Traditional Testing Approaches

• Black-box testing
  – Program specification
  – Operational profile

• White-box testing
  – Statement coverage
  – Branch coverage
  – Data-Flow coverage
  – Mutation coverage
  – Path coverage
The OO Pitfalls - Inheritance

A subclass may re-define its inherited functions and other functions may or may not be affected by the re-defined functions.

```cpp
Class foo {
    int local_var;
    :
    int f1() { return 1; }
    int f2() { return 1/f1(); }
}
Class foo_child :: Public foo {
    // child class of foo
    int f1() { return 0; }
}
```
The OO Pitfalls - Polymorphism

An object may be bound to different classes during the run time.

// beginning of function foo
:
P1 p;
P2 c;
:
return (c.f1()/p.f2());
// end of function foo
The oo Pitfalls - State

STACK

public:

STACK (int stack_size);
~STACK(); { free (ele_array) };
int push (int new_item);
int pop (int *pop_item);
BOOLEAN is_empty();
BOOLEAN is_full();

protected:

int max_size;
int top;
int * ele_array;
BOOLEAN empty;
BOOLEAN full;

main()
{
  .
  cin >> max_size;
  .
  STACK st = STACK ( max_size ); — (1)
  .
  while (. . ) {
    if (.)
      st.push(); ———— (2)
    else
      st.pop(); ———— (3)
  }
}

STACK :: STACK (int stack_size)
{
  ele_array = (int *) malloc (stack_size);
top = 0;
max_size = stack_size;
full = FALSE;
}

STACK :: push (int new_item)
{
  if ( is_full() )
    return (ERROR);
  ele_array[top++] = new_item;
  if ( top >= max_size )
    full = TRUE;
  return (SUCCESS);
}

STACK :: pop (int *pop_item)
{
  if ( is_empty() )
    return (ERROR);
  pop_item = ele_array[top − 1];
top--;
  if ( top <= 0)
    empty = TRUE;
  return (SUCCESS);
}

STACK :: is_empty()
{
  return ( empty);
}

STACK :: is_full()
{
  return( full );
}
The OO Pitfalls - State

\begin{itemize}
  \item \textbf{S1}:
    \begin{itemize}
      \item empty: TRUE
      \item full : FALSE
      \item top : 0
    \end{itemize}

  \item \textbf{S2}:
    \begin{itemize}
      \item empty: FALSE
      \item full : FALSE
      \item top : (0, max\_size)
    \end{itemize}

  \item \textbf{S3}:
    \begin{itemize}
      \item empty: FALSE
      \item full : TRUE
      \item top : max\_size
    \end{itemize}

  \item \textbf{S4}
\end{itemize}

\begin{itemize}
  \item \textbf{push (max\_size > 1)}
  \item \textbf{push (max\_size = 1)}
  \item \textbf{pop (max\_size > 1)}
  \item \textbf{pop (max\_size = 1)}
\end{itemize}
Fault Classification

**Type I:** the object-oriented faults that are strongly related to the object-oriented features and are introduced by these features such as inheritance and polymorphism sm.

**Type II:** the object management faults that related to object management such as object copying, dangling reference and object memory usage faults.

**Type III:** the traditional faults that are not relevant to objects and they fall into the fault classification of the traditional software.
Object-Oriented Pitfalls

Example 1

```cpp
class OBJECT{
    Int *m_data;
    OBJECT() { m_data = new Int; *m_data = 0; }
    ~OBJECT() { delete m_data; }
    Print() { cout << m_data; }
    Inc() { m_data++; }
}

if ( P1 ) {
    obj2->Inc();
}
obj2->Print();
if ( P2 ) {
    obj2->~OBJECT();
}
obj1->Print();
if ( P3 ) {
}
}

main()
{
    OBJECT *obj1 = new OBJECT();
    obj1->Inc();
    OBJECT *obj2 = new OBJECT();
    *obj2 = *obj1
    L1:
    L2:
    L3:
    L4:
    L5:
    L6:
    L7:
    L8:
    L9:
    obj1->~OBJECT();
}
```

Example 2

```cpp
class foo {
    int local_var;
    int f1() { return 1; }
    int f2() { return 1/f1(); }
}

class foo_derived :: public foo {
    int f1() { return 0; }
}

main()
{
    int x, y;
    foo Obj;
    cin >> x, y;
    if ( x > 0 )
        Obj = new foo();
    else
        Obj = new foo_derived();
    if ( y > 0 )
        cout << Obj.f2();
    else
        cout << Obj.f1();
}
```

```cpp
class foo {
    int local_var;
    int f1() { return 1; }
    int f2() { return 1/f1(); }
}

class foo_derived :: public foo {
    int f1() { return 0; }
}

int x, y;
foo Obj;
    cin >> x, y;
    if ( x > 0 )
        Obj = new foo();
    else
        Obj = new foo_derived();
    if ( y > 0 )
        cout << Obj.f2();
    else
        cout << Obj.f1();
}
```

```cpp
int x, y;
foo Obj;
    cin >> x, y;
    if ( x > 0 )
        Obj = new foo();
    else
        Obj = new foo_derived();
    if ( y > 0 )
        cout << Obj.f2();
    else
        cout << Obj.f1();
}
```
## Fault Analysis

<table>
<thead>
<tr>
<th></th>
<th>System A</th>
<th>System B</th>
<th>System C</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLOC</td>
<td>5.6</td>
<td>21.3</td>
<td>16.0</td>
</tr>
<tr>
<td>NOC</td>
<td>20</td>
<td>45</td>
<td>27</td>
</tr>
<tr>
<td>NOF</td>
<td>35</td>
<td>80</td>
<td>85</td>
</tr>
<tr>
<td>Type I</td>
<td>14.4%</td>
<td>18.75%</td>
<td>11.7%</td>
</tr>
<tr>
<td>Type II</td>
<td>17.2%</td>
<td>16.25%</td>
<td>8.2%</td>
</tr>
<tr>
<td>Type III</td>
<td>68.4%</td>
<td>65%</td>
<td>80.1%</td>
</tr>
</tbody>
</table>

Nov. 9, 2000
Object-Oriented Testing Process

- Unit testing - class
- Integration testing - cluster
- System testing - program
Class Testing Strategies

- Testing inheritance
- Testing polymorphism
- State-oriented testing
- Data Flow testing
- Function dependence class testing
Testing Inheritance

“Incremental testing of object-oriented class structures.” Harrold et al., (1992)

**New methods:** complete testing

**Recursive methods:** limited testing

**Redefined methods:** reuse test scripts
Testing Polymorphism

“Testing the Polymorphic Interactions between Classes.”

McDaniel and McGregor (1994)

Clemson University
State-Oriented Testing

- “The state-based testing of object-oriented programs,” 1992, C. D. Turner and D. J. Robson


- The FREE approach: Binder
  http://www.rbsc.com/pages/Free.html
Data Flow Testing


Function Dependence Class testing


- Testing inheritance

- Testing polymorphism
Function Dependence Relationship

- A function *uses* a variable means that the value of the variable is referenced in a computation expression or used to decide a predicate.

- A function *defines* a variable means that the value of the variable is assigned when the function is invoked.

- A variable $x$ *uses* a variable $y$ means the value of $x$ is obtained from the value of $y$ and others. $x$ is affected when the value of $y$ is changed.

1. $f_1$ uses a variable $x$ that is defined in $f_2$,

2. $f_1$ calls $f_2$ and uses the return value of $f_2$,

3. $f_1$ is called by $f_2$ and uses a parameter $p$ that is defined in $f_2$.

4. $f_1$ uses a variable $x$ and $x$ uses a variable $y$ which is defined in $f_2$.
Object-Flow Based Testing Strategy

Object: An object is an instance of a class in the runtime which comprises an object state, expressed in terms of the actual values of a set of attributes, together with object methods operating on this state.

Define: An object is defined if its state is initiated or changed

Use: An object is used if one of its data member(s) is referenced or one of its methods that does not define any data member is invoked.
Object-Flow Coverage Criteria

**All-bindings:** Every possible binding of each object must be exercised at least once when the object is defined or used. If a statement involves multiple objects, then every combination of a possible binding needs to be tested at least once.

**All-du-pairs:** At least one definition-clear path from every definition of every object to every use of that definition must be exercised under some test.
Applications

1. System A - GUI

2. System B - Data logging system

3. System C - Network communication program
### Functional Testing

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of test cases</td>
<td>100</td>
<td>403</td>
<td>326</td>
</tr>
<tr>
<td>Number of faults detected</td>
<td>17</td>
<td>42</td>
<td>52</td>
</tr>
<tr>
<td>Type I faults detected</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Type II faults detected</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Type III faults detected</td>
<td>15</td>
<td>32</td>
<td>44</td>
</tr>
<tr>
<td>Total faults detected(%)</td>
<td>48%</td>
<td>52%</td>
<td>61%</td>
</tr>
<tr>
<td>Total OO faults detected(%)</td>
<td>18%</td>
<td>35%</td>
<td>47%</td>
</tr>
</tbody>
</table>
Testing Strategies

Strategy I:

1. *statement testing*
2. *branch testing*

Strategy II:

1. *all-states*
2. *all-transitions*

Strategy III:

1. *all-du-pairs*
2. *all-bindings*
# Data Analysis

<table>
<thead>
<tr>
<th>Strategy I</th>
<th>Strategy II</th>
<th>Strategy III</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td><strong>B</strong></td>
<td><strong>C</strong></td>
</tr>
<tr>
<td>Statement</td>
<td>All-states</td>
<td>All-du-pairs</td>
</tr>
<tr>
<td>46</td>
<td>55</td>
<td>52</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>92%</td>
<td>91%</td>
<td>92%</td>
</tr>
<tr>
<td>8%</td>
<td>9%</td>
<td>8%</td>
</tr>
<tr>
<td>Branch</td>
<td>All-transitions</td>
<td>All-bindings</td>
</tr>
<tr>
<td>23</td>
<td>41</td>
<td>33</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>89%</td>
<td>89%</td>
<td>92%</td>
</tr>
<tr>
<td>11%</td>
<td>11%</td>
<td>8%</td>
</tr>
</tbody>
</table>
## Data Analysis

<table>
<thead>
<tr>
<th></th>
<th>System A</th>
<th>System B</th>
<th>System C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>II</td>
<td>III</td>
</tr>
<tr>
<td><strong>NOT</strong></td>
<td>69</td>
<td>64</td>
<td>75</td>
</tr>
<tr>
<td><strong>OOF</strong></td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td><strong>TOF</strong></td>
<td>8</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>$\rho_1$</td>
<td>22%</td>
<td>44%</td>
<td>88%</td>
</tr>
<tr>
<td>$\rho_2$</td>
<td>44%</td>
<td>66%</td>
<td>72%</td>
</tr>
<tr>
<td>$\rho_3$</td>
<td>53%</td>
<td>72%</td>
<td>79%</td>
</tr>
<tr>
<td>$\rho_4$</td>
<td>79%</td>
<td>83%</td>
<td>82%</td>
</tr>
</tbody>
</table>
## Data Analysis

### System A

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>du</th>
<th>b</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>31</td>
<td>22</td>
<td>16</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3(27%)</td>
<td>4(36%)</td>
<td>8(72%)</td>
<td>10(90%)</td>
<td>10(90%)</td>
<td>11(100%)</td>
</tr>
<tr>
<td>23(65%)</td>
<td>25(71%)</td>
<td>30(85%)</td>
<td>33(94%)</td>
<td>33(94%)</td>
<td>34(97%)</td>
</tr>
</tbody>
</table>

### System B

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>du</th>
<th>b</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>41</td>
<td>38</td>
<td>21</td>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12(42%)</td>
<td>14(50%)</td>
<td>22(78%)</td>
<td>26(92%)</td>
<td>27(96%)</td>
<td>28(100%)</td>
</tr>
<tr>
<td>50(62%)</td>
<td>58(72%)</td>
<td>68(85%)</td>
<td>73(91%)</td>
<td>75(93%)</td>
<td>78(97%)</td>
</tr>
</tbody>
</table>

### System C

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>du</th>
<th>b</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>33</td>
<td>34</td>
<td>26</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>9(52%)</td>
<td>10(58%)</td>
<td>13(76%)</td>
<td>16(94%)</td>
<td>16(94%)</td>
<td>17(100%)</td>
</tr>
<tr>
<td>57(67%)</td>
<td>63(74%)</td>
<td>69(81%)</td>
<td>73(85%)</td>
<td>77(90%)</td>
<td>81(95%)</td>
</tr>
</tbody>
</table>

---

Integrated Approach
## Data Analysis

<table>
<thead>
<tr>
<th>System Strategy</th>
<th>System A</th>
<th>System B</th>
<th>System C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>F</td>
<td>I</td>
<td>F</td>
</tr>
<tr>
<td>NOT</td>
<td>110</td>
<td>126</td>
<td>383</td>
</tr>
<tr>
<td>OOF</td>
<td>2</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>TOF</td>
<td>17</td>
<td>17</td>
<td>42</td>
</tr>
<tr>
<td>$\rho_1$</td>
<td>N/A</td>
<td>74%</td>
<td>N/A</td>
</tr>
<tr>
<td>$\rho_2$</td>
<td>N/A</td>
<td>89%</td>
<td>N/A</td>
</tr>
</tbody>
</table>