Testing OO Software - What’s different?

- Less complexity in procedures
  Short methods
- Complexity is relocated
to the connections among components
- Less problems
based on intra-procedural and control flow
- More problems
related to interaction between classes
- Less static determinism
  many faults can now only be detected at runtime

Mutation testing defines a set of operators that simulate typical defects in OO systems. These mutation operators generate far less mutants than classical mutation operators, but also a higher percentage of equivalent mutants. Test cases to kill these mutants have to exercise classes in ways to distinguish them from ancestors or other polymorphic types, for example.

Object-oriented software only affects the module and integration test levels, the remaining test levels are independent of the type of software.

---

### OO Mutation

<table>
<thead>
<tr>
<th>Information Hiding</th>
<th>Overloading</th>
<th>Java Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMC - Access Modifier Change</td>
<td>RTC - Reference Type Change</td>
<td>TKD - this Keyword Deletion</td>
</tr>
<tr>
<td>HVD - Hiding Variable Deletion</td>
<td>OMC - Overloading Method Change</td>
<td>SMV - Static Modifier Change</td>
</tr>
<tr>
<td>HVI - Hiding Variable Insertion</td>
<td>OMD - Overloading Method Deletion</td>
<td>VID - Variable Initialization Deletion</td>
</tr>
<tr>
<td>OMD - Overriding Method Deletion</td>
<td>OMM - Overridden Method Moving</td>
<td>JDC - Java Supported Default Constructor</td>
</tr>
<tr>
<td>OMR - Overridden Method Rename</td>
<td>SKR - Super Keyword Deletion</td>
<td>ATC - Actual Type Change</td>
</tr>
<tr>
<td>SKR - Super Keyword Deletion</td>
<td>PCD - Parent Constructor Deletion</td>
<td>DTC - Declared Type Change</td>
</tr>
<tr>
<td>ATC - Actual Type Change</td>
<td>ANC - Argument Number Change</td>
<td>RTC - Reference Type Change</td>
</tr>
<tr>
<td>DTC - Declared Type Change</td>
<td>PTC - Parameter Type Change</td>
<td></td>
</tr>
</tbody>
</table>

- **Inheritance**
- **Polymorphism**

---

Actual Needs and Constraints → User Acceptance (alpha, beta test) → Delivered Package

- System Specifications → System Test → System Integration
  - Analysis / Review
- Subsystem Design/Specs → Integration Test → Subsystem
  - Analysis / Review
- Unit/Component Specs → Module Test → Unit/Components

User review of external behavior as it is determined or becomes visible
**OO Testing Levels**

- Intra-method testing: Testing individual methods within classes
- Inter-method testing: Multiple methods within a class are tested in concert
- Intra-class testing: Testing a single class, usually using sequences of calls to methods within the class
- Inter-class testing: More than one class is tested at the same time (integration)

A coupling sequence is a sequence of method calls in which a variable shared by a common instance context is defined in one method and used in another method.

A simple analysis of the informal specification of class Slots allows to identify states and transitions. This state machine allows to derive sequences of method calls for intra-class testing, for example by choosing method call sequences that cover all states or transitions in the FSM.

Test cases for intra- and inter-method testing are method calls, but for intra- and inter-class testing a test case consists of a sequence of method calls. In addition, setup code to get objects into the necessary states is required.

### Example Coupling Sequence

A coupling sequence with respect to Z::x

### Deriving an FSM and test cases

- TC-1: incorporate, isBound, bind, isBound
- TC-2: incorporate, unBind, bind, unBind, isBound