisticated technique for selecting a subset of all tests and hence reduce the time for regression testing.