Programs, Test Data, and Oracles: Revisiting the Foundations of Software Testing

Mats P. E. Heimdahl
University of Minnesota Software Engineering Center
Department of Computer Science and Engineering
University of Minnesota
4-192 EE/CS; 200 Union Street SE
Minneapolis, MN 55455
Software Development

Talented Developers?

Specification → Implementes → Program
The Big Question

Does the program accurately represent the specification?

Specification \rightarrow \text{Talented Developers?} \rightarrow \text{Program}

Implements
Testing Process

 Specification

 Talented Developers?

 Test Inputs

 Program

 Executed Over

 Oracle

 Evaluates Correctness

 Path through program

 Correct/incorrect
Testing Process

Specification

Talented Developers?

Test Inputs

Program

Executed Over

Oracle

Evaluates Correctness

Path through program

Correct/incorrect
Domains of Concern
Most research focuses on

Program

Is generated from

Specification

Is generated from

Oracle

Test Inputs

Most research focuses on
Fault Finding; MC/DC

- Program structure matters
Fault Finding; Branch Coverage

- DWM_1
- DWM_2
- Latctl_Batch
- Vertmax_Batch

Output Only
Maximum

4/3/15 UCI, ISR'15
Testing Artifacts - Relationships

- Specification directs oracle construction
- Program implements specification
- Program bounds observability
- Design program for testing
- Structure directs test inputs
- Specification directs test input
- Given test inputs, construct oracle
- Given oracle construct test inputs
- Design program for testing
Most research focuses on Specification. Program is generated from Specification. Oracle Test Inputs are generated from Program.
Importance of Understanding Relationship Between Artifacts

Unexplored testing artifacts represent potential for improving testing effectiveness

Uncontrolled factors represent a threat to validity of empirical studies

Poorly understood factors may result in misapplication of methods
Acknowledgements

• I have not done this alone
  – Matt Staats, Google, Zurich
  – Mike Whalen, U of Minnesota
  – Ajitha Rajan, Edinburgh
  – Gregory Gay, U of South Carolina
  – Rockwell Collins Inc.
    • Steve Miller, Darren Cofer

Funded by CNS-0931931 and CNS-1035715
Two Approaches

Theory of Testing

Theory of Testing

Empirical Studies

String Theory Summarized:

I just had an awesome idea. Suppose all matter and energy is made of tiny, vibrating "strings."

Okay, what would that imply?

I dunno.

My Hobby: Extrapolating

As you can see, by late next month you'll have over four dozen husbands. Better get a bulk rate on wedding cake.

Adopted from Matt Staats
Two Approaches

Theory of Testing
Ideal Test Coverage Criterion: Finds All Faults
Theory of Testing - History

Ideal Test Coverage Criterion: Finds All Faults
Theory of Testing - History

Ideal Test Coverage Criterion: Finds All Faults

- Goodenough / Gerhart
- Budd / Angluin
- Weyuker / Ostrand
- Hamlet

Test Coverage Criteria
Power
(A \geq B)

Gourlay
Theory of Testing - History

Ideal Test Coverage Criterion: Finds All Faults

Power \((A \geq B)\)

Goodenough / Gerhart → Weyuker / Ostrand → Gourlay → Weyuker / Hamlet / Weiss

Probabilistically better

Test Coverage Criteria

Theoretical Comparison of Test Coverage Criteria
Theory of Testing - History

Author Refutes Idea

Author Suggests Idea

- Goodenough / Gerhart
- Budd / Angluin
- Hamlet
- Gourlay
- Weyuker / Hamlet / Weiss

Ideal Test Coverage Criterion: Finds All Faults

Theoretical Comparison of Test Coverage Criteria

Probabilistically better

Power (A ≥ B)
A Mathematical Framework for the Investigation of Testing

John Gourlay

IEEE Transactions on Software Engineering, 1983
Gourlay’s Framework

- $S$ is a set of specifications
- $P$ is a set of programs
- $T$ is a set of tests
- $\text{corr}: P \times S$
- $\text{ok}: T \times P \times S$ (test oracle)
- $\text{corr}(p, s) \rightarrow \text{ok}(t, p, s)$
Gourlay’s Framework - Problems

- $S$ is a set of specifications
- $P$ is a set of programs
- $T$ is a set of tests
- $corr: P \times S$
- $ok: T \times P \times S$
- $corr(p, s) \rightarrow ok(t, p, s)$

Problem: no partial correctness
Gourlay’s Framework - Problems

- $S$ is a set of specifications
- $P$ is a set of programs
- $T$ is a set of tests
- $corr: P \times S$
- $ok: T \times P \times S$
- $corr(p, s) \rightarrow ok(t, p, s)$

Problem: $ok$ is fixed, cannot vary test oracle
Gourlay’s Framework - Extension

- $S$ is a set of specifications
- $P$ is a set of programs
- $T$ is a set of tests
- $O$ is a set of test oracles
- $\text{corr}: P \times S$
- $\text{corr}_t: T \times P \times S$
- $\forall t \in T, \text{corr}_t(t, p, s) \rightarrow \text{corr}(p, s)$

**Solution #1:** add predicate $\text{corr}_t$

**Solution #2:** replace $\text{ok}$ with set of predicates $O$, $\forall o \in O, o: T \times P$

---

Formalize concepts related to test oracles

- Oracle relationship to correctness
  - **Complete:** $\text{corr}_t(t, p, s) \rightarrow o(t, p)$
  - **Sound:** $o(t, p) \rightarrow \text{corr}_t(t, p, s)$
  - **Precise:** $o(t, p) \leftrightarrow \text{corr}_t(t, p, s)$

- Adequacy of testing process
  - **Oracle adequacy criterion:** $O_C: P \times S \times O$
  - **Complete adequacy criterion:** $T O_C: P \times S \times 2^T \times O$

- Formal oracle comparisons
  - Power comparison
  - Probabilistic comparison

- Some previous work is most likely not valid in the face of varying oracles (and program structures)
Two Approaches

Empirical Studies

University of Minnesota
Software Engineering Center
Test Metrics

- **Idea**: Measure how well tests cover the structure of code as an approximation of “goodness” of testing
  - Examples:
    - Statement coverage
    - Decision coverage
    - Modified Condition/Decision Coverage (MC/DC)
  - Used as adequacy criteria for critical avionics software
- **Effective** at finding faults;
  - Better than random testing for suites of the same size
  - Better than other metrics
  - It explicitly accounts for oracle
- **Robust** to simple changes in program structure
- **Reasonable** in terms of the number of required tests and coverage analysis
There Are Weaknesses

- Program structure matters
Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

- Every basic condition in a decision in the model should take on all possible outcomes at least once, and
- Each basic condition should be shown to independently affect the decision’s outcome

\[
\begin{align*}
a &= T \\
b &= F \\
c &= T
\end{align*}
\]

\[
( a \land \land b ) \lor c
\]

\[
\begin{array}{ccc}
T & F & T \\
F & & \\
& & T
\end{array}
\]
Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

- Every basic condition in a decision in the model should take on all possible outcomes at least once, and
- Each basic condition should be shown to independently affect the decision’s outcome

\[
\begin{align*}
a &= F \\
b &= F \\
c &= T \\
( a \land b ) \lor c &\equiv F \ F \ T
\end{align*}
\]
To satisfy MC/DC:

- Every basic condition in a decision in the model should take on all possible outcomes at least once, and
- Each basic condition should be shown to independently affect the decision’s outcome

\[
\begin{align*}
a &= T \\
b &= T \\
c &= T
\end{align*}
\]

\[
(a \&\& b) \text{ || } c
\]

\[
\begin{array}{ccc}
T & T & T \\
T & & \\
& & T \\
& & \\
& & T
\end{array}
\]
Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

• Every basic condition in a decision in the model should take on all possible outcomes at least once, and

• Each basic condition should be shown to independently affect the decision’s outcome

\[
\begin{align*}
a &= F \\
b &= T \\
c &= T
\end{align*}
\]

\[
(a \&\& b) \lor c
\]

\[
\begin{array}{ccc}
F & T & T \\
F & T & F \\
T & T & T \\
\end{array}
\]
Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

• Every basic condition in a decision in the model should take on all possible outcomes at least once, and

• Each basic condition should be shown to independently affect the decision’s outcome

\[
\begin{align*}
a &= T \\
b &= T \\
c &= F
\end{align*}
\]

\[
\begin{align*}
(a \&\& b) \mid\mid c
\end{align*}
\]

\[
\begin{array}{cccc}
T & T & F \\
T & T & T
\end{array}
\]
Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

• Every basic condition in a decision in the model should take on all possible outcomes at least once, and

• Each basic condition should be shown to independently affect the decision’s outcome

\[
\begin{align*}
\text{a} &= \text{F} \\
\text{b} &= \text{T} \\
\text{c} &= \text{F}
\end{align*}
\]

\[
(\text{a} \&\& \text{b}) \text{ ||} \text{c}
\]

\[
\begin{array}{ccc}
\text{F} & \text{T} & \text{F} \\
\text{F} & \text{T} & \text{F} \\
\text{F} & \text{T} & \text{F} \\
\text{F} & \text{T} & \text{F}
\end{array}
\]
Version 1:
Non-Inline Implementation

\[
\text{expr1} = \text{in1 or in2}; \\
\text{out1} = \text{expr1 and in3}; 
\]

Version 2:
Inline Implementation

\[
\text{out1} = (\text{in1 or in2}) \text{ and in3}; 
\]

Tests in green satisfy MC/DC for version 1 but not 2
Masking and Measurement of MC/DC

Version 1:
Non-Inlined Implementation

\[ \text{expr1} = \text{in1 and in2}; \]
\[ \text{out1} = \text{expr1 and in3}; \]

Version 2:
Inlined Implementation

\[ \text{out1} = (\text{in1 or in2}) \text{ and in3}; \]

Tests in green satisfy MC/DC for version 1 but not 2

Tests still pass if we replace ‘or’ with ‘and’

<table>
<thead>
<tr>
<th>In1</th>
<th>In2</th>
<th>In3</th>
<th>In1 &amp; in2</th>
<th>(in1 &amp; in2) and in3</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
MC/DC Effectiveness

Code structure has large effect!

Choice of oracle has large effect!

DWM_1

Vertmax_Batch

DWM_2
Another Way to Look at MC/DC

- Masking MC/DC can be expressed:

\[(D(t_i) \neq D[true/c_n](t_i)) \wedge (D(t_j) \neq D[false/c_n](t_j))\]

Where \( P[v/e_n] \) means, For program \( P \), the computed value for the \( nth \) instance of expression \( e \) is replaced by value \( v \)

- Describes whether a condition is observable in a decision (i.e., not masked)

- **Problem**: we can rewrite programs to make decisions large or small (and MC/DC easy or hard to satisfy!)
Idea: lift observability from decisions to programs

- Explicitly account for oracle
- Strength should be unaffected by simple program transformations (e.g., inlining)

\[
(\forall c_n \in \text{Cond}(P). \ (\exists t \in T. \ (P(t) \neq P[true/c_n](t))) \ \land \ (\exists t \in T. \ (P(t) \neq P[false/c_n](t))))
\]

where \(\text{Cond}(P)\) is the set of all conditions in program \(P\)
Tagged Semantics

• Semantic definition is unwieldy for measurement and test generation
  – Requires separate program variant for every condition
  – Run variant in parallel with original program

• Approximate by tagging semantics
  – Assign each condition a tag
  – Track these tags through program execution (both the condition’s tag and value)
  – If a tag reaches the output, the obligation is satisfied
An Example Program (in Simulink)

Does the value of input2 affect the output?  No
Does the value of input2 affect the output?  No
Does the `true` value of input2 affect the output?

Yes. If input4 is `true`, then var1 is not masked out by the AND gate, so input2 propagates.

We can define the tagging semantics by instrumenting the original program; we then use this instrumented program for both test measurement and test generation.
Experiments and Evaluation

- For each of 4 industrial avionics systems and 1 toy system:
- Create inlined and non-inlined implementations
- Test suite generation
  - Counterexample-based approach guarantees maximum possible coverage (using Kind)
  - 10 test suites each for OMC/DC and MC/DC
- Mutant generation
  - 250 mutants for each case example
  - Removed functionally equivalent mutants
    - Finite systems, decidable and fast
- Output-only and maximum oracles
  - Output-only oracle compares values only for output variables
  - Maximum oracle compares values for all internal variables and outputs
# Achievable Obligations

<table>
<thead>
<tr>
<th></th>
<th>Structure</th>
<th>OMC/DC</th>
<th>MC/DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWM1</td>
<td>Non-Inlined</td>
<td>99.9%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Inlined</td>
<td>68.7%</td>
<td>98.1%</td>
</tr>
<tr>
<td>DWM2</td>
<td>Non-Inlined</td>
<td>89.8%</td>
<td>95.3%</td>
</tr>
<tr>
<td></td>
<td>Inlined</td>
<td>57.5%</td>
<td>64.8%</td>
</tr>
<tr>
<td>Latctl</td>
<td>Non-Inlined</td>
<td>93.4%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Inlined</td>
<td>92.7%</td>
<td>99.6%</td>
</tr>
<tr>
<td>Vertmax</td>
<td>Non-Inlined</td>
<td>98.2%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Inlined</td>
<td>96.4%</td>
<td>99.1%</td>
</tr>
<tr>
<td>Microwave</td>
<td>Non-Inlined</td>
<td>68.9%</td>
<td>98.9%</td>
</tr>
<tr>
<td></td>
<td>Inlined</td>
<td>72.2%</td>
<td>94.2%</td>
</tr>
</tbody>
</table>
Oracle Matters

![Bar chart showing comparison between Output Only and Maximum values for DWM_1, DWM_2, Latctl_Batch, and Vertmax_Batch]
More Oracle Variables is Better
Some Variables Are Better

![Bar Chart]

- Percentage of Internal Variables/Outputs
- Number of Faults Found
- Categories: 0-1, 1-2, 2-5, 5-10, 10-15, 15-20, 20-50, 50-100, 100-150, 150-200
- Legends: Latctl, DWM_2, Vertmax, FGS, DWM_1
Oracle Selection Process

Common Pattern for Structure-based, Random Tests:
Summary and Future Work

• Testing effectiveness is influenced by many factors
  – Interrelationship between Program, Specification, Test Set, and Oracle
• Potential benefits in examining other artifacts in software testing
  – Can we discover “good” combinations?
• Potential dangers in adopting too narrow a view of a software testing

• Much more work to be done!

• Observable MC/DC
  – Robust to program structure
  – Better fault finding than MC/DC
  – Explicitly accounts for oracle
• Oracle discovery
  – Find the best variables to monitor
• Future work
  – Discover “complete” coverage criteria
    • Match program, specification, tests, and oracle in “good” ways
  – Larger studies with C and Java code
  – Dismiss uncoverable code
Questions