Integration and Object-Oriented Testing

Integration Testing
Testing connections among separate program units

- In Java, testing the way classes, packages and components are connected
  - “Component” is used as a generic term

- This tests features that are unique to object-oriented programming languages
  - inheritance, polymorphism and dynamic binding

- Integration testing is often based on couplings – the explicit and implicit relationships among software components
Grammar Integration Testing (5.3.1)

There is no known use of grammar testing at the integration level
Integration Mutation (5.3.2)

- Faults related to component integration often depend on a mismatch of assumptions
  - Callee thought a list was sorted, caller did not
  - Callee thought all fields were initialized, caller only initialized some of the fields
  - Caller sent values in kilometers, callee thought they were miles

- Integration mutation focuses on mutating the connections between components
  - Sometimes called “interface mutation”
  - Both caller and callee methods are considered

Four Types of Mutation Operators

- Change a calling method by modifying values that are sent to a called method
- Change a calling method by modifying the call
- Change a called method by modifying values that enter and leave a method
  - Includes parameters as well as variables from higher scopes (class level, package, public, etc.)
- Change a called method by modifying return statements from the method
## Five Integration Mutation Operators

1. **IPVR — Integration Parameter Variable Replacement**
   
   Each parameter in a method call is replaced by each other variable in the scope of the method call that is of compatible type.
   
   - This operator replaces primitive type variables as well as objects.

2. **IUOI — Integration Unary Operator Insertion**
   
   Each expression in a method call is modified by inserting all possible unary operators in front and behind it.
   
   - The unary operators vary by language and type

3. **IPEX — Integration Parameter Exchange**
   
   Each parameter in a method call is exchanged with each parameter of compatible types in that method call.
   
   - max (a, b) is mutated to max (b, a)

4. **IMCD — Integration Method Call Deletion**
   
   Each method call is deleted. If the method returns a value and it is used in an expression, the method call is replaced with an appropriate constant value.
   
   - Method calls that return objects are replaced with calls to “new ()”

5. **IREM — Integration Return Expression Modification**
   
   Each expression in each return statement in a method is modified by applying the UOI and AOR operators.
Object-Oriented Mutation

- **Testing Levels**
  - intra-method
  - inter-method
  - intra-class
  - inter-class

- These five operators can be applied to non-OO languages
  - C, Pascal, Ada, Fortran, ...
- They do not support object oriented features
  - Inheritance, polymorphism, dynamic binding
- Two other language features that are often lumped with OO features are information hiding (encapsulation) and overloading
- Even experienced programmers often get encapsulation and access control wrong

Encapsulation, Information Hiding and Access Control

- **Encapsulation**: An abstraction mechanism to implement information hiding, which is a design technique that attempts to protect parts of the design from parts of the implementation
  - Objects can restrict access to their member variables and methods

- Java provides four access levels (C++ & C# are similar)
  - private
  - protected
  - public
  - default (also called package)

- Often not used correctly or understood, especially for programmers who are not well educated in design
### Access Control in Java

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Same class</th>
<th>Same package</th>
<th>Different package subclass</th>
<th>Different package non-subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>Y</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>package</td>
<td>Y</td>
<td>Y</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>protected</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>n</td>
</tr>
<tr>
<td>public</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

- Most class variables should be **private**
- Public variables should seldom be used
- Protected variables are particularly dangerous -- future programmers can accidentally override (by using the same name) or accidentally use (by mis-typing a similar name)
  - They should be called “unprotected”

### Access Control in Java (2)

#### Class 1
- public members
- protected members
- default
- private members

#### Class 2
- inheritance

#### Class 3

#### Class 4

#### Class 5

#### Package
Object-Oriented Language Features (Java)

- **Method overriding**
  Allows a method in a subclass to have the same name, arguments and result type as a method in its parent

- **Variable hiding**
  Achieved by defining a variable in a child class that has the same name and type of an inherited variable

- **Class constructors**
  Not inherited in the same way other methods are – must be explicitly called

- **Each object has …**
  - a declared type: `Parent P;`
  - an actual type: `P = new Child ();` or assignment: `P = Pold;`
  - Declared and actual types allow uses of the same name to reference different variables with different types

OO Language Feature Terms

- **Polymorphic attribute**
  - An object reference that can take on various types
  - Type the object reference takes on during execution can change

- **Polymorphic method**
  - Can accept parameters of different types because it has a parameter that is declared of type Object

- **Overloading**
  - Using the same name for different constructors or methods in the same class

- **Overriding**
  - A child class declares an object or method with a name that is already declared in an ancestor class
  - Easily confused with overloading because the two mechanisms have similar names and semantics
  - Overloading is in the same class, overriding is between a class and a descendant
More OO Language Feature Terms

• Members associated with a class are called class or instance variables and methods
  – Static methods can operate only on static variables; not instance variables
  – Instance variables are declared at the class level and are available to objects

• 29 object-oriented mutation operators defined for Java – muJava

• Broken into 4 general categories

Class Mutation Operators for Java

1. Encapsulation
   - AMC
2. Inheritance
   - IHI, IHD, IOD, IOP, IOR, ISI, ISD, IPC
3. Polymorphism
   - PNC, PMD, PPD, PCI, PCD, PCC, PRV, OMR, OMD, OAC
4. Java-Specific
   - JTI, JTD, JSI, JSD, JID, JDC, EOA, EOC, EAM, EMM
OO Mutation Operators—Encapsulation

1. AMC — Access Modifier Change

The access level for each instance variable and method is changed to other access levels.

OO Mutation Operators—Example

1. AMC – Access Modifier Change

<table>
<thead>
<tr>
<th>point</th>
</tr>
</thead>
<tbody>
<tr>
<td>private int x;</td>
</tr>
<tr>
<td>(\Delta_1) public int x;</td>
</tr>
<tr>
<td>(\Delta_2) protected int x;</td>
</tr>
<tr>
<td>(\Delta_3) int x;</td>
</tr>
</tbody>
</table>
**OO Mutation Operators—*Inheritance***

2. **IHI — Hiding Variable Insertion**
   A declaration is added to hide the declaration of each variable declared in an ancestor.

3. **IHD — Hiding Variable Deletion**
   Each declaration of an overriding or hiding variable is deleted.

4. **IOD — Overriding Method Deletion**
   Each entire declaration of an overriding method is deleted.

5. **IOP — Overriding Method Calling Position Change**
   Each call to an overridden method is moved to the first and last statements of the method and up and down one statement.

---

**OO Mutation Operators—*Example***

2. **IHI — Hiding Variable Insertion**
   ```java
   int x;
   int y;
   ```

3. **IHD — Hiding Variable Deletion**
   ```java
   int x;
   int y;
   ```

   ```java
   // int x;
   int y;
   ```

   ```java
   // int y;
   ```

   ```java
   ∆1 int x;
   ∆2 int y;
   ```

   ```java
   int x;
   ```

   ```java
   // int x;
   int y;
   ```

   ```java
   // int y;
   ```
**OO Mutation Operators—Example**

4. **IOD — Overriding Method Deletion**

```
point
void set (int x, int y)
```

```
colorpoint
void set (int x, int y)
```

```
// void set (int x, int y)
```

```
point
void set (int x, int y)
{ width = 5;...}
```

```
colorpoint
void set (int x, int y)
{ super.set (x, y); width = 10;}
```

```
Δ { width=10; super.set (x, y); }
```

**5. IOP — Overriding Method Calling Position Change**

```
point
void set (int x, int y)
```

```
colorpoint
void set (int x, int y)
```

```
void set (int x, int y)
{ super.set (x, y); width = 10;}
```

```
Δ { width=10; super.set (x, y); }
```

**OO Mutation Operators—Inheritance**

6. **IOR — Overriding Method Rename**

Renames the parent's versions of methods that are overridden in a subclass so that the overriding does not affect the parent's method.

7. **ISI — Super Keyword Insertion**

Insert the keyword `super` in front of each overriding method or variable.

8. **ISD — Super Keyword Deletion**

Delete each occurrence of the `super` keyword.

9. **IPC — Explicit Call of a Parent’s Constructor Deletion**

Each call to a `super` constructor is deleted.
**OO Mutation Operators—**Example

6. IOR – Overriding Method Rename

```java
void set (int x, int y)
void setP (int x, int y)
```

```java
void setDimension (int d)
{
  set(x, y);
  setP(x, y);
}
```

7. ISI – Super keyword Insertion

```java
void set (int x, int y)
```

```java
point p;
p = new colorpoint ();
p.set (1, 2);
p.setDimension (3);
```

8. ISD – Super keyword Deletion

```java
void set (int x, int y)
```

```java
set(x, y);
```

9. IPC – Explicit Call of a Parent’s Constructor Deletion

```java
void set (int x, int y)
```

```java
colorpoint (int x, int y, int color)
{
  super (x, y);
  return x;
}
```

10. IPC – Explicit Call of a Parent’s Constructor Deletion
### OO Mutation Operators—Polymorphism

10. **PNC — New Method Call With Child Class Type**
An object's actual type is changed to a child of the original actual type in the `new()` statement.

11. **PMD — Member Variable Declaration With Parent Class Type**
An object's declared type is changed to the parent of the original actual type in the declaration statement.

12. **PPD — Parameter Variable Declaration With Child Class Type**
Changes the declared type of a parameter object reference to be the parent of its original declared type.

13. **PCI — Type Cast Operator Insertion**
Change the actual type of an object reference to the parent or to the child of the declared type.

### OO Mutation Operators—Example

#### 10. PNC – New Method Call with Child Class Type
```
point p;
p = new point();
\[\downarrow\]
colorpoint
\[\uparrow\]
```

```
point p;
\[\downarrow\]
colorpoint
\[\uparrow\]
p = new colorpoint();
```

#### 11. PMD – Member Variable Declaration with Parent Class Type
```
point p;
\[\downarrow\]
colorpoint
\[\uparrow\]
p = new point();
\[\downarrow\]
colorpoint
\[\uparrow\]
p = new colorpoint();
````
**OO Mutation Operators—**Example

12. **PPD** – Parameter Variable Declaration with Child Class Type

```
point

colorpoint
```

```
boolean equals (point p)
{ . . . }
```

```
\Delta boolean equals (colorpoint)
{ . . . }
```

13. **PCI** – Type Case Operator Insertion

```
point

colorpoint
```

```
colorpoint pc;
point p = cp;
p.getX();
\Delta ((colorpoint) p).getX();
```

---

**OO Mutation Operators—**Polymorphism

14. **PCD** — Type Cast Operator Deletion

Delete a type casting operator

15. **PCC** — Cast Type Change

Change the type to which an object reference is being cast

16. **PRV** — Reference Assignment With Other Comparable Variable

Changes operands of reference assignments to be assigned to objects of subclasses.

17. **OMR** — Overloading Method Contents Replace

For each pair of methods that have the same name, the bodies are interchanged.
**OO Mutation Operators—Example**

14. **PCD – Type Case Operator Deletion**

- `point` to `colorpoint`
- `point p = cp;` to `((colorpoint) p).getX();`

15. **PCC – Cast Type Change**

- `point` to `colorpoint`
- `((point p).getX();` to `((colorpoint) p).getX();`

16. **PRV – Reference Assignment with Other Comparable Variable**

- `point p;` to `colorpoint cp = new colorpoint(0, 0);`
- `point3D p3d = new point3D(0, 0, 0);`
- `p = cp;` to `Δ p = p3d;`  

17. **OMR – Overloading Method Contents Replace**

- `void set (int x, int y) { ...}  Δ // void set (int x, int y) { ...}`
- `void set (int x, int y, int z) { ...}`
**OO Mutation Operators—Polymorphism**

18. OMD — Overloading Method Deletion

Each overloaded method declaration is deleted, one at a time.

19. OAC — Arguments Of Overloading Method Call Change

Changes the number and order of arguments in method invocations, but only if there is an overloading method that can accept the new argument list.

---

**OO Mutation Operators—Example**

18. OMD – Overloading Method Deletion

\[
\text{point}
\]

\[
\text{point3D}
\]

\[
\Delta \quad \text{void set (int x, int y) \{ ... \}}
\]

\[
\Delta \quad \text{void set (int x, int y) \{ ... \}}
\]

\[
\text{void set (int x, int y, int z) \{ ... \}}
\]

---

19. OAC – Arguments of Overloading Method Call Change

\[
\text{point}
\]

\[
\text{point3D}
\]

\[
\text{void set (int x, int y, int z);}
\]

\[
\text{void set (int x, int y);}
\]

\[
\text{void set (int z);}
\]

\[
\text{point3D p;}
\]

\[
\text{p.set (1, 2, 3);}
\]

\[
\Delta \quad \text{p.set (1, 3, 2);}
\]

\[
\Delta \quad \text{p.set (2, 3);}
\]

\[
\Delta \quad \text{p.set (3);}
\]
**OO Mutation Operators—Language Specific**

20. **JTI — This Keyword Insertion**

The keyword `this` is added when appropriate.

21. **JTD — This Keyword Deletion**

Each occurrence of the keyword `this` is deleted.

22. **JSI — Static Modifier Insertion**

The `static` modifier is added to instance variables.

23. **JSD — Static Modifier Deletion**

Each instance of the `static` modifier is removed.

---

**OO Mutation Operators—Example**

20. **JTI — This Keyword Insertion**

```java
... void set (int x, int y) {
    this.x = x;
    ∆ this.x = this.x;
    this.y = y;
    ∆ this.y = this.y;
} ...
```

21. **JTD — This Keyword Deletion**

```java
... void set (int x, int y) {
    this.x = x;
    ∆ x = x;
    this.y = y;
    ∆ y = y;
} ...
```
## OO Mutation Operators—*Example*

### 22. JSI – Static Modifier Insertion

<table>
<thead>
<tr>
<th>point</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int x = 0;</td>
</tr>
<tr>
<td>∆ public static int x = 0;</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

### 23. JSD – Static Modifier Deletion

<table>
<thead>
<tr>
<th>point</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static int x = 0;</td>
</tr>
<tr>
<td>∆ public int x = 0;</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

## OO Mutation Operators—*Language Specific*

### 24. JID — Member Variable Initialization Deletion

Remove initialization of each member variable.

### 25. JDC — Java-supported Default Constructor Creation

Remove the implemented default constructor, forcing Java to create a default constructor.

### 26. EOA — Reference Assignment and Content Assignment Replacement

Replaces the right hand side of an object assignment with an assignment of a clone of the object.

### 27. EOC — Reference Comparison and Content Comparison Replacement

Replaces a comparison of two objects using the "==" operator with the "equals()" method.
**OO Mutation Operators—Example**

24. **JID** — Member Variable Initialization Deletion

```
int x = 0;
∆ int x;
...
```

25. **JDC** — Java-supported Default Constructor Creation

```
point()
∆ // point()
...
```

---

**OO Mutation Operators—Example**

26. **EOA** — Reference Assignment and Content Assignment Replacement

```
point p1, p2;
p1 = new point (1, 2);
p2 = p1;
∆ p2 = p1.clone();
```

27. **EOC** — Reference Comparison and Content Comparison Replacement

```
point p1 = new point (1, 2);
point p2 = new point (1, 2);
...
if (p1 == p2)
∆ if (p1.equals (p2))
{
    ...
}
Replaces calls to “getter” methods (getX()) with calls to other compatible getter methods (getY()).

Replaces calls to “setter” methods (setX()) with calls to other compatible getter methods (setY()).

\[\begin{array}{|c|c|}
\hline
\text{point} & \text{point} \\
\hline
\text{public int getX () \{ ... \}} & \text{public int getX () \{ ... \}} \\
\text{public int getY () \{ ... \}} & \text{public int getY () \{ ... \}} \\
\text{public void setX (int x) \{ ... \}} & \text{public void setX (int x) \{ ... \}} \\
\text{public void setY (int y) \{ ... \}} & \text{public void setY (int y) \{ ... \}} \\
\hline
\end{array}\]

point p1 = new point (1, 2):
\[
\begin{align*}
\cdots \\
p1.getX(); \\
\Delta p1.getY(); \\
\end{align*}
\]

point p1 = new point (1, 2):
\[
\begin{align*}
\cdots \\
p1.setX(4); \\
\Delta p1.setY(4); \\
\end{align*}
\]
Integration Mutation Summary

• Integration testing often looks at couplings
• We have not used grammar testing at the integration level
• Mutation testing modifies callers and callees
• OO mutation focuses on inheritance, polymorphism, dynamic binding, information hiding and overloading
  – The access levels make it easy to make mistakes in OO software
• muJava is an educational / research tool for mutation testing of Java programs
  – http://ise.gmu.edu/~offutt/mujava/