Mutation Coverage

Yegor Bugayenko

Lecture #16 out of 24
80 minutes

Example, Part I

Live Code:

```
int fibonacci(int n) {
  if (n <= 2) {
    return 1;
  }
  return fibonacci(n - 1)
    + fibonacci(n - 2);
}
```

Test Code:

```
assert fibonacci(2) == 1;
assert fibonacci(5) > 5;
```

$\texttt{Cov} = 7/7 = 100\%$

Example, Part II

Live Code:

```
int fibonacci(int n) {
  if (n <= 2) {
    return 1;
  }
  return fibonacci(n - 1)
    + fibonacci(n - 2);
}
```

Mutant #1:

```
int fibonacci(int n) {
  if (n <= 2) {
    return 1;
  }
  return fibonacci(n + 1)
    + fibonacci(n - 2);
}
```

Mutant #2:

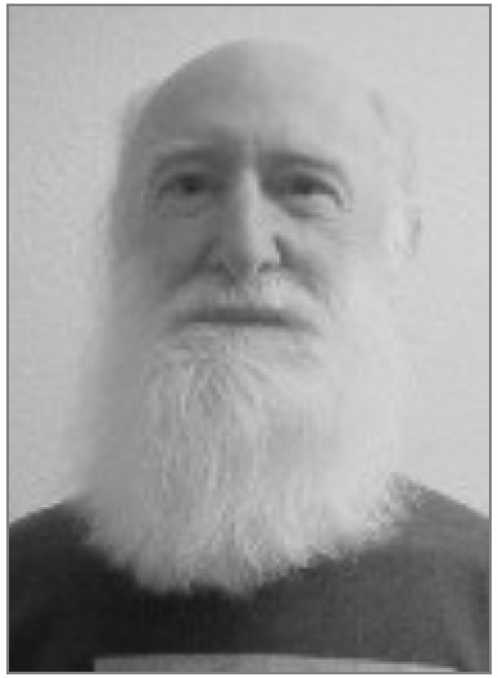
```
int fibonacci(int n) {
  if (n == 2) {
    return 1;
  }
  return fibonacci(n - 1)
    + fibonacci(n - 2);
}
```

Test Code:

```
assert fibonacci(2) == 1;
assert fibonacci(5) > 5;
```

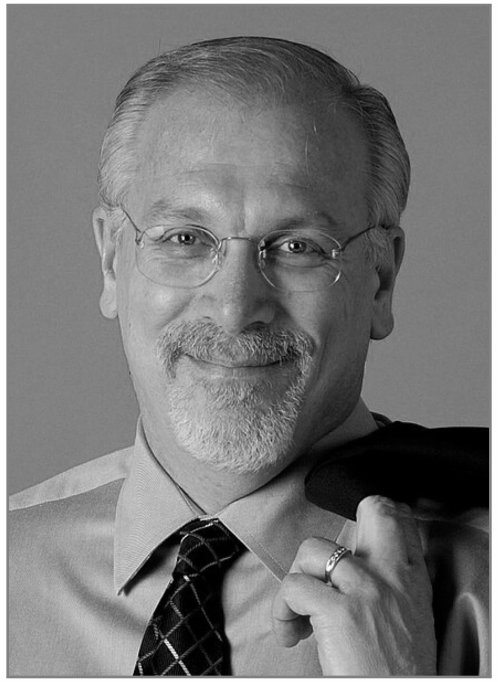
Mutation Operators

- Statement deletion
- Statement duplication or insertion
- Replacement of boolean subexpressions with `TRUE` and `FALSE`
- Replacement of some arithmetic operations, e.g. + to *, - to /
- Replacement of some boolean relations, e.g. > to >=, == to <=
- Replacement of variables with others from the same scope
- Remove method body



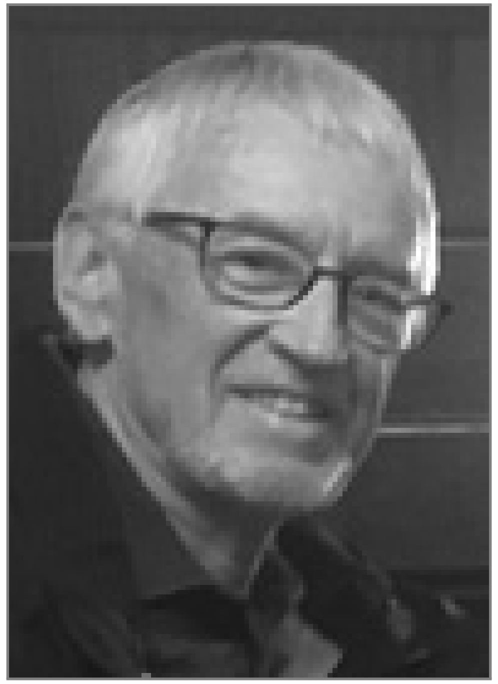
“???”

— Richard G. Hamlet, *Testing Programs with the Aid of a Compiler*, IEEE Transactions on Software Engineering, 4, 1977



"Our groups at Yale University and the Georgia Institute of Technology have constructed a system whereby we can determine the extent to which a given set of test data has adequately tested a Fortran program by direct measurement of the number and kinds of errors it is capable of uncovering."

— Richard A. DeMillo, Richard J. Lipton, Frederick G. Sayward, *Hints on test Data Selection: Help for the Practicing Programmer*, IEEE Computer 11(4), 1978



"In weak mutation testing method, tests are constructed which are guaranteed to force program statements which contain certain classes of errors to act incorrectly during the execution of the program over those tests."

— William E. Howden, *Weak Mutation Testing and Completeness of Test Sets*, IEEE Transactions on Software Engineering 4, 1982

"Our results indicate that weak mutation can be applied in a manner that is almost as effective as mutation testing, and with significant computational savings."

— Jeff Offutt and Stephen D. Lee, *An Empirical Evaluation of Weak Mutation*, IEEE Transactions on Software Engineering 20(5), 1994

"Our analysis suggests that mutants, when using carefully selected mutation operators and after removing equivalent mutants, can provide a good indication of the fault detection ability of a test suite."

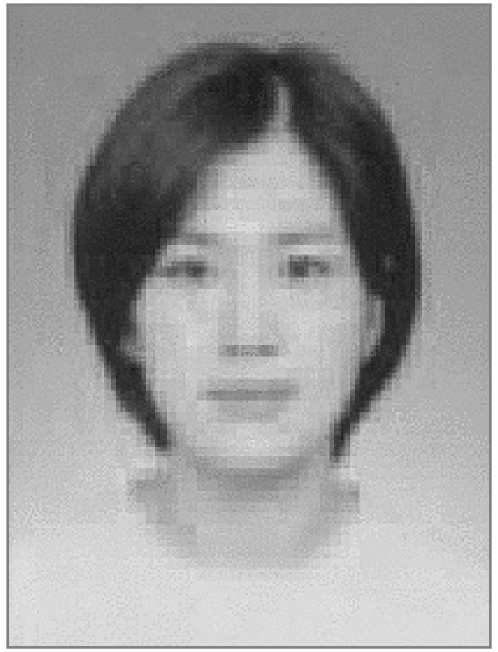
— James H. Andrews, Lionel C. Briand and Yvan Labiche, *Is Mutation an Appropriate Tool for Testing Experiments?*, Proceedings of the 27th International Conference on Software Engineering (ICSE), 2005

Table 3. Matched Pairs *t*-test Results – test suite size = 100

Subject Programs	Matched Pairs Results: Mean Af(S) – Am(S)	*t*-ratio	*p*-value
Space	0.014	16.87	< 0.0001
Replace	-0.266	-233.96	0.0000
Printtokens	-0.344	-158.2	0.0000
Printtokens2	-0.061	-59.39	0.0000
Schedule	-0.298	-161.33	0.0000
Schedule2	-0.327	-152.19	0.0000
Tcas	-0.1128	-57.56	0.0000
Totinfo	-0.1037	-145.78	0.0000

"Average differences range from 6% to 34%, with an average of 22%. "

Source: James H. Andrews, Lionel C. Briand and Yvan Labiche, *Is Mutation an Appropriate Tool for Testing Experiments?*, Proceedings of the 27th International Conference on Software Engineering (ICSE), 2005



"Comparing with previous mutation systems for procedural programs, MuJava is very fast. However, it is relatively slow when it generates and runs lots of mutants."

— Yu-Seung Ma, Jeff Offutt, and Yong-Rae Kwon, *MuJava: A Mutation System for Java*, Proceedings of the 28th International Conference on Software Engineering (ICSE), 2006

Operator	Description
IHD	Hiding variable deletion
IHI	Hiding variable insertion
IOD	Overriding method deletion
IOP	Overridden method calling position change
IOR	Overridden method rename
ISI	*super* keyword insertion
ISD	*super* keyword deletion
IPC	Explicit call of a parent's constructor deletion
PNC	*new* method call with child class type
PMD	Instance variable declaration with parent class type
PPD	Parameter variable declaration with child class type
PCI	Type cast operator insertion
PCC	Cast type change
PCD	Type cast operator insertion
PRV	Reference assignment with other compatible type
OMR	Overloading method contents change
OMD	Overloading method deletion
OAC	Argument order change
JTI	*this* keyword insertion
JTD	*this* keyword deletion
JSI	*static* modifier insertion
JSD	*static* modifier deletion
JID	Member variable initialization deletion
JDC	Java-supported default constructor create
EOA	Reference and content assignment replacement
EOC	Reference and content assignment replacement
EAM	Accessor method change
EMM	Modifier method change

Table 2: Class-level Mutation Operators for Java

"Method-level mutation operators handle primitive features of programming languages. They modify expressions by replacing, deleting, and inserting primitive operators. Class-level mutation operators handle object-oriented specific features such as inheritance, polymorphism and dynamic binding."

Source: Yu-Seung Ma, Jeff Offutt, and Yong-Rae Kwon, *MuJava: A Mutation System for Java*, Proceedings of the 28th International Conference on Software Engineering (ICSE), 2006

"... RIP Model ..."

— Paul Ammann and Jeff Offutt, *Introduction to Software Testing*, 2016

Mutation Coverage can be calculated by a few tools:

- PIT for Java
- StrykerJS for JavaScript
- Mutate++ for C++
- mutatest for Python
- mutant for Ruby

Read this: