Metamorphic Testing of Developer Tools

Michael Pradel
Software Lab – University of Stuttgart
Joint work with Daniel Lehmann, Matteo Paltenghi, and Sandro Tolksdorf
Developer Tools
Developer Tools

- Debuggers
- Compilers
- Static analyzers
- Runtime engines
- Bug trackers
- Version control systems
Metamorphic Testing of Developer Tools

Michael Pradel
Software Lab – University of Stuttgart
Joint work with Daniel Lehmann, Matteo Paltenghi, and Sandro Tolksdorf
Why Testing of Developer Tools?
Why Testing of Developer Tools?

Foundation of successful software engineering

Buggy tools cause

- Misbehaving programs
- Confused developers
Metamorphic Testing of Developer Tools

Michael Pradel
Software Lab – University of Stuttgart
Joint work with Daniel Lehmann, Matteo Paltenghi, and Sandro Tolksdorf
Metamorphic Testing
Metamorphic Testing
Metamorphic Testing

Initial Test Run

Follow-up Test Run

Input\textsubscript{A} \rightarrow \text{Transform} \rightarrow \text{Input\textsubscript{B}}

\text{Output\textsubscript{A}} \rightarrow \text{Compare} \rightarrow \text{Output\textsubscript{B}}

R\textsubscript{I}

R\textsubscript{O}?
Metamorphic Testing

\[
x \rightarrow x + 2\pi
\]

\[
\sin(\ldots)
\]
Why use Metamorphic Testing?
Why use Metamorphic Testing?

General answer: Addresses oracle problem

Specific to developer tools:
- Inputs (e.g., programs) have well-defined semantics
- Can design metamorphic transformations on top
This Talk

- Interactive Metamorphic Testing of Debuggers [ISSTA’19]
- MorphQ: Metamorphic Testing of the Qiskit Quantum Computing Platform [ICSE’23]
- Lessons learned and open challenges [ICSE’24, ’25, etc. ?]
Motivating Example

Debugger pauses at a breakpoint in dead code:

Firefox bug # 1370648
Testing of Debuggers

- **Inputs**
  - Program-to-debug
  - Sequence of actions (e.g., set breakpoint)

- **Output**
  - Debugging trace (e.g., pausing, program state)
Goal & Challenges

Goal:
Automatically test interactive debuggers

Challenges:
- Complex input
- No well-defined oracle
- Interactive nature of debuggers
Goal & Challenges

Goal:
Automatically test interactive debuggers

Challenges:
- Complex input
- No well-defined oracle
- Interactive nature of debuggers

Debugging actions depend on program
Goal & Challenges

Goal:
Automatically test interactive debuggers

Challenges:
- Complex input
- No well-defined oracle
- Interactive nature of debuggers
- Pause at a breakpoint on a comment line?
Goal & Challenges

Goal:

Automatically test interactive debuggers

Challenges:

- Complex input
- No well-defined oracle
- Interactive nature of debuggers

Expected semantics of debugging actions become clear only when program executes
Overview
Action Transformations

- Add breakpoint and continue
- Replace continue by step
- Breakpoint sliding
Action Transformations

- Add breakpoint and continue
- Replace continue by step
- Breakpoint sliding

Adding a breakpoint at line $l$ should cause only additional pauses at $l$
Action Transformations

- Add breakpoint and continue
- Replace continue by step
- Breakpoint sliding

Adding a breakpoint at line $l$ should cause only additional pauses at $l$
Action Transformations

- Add breakpoint and continue
- Replace continue by step
- Breakpoint sliding

Adding a breakpoint at line $l$ should cause only additional pauses at $l$
Action Transformations

- Add breakpoint and continue
- Replace continue by step
- Breakpoint sliding

Setting breakpoint at $l$, which slides to $l'$, should be equal to directly setting it at $l'$
Program Transformations

- Insert or remove dead code
- Add parameter
- Add no-op
- Replace literal with expression
Program Transformations

- Insert or remove dead code
- Add parameter
- Add no-op
- Replace literal with expression

Should have no influence except changed line numbers

```c
1 if (false) {
2     variable = value;
3 }
```
Program Transformations

- Insert or remove dead code
- Add parameter
- Add no-op
- Replace literal with expression

Should show additional variable in program state

```javascript
function foo(p1,p2) {
    // p1, p2 are in scope
}
foo();
```
Program Transformations

- Insert or remove dead code
- Add parameter
- Add no-op
- Replace literal with expression

Should show additional variable in program state

```plaintext
1:    function foo(p1,p2) {
2:          // p1, p2 are
3:          // in scope
4:    }
5:    foo();

1:    function foo(p1,p2,fresh) {
2:          // now also expect
3:          // fresh == undefined
4:    }
5:    foo();
```
Interactive Metamorphic Testing

**Traditional metamorphic testing:**
- Apply transformations *without* executing the program

**Here:**
- Need to execute to know which transformations are applicable
Interactive Metamorphic Testing

**Traditional metamorphic testing:**

- Apply transformations **without** executing the program

**Here:**

- Need to execute to know which transformations are applicable

E.g., knowing what line a breakpoint slides to

```javascript
1  // requested breakpoint at this comment line...
2 var x = 0; // ...is moved to next statement
```
Evaluation

- **Target:** JavaScript debugger of Chromium

- **47k JavaScript programs**
  - Initial debugging actions: Randomly created by DBDB [FSE’18]
  - One follow-up input for each program
## Effectiveness

<table>
<thead>
<tr>
<th>Issue ID</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>862978</td>
<td>Cannot set breakpoint</td>
<td>Assigned</td>
</tr>
<tr>
<td>889481</td>
<td>Debugger does not pause</td>
<td>Assigned</td>
</tr>
<tr>
<td>892622</td>
<td>Debugger does not pause</td>
<td>Assigned, release-blocking</td>
</tr>
<tr>
<td>892653</td>
<td>Pauses at location without breakpoint</td>
<td>Assigned</td>
</tr>
<tr>
<td>901811</td>
<td>Missing variable in scope</td>
<td>Assigned</td>
</tr>
<tr>
<td>901814</td>
<td>Step-in does not enter function</td>
<td>Assigned</td>
</tr>
<tr>
<td>901816</td>
<td>Missing variable in scope</td>
<td>Assigned</td>
</tr>
<tr>
<td>901819</td>
<td>Debugger does not pause</td>
<td>Assigned</td>
</tr>
<tr>
<td>908054</td>
<td>Debugging changes program behavior</td>
<td>Won’t fix</td>
</tr>
</tbody>
</table>
Examples

Fails to stop at breakpoint:

```
1  // Original input:
2  var a = 5;  // (i) pauses --> continue
3  var slideOverMe;
4  var C = class{};  // (ii) pauses --> continue
5  var b = 42;  // (iii) pauses --> continue

1  // Transformed input:
2  var a = 5;  // (i) pauses --> continue
3  var slideOverMe;
4  var C = class{};  // (no pausing)
5  var b = 42;  // (ii) pauses
```

Chromium bug #889481
Examples

Incorrect program state:

```javascript
// Original input:
function * t({x: y}) { // pauses, y is in scope
    var a = function() {
    }
}
t({x: 1});

// Transformed input:
function * t({x: y}) { // pauses, y is missing
    var a = function() {
        if (false) { // dead code
            y = 5;
        }
    }
}
t({x: 1});
```

Chromium bug #901811
This Talk

- Interactive Metamorphic Testing of Debuggers [ISSTA’19]
- MorphQ: Metamorphic Testing of the Qiskit Quantum Computing Platform [ICSE’23]
- Lessons learned and open challenges [ICSE’24, ’25, etc. ?]
Quantum Computing Stack

- Algorithms
- Platforms (e.g., IBM’s Qiskit and Google’s Circ)
- Quantum computers
Quantum Computing Stack

- Algorithms
- Platforms (e.g., IBM’s Qiskit and Google’s Circ)
- Quantum computers

Our goal: Test this
Why Relevant?

- Quantum computing: Emerging field with huge investments
- Reliable platforms are crucial
- Novel, quantum-specific bug patterns

[OOPSLA’22]
Background: Quantum Software

Quantum algorithm (in Qiskit):

```python
1  # Create circuit
2  circ = QuantumCircuit(2)
3  circ.h(0)    # Hadamard gate
4  circ.cx(0, 1)    # Controlled not gate
5  circ.measure_all()
6  # Transpile for simulator
7  simulator = Aer.get_backend(‘aer_simulator’)  
8  circ = transpile(circ, simulator)
9  # Run and get counts
10 result = simulator.run(circ, shots=1024).result()
11 counts = result.get_counts(circ)
12 # output: {‘00’: 530, ’11’: 494}
```
Background: Quantum Software

```python
# Create circuit
circ = QuantumCircuit(2)
circ.h(0)  # Hadamard gate
circ.cx(0, 1)  # Controlled not gate
circ.measure_all()
# Transpile for simulator
simulator = Aer.get_backend('aer_simulator')
circ = transpile(circ, simulator)
# Run and get counts
result = simulator.run(circ, shots=1024).result()
counts = result.get_counts(circ)
# output: {'00': 530, '11': 494}
```

Visual representation
Background: Quantum Software

```python
# Create circuit
circ = QuantumCircuit(2)
circ.h(0)  # Hadamard gate
circ.cx(0, 1)  # Controlled not gate
circ.measure_all()
# Transpile for simulator
simulator = Aer.get_backend('aer_simulator')
circ = transpile(circ, simulator)
# Run and get counts
result = simulator.run(circ, shots=1024).result()
counts = result.get_counts(circ)
# output: {'00': 530, '11': 494}
```

Output: Probability distribution

![Probability distribution graph](chart.png)
Goal & Challenges

Goal: **Automatically test quantum computing platforms**

Challenges:
- Relatively few quantum programs
- No well-defined oracle
- Unreliable and difficult-to-access hardware
Goal & Challenges

Goal: Automatically test quantum computing platforms

Challenges:
- Relatively few quantum programs
- No well-defined oracle
- Unreliable and difficult-to-access hardware

New and emerging domain
Goal & Challenges

Goal: Automatically test quantum computing platforms

Challenges:

- Relatively few quantum programs
- No well-defined oracle
- Unreliable and difficult-to-access hardware
- Low-level operations with sometimes counterintuitive semantics
Goal & Challenges

Goal: Automatically test quantum computing platforms

Challenges:
- Relatively few quantum programs
- No well-defined oracle
- Unreliable and difficult-to-access hardware
- Quantum noise induced by stray electromagnetic fields or material defects
Overview of MorphQ

1. Program Generation
   Source
   
   \( q_0 \rightarrow A \rightarrow B \rightarrow D \rightarrow C \rightarrow E \)
   
   Exec. Settings
   
   \( \text{in}_s \)

2. Metamorphic Transformation
   Follow-Up
   
   \( q_0 \rightarrow A \rightarrow B \rightarrow D \rightarrow C \rightarrow X \rightarrow X \rightarrow E \)
   
   Exec. Settings
   
   \( \text{in}_f \)

3. Execution & Check Behavior
   \( \text{out}_s \)
   
   no crash → crash
   
   same? \( (R_{\text{OUTPUT}}) \)
   
   \( \text{out}_f \)
   
   no crash → crash
Generating Programs

- Template- and grammar-based, randomized algorithm
- Guarantee: Produces non-crashing program
Generating Programs

```python
# Section: Prologue
<ALL_IMPORTS>

# Section: Circuit
qr = QuantumRegister(<N_QUBITS>, name='qr')
cr = ClassicalRegister(<N_QUBITS>, name='cr')
qc = QuantumCircuit(qr, cr, name='qc')

# Section: Measurement
qc.measure(qr, cr)

# Section: Transpilation/compilation
qc = transpile(qc,
    basis_gates=<TARGET_GATE_SET>,
    optimization_level=<OPT_LEVEL>,
    coupling_map=<COUPLING_MAP>)

# Section: Execution
simulator = Aer.get_backend(<BACKEND_NAME>)
counts = execute(qc, backend=simulator,
    shots=<N_SHOTS>).result().get_counts(qc)
```
Metamorphic Transformations

1) **Circuit transformations**

- Change qubit order
- Inject null-effect operation
- Add quantum register
- Inject parameters
- Partitioned execution
Metamorphic Transformations

1) Circuit transformations

- Change qubit order
- Inject null-effect operation
- Add quantum register
- Inject parameters
- Partitioned execution
Metamorphic Transformations

1) Circuit transformations
   ■ Change qubit order
   ■ Inject null-effect operation
   ■ Add quantum register
   ■ Inject parameters
   ■ Partitioned execution

Change qubit order
Metamorphic Transformations

1) Circuit transformations
- Change qubit order
- Inject null-effect operation
- Add quantum register
- Inject parameters
- Partitioned execution

Partitioned execution
2) **Representation transformations**

- Roundtrip conversion via QASM

3) **Execution transformations**

- Change of coupling map
- Change of gate size
- Change of optimization level
- Change of backend
Metamorphic Transformations

2) **Representation transformations**
   - Roundtrip conversion via QASM

3) **Execution transformations**
   - Change of coupling map
   - Change of gate size
   - Change of optimization level
   - Change of backend
Metamorphic Transformations

IBM Stuttgart, Germany

IBM Melbourne, Australia
Comparing Behavior

- Expected output relationship:
  Equivalence modulo changes in distribution
  - E.g., changing qubit order will change measured bitstrings

- Two oracles
  - Crash vs. non-crash
  - Distribution differences
    (via Kolmogorov-Smirnov test)
Evaluation

- **Target:** IBM’s Qiskit quantum computing platform
- **48-hour run**
  - 8,360 generated programs
  - Same number of follow-up programs
    - 23.2% of follow-up programs crash
    - 0.7% of non-crashing have distribution differences
# Effectiveness

## Bugs filed after

- Automated clustering of warnings
- Delta-debugging to reduce bug-triggering program

<table>
<thead>
<tr>
<th>ID</th>
<th>Report</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#7694</td>
<td>confirmed</td>
</tr>
<tr>
<td>2</td>
<td>#7700</td>
<td>confirmed</td>
</tr>
<tr>
<td>3</td>
<td>#7750</td>
<td>confirmed</td>
</tr>
<tr>
<td>4</td>
<td>#7749</td>
<td>confirmed</td>
</tr>
<tr>
<td>5</td>
<td>#7641</td>
<td>confirmed</td>
</tr>
<tr>
<td>6</td>
<td>#7326</td>
<td>confirmed</td>
</tr>
<tr>
<td>7</td>
<td>#7756</td>
<td>confirmed</td>
</tr>
<tr>
<td>8</td>
<td>#7748</td>
<td>fixed</td>
</tr>
<tr>
<td>9</td>
<td>#8224</td>
<td>fixed</td>
</tr>
<tr>
<td>10</td>
<td>#7769</td>
<td>reported</td>
</tr>
<tr>
<td>11</td>
<td>#7771</td>
<td>reported</td>
</tr>
<tr>
<td>12</td>
<td>#7772</td>
<td>reported</td>
</tr>
<tr>
<td>13</td>
<td>#7773</td>
<td>reported</td>
</tr>
</tbody>
</table>
Example

Detected by **changing optimization level** and **injecting null-effect operation**

```python
qr = QuantumRegister(11, name='qr')
cr = ClassicalRegister(11, name='cr')
qc = QuantumCircuit(qr, cr, name='qc')
subcircuit = QuantumCircuit(qr, cr, name='subcirc')
subcircuit.x(3)
qc.append(subcircuit, qargs=qr, cargs=cr)
qc.x(3)
qc = transpile(qc, optimization_level=2)
# ValueError: too many subscripts in einsum
```
This Talk

- Interactive Metamorphic Testing of Debuggers [ISSTA’19]
- MorphQ: Metamorphic Testing of the Qiskit Quantum Computing Platform [ICSE’23]
- Lessons learned and open challenges [ICSE’24, ’25, etc. ?]
Lessons Learned

Key ingredient:
Metamorphic transformations

- Inherently domain-specific
- Relies on some “model” of the program-under-test
  - E.g., debuggers transform programs and debugging actions into a debugging trace
Lessons Learned

Key ingredient:
Metamorphic transformations

- Inherently domain-specific
- Relies on some “model” of the program-under-test
  - E.g., debuggers transform programs and debugging actions into a debugging trace

The better the transformations, the more bugs you find
Lessons Learned (2)

Vaguely specified programs:
Difficult to define precise metamorphic oracles

- Negative example:
  Testing *git* version control system
  - Many underspecified corner cases
  - Failed to effectively test it
Lessons Learned (2)

Vaguely specified programs:
Difficult to define precise metamorphic oracles

- Negative example:
  Testing *git* version control system
  - Many underspecified corner cases
  - Failed to effectively test it

Make sure to know (at least parts of) the program’s intended behavior
Lessons Learned (3)

Programs that operate on programs:
Excellent target for metamorphic testing

- Indended semantics are (relatively) clearly defined
- Can derive metamorphic relationships from PL semantics
Lessons Learned (3)

Programs that operate on programs:
Excellent target for metamorphic testing

- Indended semantics are (relatively) clearly defined
- Can derive metamorphic relationships from PL semantics

More developer tools are waiting to be tested
Open Challenges

- **False positives**
  - Debugger testing: 29/59 warnings
  - MorphQ: All warnings due to distribution differences

- **Automate creation of metamorphic relationships**
  - Initial evidence that ML-based prediction may help *

*Code Generation Tools (Almost) for Free? A Study of Few-Shot, Pre-Trained Language Models on Code (Bareiß et al., 2022)*
Summary

- Interactive Metamorphic Testing of Debuggers [ISSTA’19]
- MorphQ: Metamorphic Testing of the Qiskit Quantum Computing Platform [ICSE’23]
- Lessons learned and open challenges [ICSE’24, ’25, etc. ?]

Thanks!