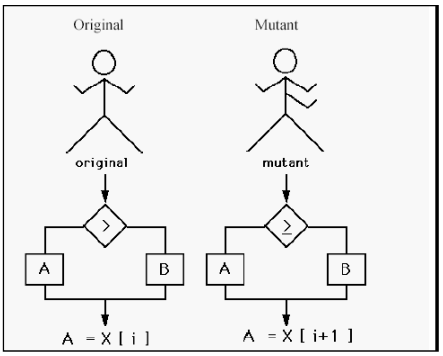
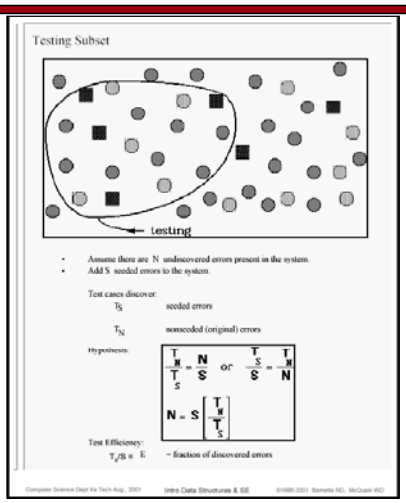
Mutation Testing



This lecture material is copyrighted by Laurie Williams (2007). However, you are encouraged to download, forward, copy, print, or distribute it, provided you do so in its entirety (including this notice) and do not sell or otherwise exploit it for commercial purposes.

For PowerPoint version of the slides, contact Laurie Williams at williams@csc.ncsu.edu.

Fault Injection/Fault Seeding



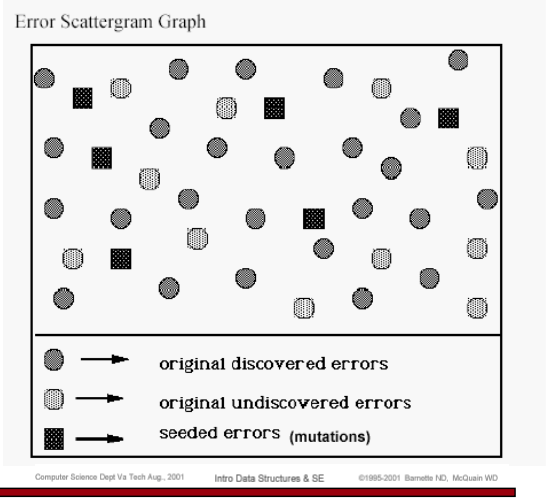
Mutation Testing

Mutation testing introduces faults into software by creating many different versions of the program.

Each version has one very small change (which introduces a fault) compared to the actual implementation.

The idea is to see if the test cases that were written can detect the new fault.

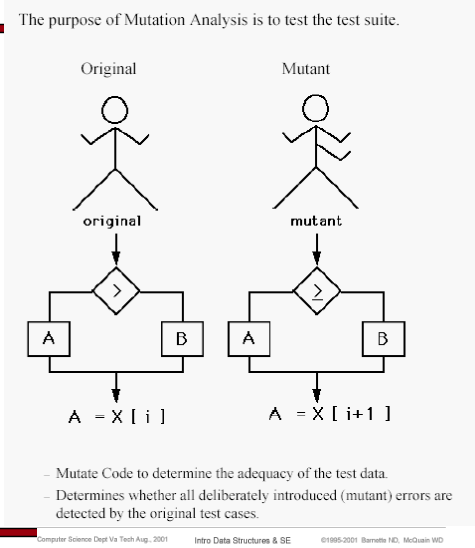
Mutation testing is a way to measure the quality of a test suite.



Research Group Software Engineering @ NCSU
© 2007 Laurie Williams
NC STATE UNIVERSITY

Mutation Operators

- It is a rule that is applied to a program to create mutants, for example:
  - Replace each operand by every other syntactically legal operand
  - Modify expressions by replacing operators and inserting new operators
  - Delete entire statements



Research Group Software Engineering @ NCSU
© 2007 Laurie Williams
NC STATE UNIVERSITY

Offutt & Untch, *Mutation 2000: Uniting the Orthogonal*

Method-level mutation operators

Operator	Description
AOR	Arithmetic Operator Replacement
AOI	Arithmetic Operator Insertion
AOD	Arithmetic Operator Deletion
ROR	Relational Operator Replacement
COR	Conditional Operator Replacement
COI	Conditional Operator Insertion
COD	Conditional Operator Deletion
SOR	Shift Operator Replacement
LOR	Logical Operator Replacement
LOI	Logical Operator Insertion
LOD	Logical Operator Deletion
ASR	Assignment Operator Replacement

Research Group Software Engineering @ NCSU
© 2007 Laurie Williams
NC STATE UNIVERSITY

From: http://ise.gmu.edu/~ofut/mujava/mutopsMethod.pdf

Method-level mutation operators

1 Arithmetic Operators

The Java programming language supports five arithmetic operators for all floating-point and integer numbers; (1) +, (2) -, (3) *, (4) /, and (5) %. These operators are all binary. However, both + and - have unary versions. Four short-cut arithmetic operators are defined, (1) op++, (2) ++op, (3) op−−, and (4) −−op.

- **AOR**$_B$: Arithmetic Operator Replacement
  Replace basic binary arithmetic operators with other binary arithmetic operators.
- **AOR**$_U$: Arithmetic Operator Replacement
  Replace basic unary arithmetic operators with other unary arithmetic operators.
- **AOR**$_S$: Arithmetic Operator Replacement
  Replace short-cut arithmetic operators with other unary arithmetic operators.
- **AOI**$_U$: Arithmetic Operator Insertion
  Insert basic unary arithmetic operators.
- **AOI**$_S$: Arithmetic Operator Insertion
  Insert short-cut arithmetic operators.
- **AOD**$_U$: Arithmetic Operator Deletion
  Delete basic unary arithmetic operators.
- **AOD**$_S$: Arithmetic Operator Deletion
  Delete short-cut arithmetic operators.

Research Group Software Engineering @ NCSU
© 2007 Laurie Williams
NC STATE UNIVERSITY

From: http://ise.gmu.edu/~ofut/mujava/mutopsMethod.pdf

Class-level Mutation Operators

Language Feature	Operator	Description
Encapsulation	AMC	Access modifier change
Inheritance	IHD	Hiding variable deletion
	IHI	Hiding variable insertion
	IOD	Overriding method deletion
	IOP	Overriding method calling position change
	IOR	Overriding method rename
	ISI	`super` keyword insertion
	ISD	`super` keyword deletion
	IPC	Explicit call to a parent's constructor deletion
Polymorphism	PNC	new method call with child class type
	PMD	Member 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 deletion
	PRV	Reference assignment with other comparable variable
	OMR	Overloading method contents replace
	OMD	Overloading method deletion
	OAC	Arguments of overloading method call change
Java-Specific Features	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 creation
	EOA	Reference assignment and content assignment replacement
	EOC	Reference comparison and content comparison replacement
	EAM	Acessor method change
	EMM	Modifier method change

From: http://ise.gmu.edu/~ofut/mujava/mutopsClass.pdf

Hiding Variable Deletion

Original Code

```
class List {
    int size;
    ... ...
}
class Stack extends List {
    int size;
    ... ...
}
```

IHD Mutant

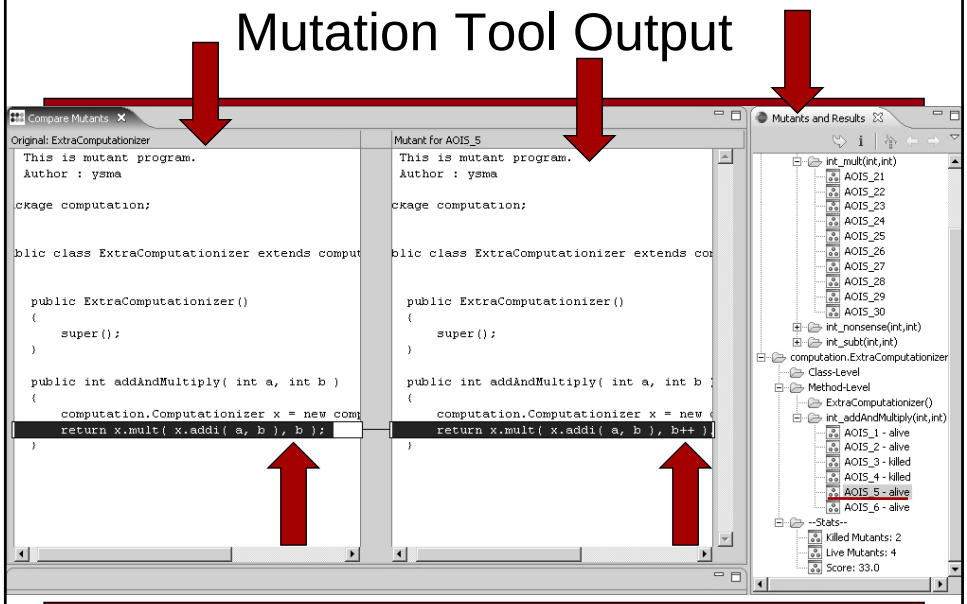
```
class List {
    int size;
    ... ...
}
class Stack extends List {
    // int size;
    ... ...
}
```

Terminology

- Killed Mutant: test case(s) detect the one fault injected via a mutation operator
- Alive/Live Mutant: test cases do not detects the one fault fault injected via a mutation operator
- Stubborn Mutant: mutant cannot be killed by adding any test cases

Research Group Software Engineering @ NCSU
© 2007 Laurie Williams
NC STATE UNIVERSITY

Mutation Tool Output



Research Group Software Engineering @ NCSU

NC STATE UNIVERSITY

Mutation Score (MS)

- Mutation Score (MS):

$$MS = \frac{\text{\# of Killed Mutants}}{\text{\# of Mutants}}$$

Testers add new tests, verify correctness, and kill mutants until satisfied with the mutation score

Research Group
Software Engineering @ NCSU
© 2007 Laurie Williams
NC STATE UNIVERSITY