Coupling-based Criteria for Integration Testing

Jeff Offutt


TOOLS USA 1999, Santa Barbara, CA, August 1999, Roger Alexander and Jeff Offutt.
Motivation

As OO is becoming increasingly prevalent, the emphasis on testing is shifting from unit and system to module and integration.

Module testing

OO testing

Integration testing
Motivation

FAA has recognized this by imposing requirements on structural coverage of software:

“the analysis (testing) should confirm the data coupling and control coupling between the code components”
Test Specifications & Requirements

Software

• Spec/Req -- **What** the software does
• Design     -- **How** the software does it

Tests

• **Requirements**: Specific things that must be satisfied or covered
  (statements, branches, etc.)
• **Criterion**: A set of rules that imposes test requirements
• **Coverage**: Extent to which a criterion is satisfied
Advantages of Formal Test Criteria

- Formal basis for testing
- Well defined stopping rules
- Automation
- Early development of tests

Formal test criteria allow tests to be cheaper and better – reducing the risk of using the resulting software.
Software Testing Levels

**Unit and Module Testing**: Testing individual procedures and groups of related procedures.

**Integration Testing**: Testing for interface problems and incompatibilities between objects.

**System Testing**: Testing a complete system from an external perspective.
OO Testing Issues

Unit Testing
- Testing is essentially the same
- Must watch for “hidden” methods and operations
- If reuse, must know extent of previous testing

Class and Integration Testing
- More integration testing is necessary
- Module testing at the class level is critical
- More faults will be found

System Testing
- Some specific techniques are helpful for OO
Integration Testing

Do the components communicate correctly?

Do they make conflicting assumptions?

Do they leave anything out?
Testing Interface Couplings

Integration testing is primarily about finding faults in the interfaces.

Interfaces are often measured by the couplings between classes.

Integration testing can be guided by covering these couplings.
Coupling Overview

Coupling measures the dependency relations between two units.

- Reflects interconnections between units
- Faults in one unit may affect the coupled unit
- Faults in the interface affects both units

Coupling summarizes design and structure

Coupling has been found to be correlated with faults
Old Coupling Types

0. Independent Coupling
1. Call Coupling
2. Scalar Data Coupling
3. Stamp Data Coupling
4. Scalar Control Coupling
5. Stamp Control Coupling
6. Scalar Data/Control Coupling
7. Stamp Data/Control Coupling
8. External Coupling
9. Non-Local Coupling
10. Global Coupling
11. Tramp Coupling
Coupling-based Testing Terms

- Caller
- Callee
- Actual Parameters
- Formal Parameters
- Interface

The program must execute from definitions of actual parameters through calls to uses of formal parameters.
Three Coupling Types

Parameter Coupling: Information passed by parameters.

Shared Data Coupling: Two procedures that refer to the same data object.

External Device Coupling: Two procedures that access the same external medium.
Coupling-based Path Criteria

- **Parameter Coupling Path**: For each actual parameter $X$ and each last definition of $X$ before a call-site, a path from the last definition through the call-site to each first use of the parameter.

- **Shared Data Coupling Path**: For each shared data object $G$ defined in $A$ and used in $B$, and each definition of $G$ in $A$, a path from the definition to each first use of $G$ in $B$.

- **External Device Coupling Path**: For each pair of references $(i, j)$ to the same external device, both $i$ and $j$ must be executed on the same execution path.
Coupling Paths Example 1-A

procedure QUADRATIC is
  in
  ET (Control_Flag);
  Control_flag = 1 then
    GET (X, Y, Z);
  else
    X := 10;
    Y := 9;
    Z := 12;
  end if;
  OK := TRUE;
  ROOT (X, Y, Z, R1, R2, OK);
  if OK then
    PUT (R1, R2);
  else
    PUT ("No solution.");
  end if;
end QUADRATIC

procedure ROOT ( A, B, C: in REAL;
                 Root1, Root2 : out REAL;
                 Result: in out BOOLEAN) is
  D : REAL;
  begin
    D := B**2-4.0*A*C;
    if Result and D < 0.0 then
      Result := FALSE;
      return;
    end if;
    D := (B + sqrt(D)) / (2.0*A);
    Root1 := (B - sqrt(D)) / (2.0*A);
    Result := TRUE;
end ROOT
procedure QUADRATIC is
  in
  ET (Control_Flag);
  Control_flag = 1 then
  GET (X, Y, Z);
  else
  X := 10;
  Y := 9;
  Z := 12;
  end if;
  OK := TRUE;
  ROOT (X, Y, Z, R1, R2, OK);
  if OK then
  PUT (R1, R2);
  else
  PUT ("No solution.");
  end if;
end QUADRATIC

procedure ROOT ( A, B, C: in REAL;
  Root1, Root2 : out REAL;
  Result: in out BOOLEAN) is
  D : REAL;
  begin
  D := B**2-4.0*A*C;
  if Result and D < 0.0 then
    Result := FALSE;
    return;
  end if;
  Root1 := (-B + sqrt(D)) / (2.0*A);
  Root2 := (-B - sqrt(D)) / (2.0*A);
  Result := TRUE;
end ROOT
Coupling Paths Example 1-C

procedure QUADRATIC is
  in
  ET (Control_Flag);
  Control_flag = 1 then
  GET (X, Y, Z);
  else
    X := 10;
    Y := 9;
    Z := 12;
  end if;
  OK := TRUE;
  ROOT (X, Y, Z, R1, R2, OK);
  if OK then
    PUT (R1, R2);
  else
    PUT ("No solution.");
  end if;
end QUADRATIC

procedure ROOT ( A, B, C: in REAL;
                Root1, Root2 : out REAL;
                Result: in out BOOLEAN) is
  D : REAL;
  begin
    D := B**2 - 4.0*A*C;
    if Result and D < 0.0 then
      Result := FALSE;
      return;
    end if;
    Root1 := (-B + sqrt(D)) / (2.0*A);
    Root2 := (-B - sqrt(D)) / (2.0*A);
    Result := TRUE;
  end ROOT
Coupling Paths Example 2

global / nonlocal variable G

1

\[ \cdots \]
\[ \text{G} := \ldots ; \quad \text{def (G)} \]

\[ \cdots \]

Q1

\[ \cdots \quad \text{if G > 100 then use (G)} \]

\[ \cdots \]

P2

\[ \cdots \]
\[ \text{G} := \ldots ; \quad \text{def (G)} \]

\[ \cdots \]

Q2

\[ \cdots \quad t := G^{*2}+X; \quad \text{use(G)} \]

\[ \cdots \]
Coupling-based Test Criteria

- **Call Coupling:** All call-sites are executed.

- **All-Coupling-Defs:** A path is executed from each last def of $X$ in $A$ to a first use in $B$.

- **All-Coupling-Uses:** A path is executed from each last def of $X$ in $A$ to every first use in $B$.

- **All-Coupling-Paths:** All subpath sets are executed from each last def of $X$ in $A$ to every first use in $B$. (Loops are bypassed and executed at least once.)
Coupling Criteria Example

\begin{center}
\begin{tikzpicture}[node distance=2cm, auto]
    
    \node (x5) {X = 5} [draw] ;
    \node (x4) [below of=x5] {X = 4} [draw] ;
    \node (x3) [below of=x4] {X = 3} [draw] ;
    \node (b) [below of=x3] {B (X)} [draw] ;
    \node (y) [right of=x5] {B (int y)} [draw] ;
    \node (z) [right of=y] {Z = y} [draw] ;
    \node (t) [right of=z] {T = y} [draw] ;
    \node (print) [right of=t] {print (y)} [draw] ;

    \draw [->] (x5) -- (x4);
    \draw [->] (x4) -- (x3);
    \draw [->] (x3) -- (b);
    \draw [->] (y) -- (z);
    \draw [->] (y) -- (t);
    \draw [->] (z) -- (print);
    \draw [->] (t) -- (print);

\end{tikzpicture}
\end{center}
Coupling Criteria Example

First Uses
11, 12

Last Defs
2, 3

X = 5
X = 4
X = 3
B (X)

B (int y)

Z = y
T = y

print (y)
Coupling-based Criteria Subsumption

All-coupling-paths

All-coupling-uses

All-coupling-defs

Call Coupling
Measuring Coupling-based Criteria

- Instrumentation
- Relies on Call Tree
Instrumenting for Coverage Analysis

- $P'$ has all the functionality of $P$ plus extra statements
- Extra statements keep track of whether portions of the software have been executed
- Counters are kept for calls, last definitions, and first uses
Call Tree
Call Tree Instrumentation

```c
ab [] = 0;
// element for all-site.

int CTab[] = 0;
// one element for each site.

CTab[1] ++;
CTab[2] ++;
CTab[3] ++;
CTab[4] ++;
CTab[5] ++;
CTab[6] ++;
CTab[7] ++;
```

Diagram:

- A
  - B
    - E
    - F
  - C
  - D
    - B
    - C
    - G

 edge labels:
- CTab[1] ++;
- CTab[2] ++;
- CTab[3] ++;
- CTab[4] ++;
- CTab[5] ++;
- CTab[6] ++;
- CTab[7] ++;
Call Tree Instrumentation

- After testing, if CTab[i] is 0, that call-site has not been covered.

- CTab[] must be saved to disk between executions.

- CTab[] is used as a basis from which to generate a report.
All-Coupling-Defs Instrumentation

1. $X = 5$

2. $X = 4$
   - LastDef[$X$] = 2;

3. $X = 3$
   - LastDef[$X$] = 3;

4. $B(X)$

10. $B$ (int $y$)

11. $Z = y$

12. $T = y$

13. print ($y$)

CTab[LastDef[$y$]] ++;
All-Coupling-Defs Instrumentation

1. $X = 5$

2. $X = 4$
   - $X = 2$

3. $X = 3$
   - $X = 3$
   - $X = 2$

4. $B(X)$

CTab [LastDef [ ActualOf [ ] ] ] ++

10. $B(int y)$

11. $Z = y$
    - $T = y$

12. print (y)

13. ActualOf [y] = X;
All-Coupling-Uses Instrumentation

1. $X = 5$
2. $X = 4$
3. $X = 3$
4. $B(X)$

LastDef[$X$] = 2;

CTab[11, LastDef[ActualOf[$y$]]] ++;

5. $Z = y$
6. $T = y$
7. $B$(int $y$)
8. $print(y)$

Copyright Jeff Offutt, 2000
Measuring Coupling-based Criteria

\[
\begin{align*}
\text{call - coupling} & = \frac{\text{edges-covered}}{\text{edges}} \\
\text{coupling - def} & = \frac{\text{coupling-defs covered}}{\text{coupling-defs}} \\
\text{coupling - use} & = \frac{\text{def-use pairs covered}}{\text{def-use pairs}} \\
\text{coupling - path} & = \frac{\text{coupling-paths covered}}{\text{coupling-paths}}
\end{align*}
\]
## Evaluation of Coupling-based Testing

<table>
<thead>
<tr>
<th></th>
<th>Category partition</th>
<th>Coupling based</th>
<th>Inter-procedural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test cases</td>
<td>72</td>
<td>37</td>
<td>53</td>
</tr>
<tr>
<td>Faults found</td>
<td>7</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>Faults missed</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Current Work

- Coverage analysis tool (based on instrumentation) being developed
- Empirical evaluation on larger programs ongoing
- Object-oriented language features (inheritance and polymorphism):
  - Analysis techniques developed
  - Testing criteria defined
  - Coverage analysis tool under development

- Metrics

- Test data generation
Coverage Analysis Tool

Java Grammar → JCC → Parser

Java Tree Builder

AST

Visitors

A class for each node

Instrumentors

P

Pcall coupling

Pall-coupling def

Pall-coupling use

Pall-coupling suit
Testing OO Programs

Assumption: (Re)testig will be easier.
  – Inheritance would eliminate much of the testing.

Reality: A lot more to test!
  – Complexity has moved to the connections.
  – Polymorphism introduces non-determinism.
Overriding Methods

An overriding subclass needs to be tested.

– Methods computing semantically close functions may require different tests (Antiextensionality axiom)

Reasons for new test cases

– Different program texts
– Possibly different behavior (a bad idea!!)
Example

Consider what happens when an overriding method has a different def-set than the overridden method

<table>
<thead>
<tr>
<th>Method</th>
<th>Defs</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>A::m</td>
<td>{A::u,A::v}</td>
<td></td>
</tr>
<tr>
<td>A::n</td>
<td></td>
<td>{A::v}</td>
</tr>
<tr>
<td>B::n</td>
<td></td>
<td>{A::u}</td>
</tr>
<tr>
<td>B::l</td>
<td></td>
<td>{A::v}</td>
</tr>
<tr>
<td>C::m</td>
<td>{A::v}</td>
<td></td>
</tr>
</tbody>
</table>
Polymorphism Headaches (Yo-Yo)

Object is of type A
A::d ()
Polymorphism Headaches (Yo-Yo)

Object is of type B
B::d ()
Polymorphism Headaches (Yo-Yo)

Object is of type C, C::d()
Conclusions

Test coverage criteria for integration testing
Coverage measurement can quantitatively measure testing results
Highly automatable
Satisfies USA’s Federal Aviation Administration requirements on structural coverage
Object-oriented software presents many interesting new twists
Many open problems
True software engineering: Technically challenging and practical