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

- Senior Research Scientist & Project Manager, Bellcore (Bell Communications Research)
  - Former part of AT&T Bell Labs
  - Now Telcordia Technologies
- Professor & Director of International Outreach
  Department of Computer Science
  University of Texas at Dallas
- 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 (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)

```c
1   While (ring receive buffer | empty and side buffer | empty)
2     {                          
3     Initialize pointer to first message in side buffer or ring received buffer
4     Get a copy of buffer     
5     Switch (message) {       
6     Case incoming message: if (sending switch = out of service)
7     {                       
8     if (ring write buffer = empty)  
9     Send in service to states map manager.  
10    Else             
11    Break;            
12    }                 
13    Process incoming message, set up pointers to optional parameters
14    Break;            
15    }                
16    }                
17    Do optional parameter work
18     }
```

Bug!

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 ≤ 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

```c
int sum, min, count, average;
sum = 0;
min = -1;
read(‘count’);
for (int i = 0; i <= count; i++)
{ read(‘num’);
  sum += sum;
  if (sum < num) {
    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?

<table>
<thead>
<tr>
<th>Test case</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>T&lt;sub&gt;1&lt;/sub&gt;</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T&lt;sub&gt;2&lt;/sub&gt;</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>T&lt;sub&gt;3&lt;/sub&gt;</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>T&lt;sub&gt;4&lt;/sub&gt;</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>T&lt;sub&gt;5&lt;/sub&gt;</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>T&lt;sub&gt;6&lt;/sub&gt;</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**Quiz:** Should T<sub>6</sub> be selected?
An Example (2)

A patch is installed

```plaintext
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;
end;
write(class, area);
```

An Example (3)

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

```plaintext
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;
end;
write(class, area);
```

Quiz: Should $T_2$ be selected?
An Example (4)

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

```java
read (a, b, c);
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)

<table>
<thead>
<tr>
<th>Test case</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>class</td>
</tr>
<tr>
<td>$T_1$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$T_5$</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$T_6$</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$T_2$</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$T_4$</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>$T_3$</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

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 ($T_{\text{sub}}$) of the original test set such that successful execution of the modified code ($P'$) against $T_{\text{sub}}$ 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

<table>
<thead>
<tr>
<th>Coverage</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same</td>
<td>Reduced Significantly</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>$\text{atc}$</th>
<th>$\text{mb}$</th>
<th>$\text{atc}$</th>
<th>$\text{test}$</th>
<th>$\text{trace}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>$%$ blocks</td>
<td>$%$ decisions</td>
<td>$%$ others</td>
<td>$% P$ others</td>
</tr>
<tr>
<td>1/0</td>
<td>63(25/50)</td>
<td>57(20/25)</td>
<td>43(30/30)</td>
<td>60(21/50)</td>
</tr>
<tr>
<td>50</td>
<td>16(9/10)</td>
<td>14(2/25)</td>
<td>11(7/20)</td>
<td>61(3/10)</td>
</tr>
<tr>
<td>29</td>
<td>16(20/50)</td>
<td>20(24/25)</td>
<td>20(24/25)</td>
<td>58(16/25)</td>
</tr>
<tr>
<td>100</td>
<td>18(4/10)</td>
<td>14(4/25)</td>
<td>11(4/20)</td>
<td>51(4/10)</td>
</tr>
<tr>
<td>40</td>
<td>20(15/50)</td>
<td>20(15/25)</td>
<td>13(4/20)</td>
<td>52(4/10)</td>
</tr>
<tr>
<td>1/0</td>
<td>53(20/50)</td>
<td>60(20/25)</td>
<td>52(15/20)</td>
<td>71(22/50)</td>
</tr>
<tr>
<td>20</td>
<td>31(10/50)</td>
<td>31(10/25)</td>
<td>31(10/20)</td>
<td>61(3/31)</td>
</tr>
<tr>
<td>20</td>
<td>75(20/50)</td>
<td>68(20/25)</td>
<td>68(20/20)</td>
<td>68(20/20)</td>
</tr>
<tr>
<td>10</td>
<td>52(20/50)</td>
<td>60(20/25)</td>
<td>60(20/20)</td>
<td>60(20/20)</td>
</tr>
<tr>
<td>50</td>
<td>61(20/50)</td>
<td>60(20/25)</td>
<td>60(20/20)</td>
<td>60(20/20)</td>
</tr>
<tr>
<td>50</td>
<td>61(20/50)</td>
<td>60(20/25)</td>
<td>60(20/20)</td>
<td>60(20/20)</td>
</tr>
<tr>
<td>50</td>
<td>29(15/50)</td>
<td>29(15/25)</td>
<td>29(15/20)</td>
<td>29(15/20)</td>
</tr>
<tr>
<td>40</td>
<td>29(15/50)</td>
<td>29(15/25)</td>
<td>29(15/20)</td>
<td>29(15/20)</td>
</tr>
<tr>
<td>100</td>
<td>60(20/50)</td>
<td>60(20/25)</td>
<td>60(20/20)</td>
<td>60(20/20)</td>
</tr>
</tbody>
</table>

**Minimization w.r.t. Block Coverage**

```
$ atc -M -mb main,atc wc,atc wordcount.trace
$ blocks test
--------------
75(20/50) wordcount.0
50(15/50) wordcount.3
20(10/50) wordcount.14
31(15/50) wordcount.5
100(50)  = all =
```

**Minimization w.r.t. Block Coverage**

```
$ atc -M -mb -q -H main,atc wc,atc wordcount.trace
$ cost $ % blocks $ test
--------------
10 75(20/50) wordcount.0
30 86(44/50) wordcount.3
70 94(48/50) wordcount.14
110 100(50)  wordcount.15
```
Test Set Minimization (3)

Minimization w.r.t. Block and Decision Coverage

<table>
<thead>
<tr>
<th>cost</th>
<th>test cases</th>
<th>% blocks</th>
<th>% decisions</th>
<th>test coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>76 (0/51)</td>
<td>66 (29/90)</td>
<td>66 (29/90)</td>
<td>wordcount.9</td>
</tr>
<tr>
<td>50</td>
<td>98 (44/51)</td>
<td>77 (17/80)</td>
<td>77 (17/80)</td>
<td>wordcount.3</td>
</tr>
<tr>
<td>70</td>
<td>94 (48/51)</td>
<td>83 (29/90)</td>
<td>83 (29/90)</td>
<td>wordcount.14</td>
</tr>
<tr>
<td>100</td>
<td>100 (50/51)</td>
<td>93 (32/95)</td>
<td>93 (32/95)</td>
<td>wordcount.15</td>
</tr>
<tr>
<td>200</td>
<td>100 (50/51)</td>
<td>100 (100)</td>
<td>100 (100)</td>
<td>wordcount.7</td>
</tr>
</tbody>
</table>

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

```bash
$ atac -q wordcount.10 -c `wordcount.trace`
$ atac -M -q mb main,otac wc,otac wordcount.trace
```

<table>
<thead>
<tr>
<th>blocks</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>61(31/51)</td>
<td>wordcount.10</td>
</tr>
<tr>
<td>14(3/13)</td>
<td>wordcount.9</td>
</tr>
<tr>
<td>13(27/51)</td>
<td>wordcount.3</td>
</tr>
<tr>
<td>29(16/51)</td>
<td>wordcount.5</td>
</tr>
<tr>
<td>100(51)</td>
<td>all</td>
</tr>
</tbody>
</table>

```bash
$ atac -M -q mb main,otac wc,otac wordcount.trace
```

<table>
<thead>
<tr>
<th>blocks (cumulative)</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>61(31/51)</td>
<td>wordcount.10</td>
</tr>
<tr>
<td>84(17/51)</td>
<td>wordcount.9</td>
</tr>
<tr>
<td>94(46/51)</td>
<td>wordcount.3</td>
</tr>
<tr>
<td>94(46/51)</td>
<td>wordcount.5</td>
</tr>
<tr>
<td>100(51)</td>
<td>wordcount.14</td>
</tr>
</tbody>
</table>
Test Set Minimization (7)

Exclude wordcount.9 in the Minimized Set

```
$ stac -n wordcount.9 > wc1000.wordcount.trace
$ stac -M -n main.stac wc1000.wordcount.trace
% blocks     test
-------------------------
   75(28/51)   wordcount.8
   53(22/51)   wordcount.9
   31(18/51)   wordcount.5
   20(10/51)   wordcount.14
   10(5/51)    -- all --
$ stac -N -m main.stac wc1000.wordcount.trace
% blocks     test
(Cumulative)
-------------------------
   75(28/51)   wordcount.8
   85(44/51)   wordcount.9
   94(49/51)   wordcount.5
   100(51/51)  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 \textit{increasing cost per additional coverage}
- Select the first test case
- Repeat the above two steps until \( n \) test cases are selected

\[ \begin{array}{lcr}
\text{cost} & \% \text{ decisions} & \text{test} \\
120 & 57(20/25) & \text{wordcount.1} \\
59 & 11(4/35) & \text{wordcount.2} \\
29 & 48(17/25) & \text{wordcount.3} \\
10 & 11(4/35) & \text{wordcount.4} \\
40 & 74(25/35) & \text{wordcount.5} \\
60 & 60(21/28) & \text{wordcount.6} \\
89 & 11(4/35) & \text{wordcount.7} \\
20 & 66(23/25) & \text{wordcount.8} \\
10 & 68(25/25) & \text{wordcount.9} \\
70 & 60(21/28) & \text{wordcount.10} \\
59 & 60(21/28) & \text{wordcount.11} \\
59 & 60(21/28) & \text{wordcount.12} \\
50 & 20(7/25) & \text{wordcount.13} \\
40 & 14(5/25) & \text{wordcount.14} \\
60 & 60(21/28) & \text{wordcount.15} \\
20 & 26(9/25) & \text{wordcount.16} \\
190 & 54(19/25) & \text{wordcount.17} \\
900 & 100(35) & \text{all} \\
\end{array} \]
### Test Case Prioritization (3)

Cumulative decision coverage and cost per test case

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Cost (Cumulative)</th>
<th>Cost (Cumulative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>60(30/30)</td>
<td>60(30/30)</td>
</tr>
<tr>
<td>30</td>
<td>80(40/40)</td>
<td>80(40/40)</td>
</tr>
<tr>
<td>40</td>
<td>90(50/50)</td>
<td>90(50/50)</td>
</tr>
<tr>
<td>60</td>
<td>90(50/50)</td>
<td>90(50/50)</td>
</tr>
<tr>
<td>100</td>
<td>120(60/60)</td>
<td>120(60/60)</td>
</tr>
<tr>
<td>140</td>
<td>160(80/80)</td>
<td>160(80/80)</td>
</tr>
<tr>
<td>200</td>
<td>180(90/90)</td>
<td>180(90/90)</td>
</tr>
<tr>
<td>280</td>
<td>200(100/100)</td>
<td>200(100/100)</td>
</tr>
<tr>
<td>350</td>
<td>290(110/110)</td>
<td>290(110/110)</td>
</tr>
<tr>
<td>400</td>
<td>320(120/120)</td>
<td>320(120/120)</td>
</tr>
<tr>
<td>500</td>
<td>360(130/130)</td>
<td>360(130/130)</td>
</tr>
<tr>
<td>600</td>
<td>400(140/140)</td>
<td>400(140/140)</td>
</tr>
<tr>
<td>700</td>
<td>450(150/150)</td>
<td>450(150/150)</td>
</tr>
<tr>
<td>800</td>
<td>500(160/160)</td>
<td>500(160/160)</td>
</tr>
<tr>
<td>900</td>
<td>550(170/170)</td>
<td>550(170/170)</td>
</tr>
</tbody>
</table>

### Test Case Prioritization (4)

Prioritized cumulative decision coverage and cost per test case

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Cost (Cumulative)</th>
<th>Cost (Cumulative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>60(30/30)</td>
<td>60(30/30)</td>
</tr>
<tr>
<td>30</td>
<td>80(40/40)</td>
<td>80(40/40)</td>
</tr>
<tr>
<td>40</td>
<td>90(50/50)</td>
<td>90(50/50)</td>
</tr>
<tr>
<td>60</td>
<td>90(50/50)</td>
<td>90(50/50)</td>
</tr>
<tr>
<td>100</td>
<td>120(60/60)</td>
<td>120(60/60)</td>
</tr>
<tr>
<td>140</td>
<td>160(80/80)</td>
<td>160(80/80)</td>
</tr>
<tr>
<td>200</td>
<td>180(90/90)</td>
<td>180(90/90)</td>
</tr>
<tr>
<td>280</td>
<td>200(100/100)</td>
<td>200(100/100)</td>
</tr>
<tr>
<td>350</td>
<td>290(110/110)</td>
<td>290(110/110)</td>
</tr>
<tr>
<td>400</td>
<td>320(120/120)</td>
<td>320(120/120)</td>
</tr>
<tr>
<td>500</td>
<td>360(130/130)</td>
<td>360(130/130)</td>
</tr>
<tr>
<td>600</td>
<td>400(140/140)</td>
<td>400(140/140)</td>
</tr>
<tr>
<td>700</td>
<td>450(150/150)</td>
<td>450(150/150)</td>
</tr>
<tr>
<td>800</td>
<td>500(160/160)</td>
<td>500(160/160)</td>
</tr>
<tr>
<td>900</td>
<td>550(170/170)</td>
<td>550(170/170)</td>
</tr>
</tbody>
</table>
Modification-based Selection followed by Test Set Minimization and/or Test Case Prioritization

How to Select Regression Tests (3)

1. Collect instrumentation information of each test case in $T$ and $T^*$ if not been collected.
2. Revalidate $F$.
3. Revalidate $F^*$.
4. Release $F$ for field use.
5. Modify $F$ to obtain $F^*$.
6. The off-line processing can start as soon as $F^*$ is available.

This is the most imperative reg.
How to Select Regression Tests (4)

1. Validate $P$
2. Release $P$ for field use
3. Modify $P$ to obtain $P^*$
4. Redundate $P^*$

For new use cases:

- This off-line processing can start as soon as $P^*$ is available.
- This information can be calculated during program modification.
- Construct $T^*$ using modification-based selection technique.
- Test set prioritization on $T^*$.
How to Select Regression Tests (6)

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 =
  \[ f(\text{inflows, outflows, fan-in, fan-out, \ldots}) \]

  \[
  \text{internal/external complexity metrics} \quad \text{#of decisions, # of def-uses, # of interface mutants, \ldots} \\
  \text{controlflow-/dataflow-/mutation-based testing metrics} \quad \text{block coverage, decision coverage, execution counts, \ldots} \\
  \text{dynamic testing effort}
  \]

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.