An Empirical Study on Failed Error Propagation in Java Programs with Real Faults

Gunel Jahangirova, David Clark, Mark Harman and Paolo Tonella

30th of January
57th CREST Open Workshop
UCL, London, UK
Failed Error Propagation

public int test(int x) {
    //pp0
    statement
    //pp1
    faulty statement
    //pp2
    statement
    ..
    ..
    ..
    //pp_ret
    return x;
}
Experimental Procedure

**Subjects**
6 projects from Defects4J benchmark:
395 real bugs, 528 methods

**Trace Alignment**
Tree Edit Distance algorithm applied to ASTs of fixed and buggy methods

**Measure FEP**

**Test Case Generation**
1000 executions that cover faulty statement
## Benchmark: Defects4J

<table>
<thead>
<tr>
<th>Project</th>
<th>Number of Bugs</th>
<th>Bugs with fix in M/C</th>
<th>Number of M/C</th>
<th>M/C &gt; 1 LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFreeChart</td>
<td>26</td>
<td>26</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>Closure Compiler</td>
<td>133</td>
<td>131</td>
<td>172</td>
<td>172</td>
</tr>
<tr>
<td>Commons Lang</td>
<td>65</td>
<td>61</td>
<td>88</td>
<td>82</td>
</tr>
<tr>
<td>Commons Math</td>
<td>106</td>
<td>104</td>
<td>146</td>
<td>135</td>
</tr>
<tr>
<td>Mockito</td>
<td>38</td>
<td>37</td>
<td>76</td>
<td>50</td>
</tr>
<tr>
<td>Joda Time</td>
<td>27</td>
<td>26</td>
<td>51</td>
<td>50</td>
</tr>
</tbody>
</table>

**Total:** 384 bugs  
**Total M/C:** 572
Test Case Generation

1. int test(int x) {
2.     x = 3 * x;
3.     if (x > 0) {
4.         x = x % 4;
5.     } else {
6.         x = x + 1;
7.     }
8.     return x;
9. }

Line Coverage Criteria:

- line_list
- goals_multiply

Budget: 10000 seconds
public Complex reciprocal() {
    if (isNaN) {
        return NaN;
    }
    if (real == 0.0 && imaginary == 0.0) {
        return NaN;
    }
    if (isInfinite) {
        return ZERO;
    }
    if (FastMath.abs(real) < FastMath.abs(imaginary)) {
        double q = real / imaginary;
        double scale = 1. / (real * q + imaginary);
        return createComplex(scale * q, -scale);
    } else {
        double q = imaginary / real;
        double scale = 1. / (imaginary * q + real);
        return createComplex(scale, -scale * q);
    }
}
public Line revert() {
    final Line reverted = new Line(this);
    reverted.direction = reverted.direction.negate();
    return reverted;
}

public Dfp multiply(final int x) {
    return multiplyFast(x);
}

public boolean isFeasible(final double[] x) {
    if (boundaries == null) {
        return true;
    }
    for (int i = 0; i < x.length; i++) {
        if (x[i] < 0) {
            return false;
        } else if (x[i] > 1.0) {
            return false;
        }
    }
    return true;
}

final double[] bLoEnc = encode(boundaries[0]);
final double[] bHiEnc = encode(boundaries[1]);
for (int i = 0; i < x.length; i++) {
    if (x[i] < bLoEnc[i]) {
        return false;
    } else if (x[i] > bHiEnc[i]) {
        return false;
    }
}
return true;
public int test(int x) {
    int y = x + 1;
    y = y % 4;
    return y;
}

public int test(int x) {
    int y = x + 1;
    y = y % 3;
    return y;
}

KEEP int y = x + 1;
CHANGE y = y % 4; to y = y % 3;
KEEP return y;
public int test(int x) {
    int y = x + 1;
    y = y % 4;
    return y;
}

public int test(int x) {
    int y = x + 1;
    y = y * 3;
    y = y % 4;
    return y;
}
```
public int test(int x) {
    int y = x + 1;
    y = y % 4;
    return y;
}
```
Algorithm 1: Program point instrumentation

Procedure visit($n, i$)

Input:
$n$: AST node to be visited
$i$: instrumentation index

begin

if $n$ is labeled as $KEEP$ or $CHANGE \land type(n)$ is not ($RETURN$ or $THROW$) then

while $\text{next}(n)$ is labeled as $DELETE$ or $INSERT$ do

$n := \text{next}(n)$

$i := i + 1$

instrumentAfter($n, pp_i$)

else

visit($\text{next}(n), i$)

if $\text{type}(n)$ is not ($FOR$ or $WHILE$) then

for $m \in \text{children}(n)$ do

visit($m, i$)

end
Program Point Instrumentation

```java
1 public int test(int x) {
2     //pp0
3     int y = x + 1;
4     //pp1
5     y = y % 4;
6     //pp2
7     return y;
8 }
```

```
1 public int test(int x) {
2     //pp0
3     int y = x + 1;
4     //pp1
5     y = y % 3;
6     //pp2
7     return y;
8 }
```
Program Point Instrumentation

```
1 public int test(int x) {
2     //pp0
3     int y = x + 1;
4     //pp1
5     y = y % 4;
6     //pp2
7     return y;
8 }
```

```
1 public int test(int x) {
2     //pp0
3     int y = x + 1;
4      y = y * 3;
5     //pp1
6     y = y % 4;
7     //pp2
8     return y;
9 }
```

KEEP int y = x + 1;
INSERT y = y * 3;
KEEP y = y % 4;
KEEP return y;
Program Point Instrumentation

```java
public int test(int x) {
    //pp0
    int y = x + 1;
    y = y % 4;
    //pp1
    return y;
}
```

```java
public int test(int x) {
    //pp0
    int y = x + 1;
    //pp1
    return y;
}
```

KEEP int y = x + 1;
DELETE y = y % 4;
KEEP return y;

14
Measuring FEP

Input:

$\text{type} = \langle \text{sys} \mid \text{unit} \rangle$: type of analysis, system-level or unit-level

$\text{out, out}'$: output of the system, used only for system-level analysis

$\text{ext, ext}'$: externally observable state after buggy/fixed methods have been executed

$\text{pp} = \langle pp_0, \ldots, pp_n \rangle$: program points executed in fixed method

$\text{pp}' = \langle pp'_0, \ldots, pp'_k \rangle$: program points executed in buggy method

$s, s'$: state by program point in buggy/fixed methods

Result:

$\text{fepType}: \langle \text{sysFEP} \mid \text{intFEP} \mid \text{extFEP} \mid \text{noFEP} \rangle$

```plaintext
begin
  if \text{type} = \text{unit} \&\& \text{ext} \neq \text{ext}' then
    \text{return noFEP}
  if \text{type} = \text{sys} then
    if \text{out} \neq \text{out}' then
      \text{return noFEP}
    if \text{ext} \neq \text{ext}' then
      \text{return closure(sysFEP, type)}
    if s[pp_n] \neq s'[pp'_k] then
      \text{return closure(extFEP, type)}
    if pp \neq pp' then
      \text{return closure(intFEP, type)}
    for i \in [1 : n - 1] do
      if s[pp_i] \neq s'[pp'_i] then
        \text{return closure(intFEP, type)}
  \text{return noFEP}
```
RQ1: What is the prevalence of unit-level failed error propagation with real faults?
RQ1: TS generation

- Joda Time: 40 methods with TC generated, 10 methods without
- Mockito: 30 methods with TC generated, 20 methods without
- Commons Math: 101 methods with TC generated, 34 methods without
- Commons Lang: 54 methods with TC generated, 28 methods without
- Closure: 107 methods with TC generated, 65 methods without
- JFreeChart: 21 methods with TC generated, 18 methods without

Legend:
- Light blue: Methods with TC generated
- Dark blue: Methods without TC generated
RQ1: Executions

- **JFreeChart**
  - No FEP (fault propagates to output): 56%
  - No FEP (fault does not affect internal state): 44%

- **Closure Compiler**
  - No FEP (fault propagates to output): 45%
  - No FEP (fault does not affect internal state): 55%

- **Commons Lang**
  - No FEP (fault propagates to output): 51%
  - No FEP (fault does not affect internal state): 49%

- **Commons Math**
  - No FEP (fault propagates to output): 32%
  - No FEP (fault does not affect internal state): 68%

- **Mockito**
  - No FEP (fault propagates to output): 66%
  - No FEP (fault does not affect internal state): 34%

- **Joda Time**
  - No FEP (fault propagates to output): 63%
  - No FEP (fault does not affect internal state): 37%

No FEP (fault does not affect internal state)
RQ1: Executions

- No FEP (fault propagates to output)
- No FEP (fault does not affect internal state)

- 163.593
- 190.062
RQ2: Does the prevalence of unit-level failed error propagation change if real faults are replaced by mutants?
RQ2: Method Groups

- 2-25 LOC: 278
- 26-50 LOC: 87
- 51-100 LOC: 109
- 101-200 LOC: 44
- > 200 LOC: 10
RQ2: Mutations

- **Strongly Killed**
- **Not Strongly Killed**

- **Joda Time**: 212
- **Mockito**: 28
- **Math**: 765
- **Lang**: 360
- **Closure**: 468
- **JFreeChart**: 18
RQ2: Executions

- JFreeChart: 76% No FEP (fault does not affect internal state), 24% No FEP (fault propagates to output)
- Mockito: 55% No FEP (fault does not affect internal state), 45% No FEP (fault propagates to output)
- Joda Time: 42% No FEP (fault does not affect internal state), 58% No FEP (fault propagates to output)
RQ2: Executions

- **Commons Math**
  - No FEP (fault propagates to output): 46.3%
  - No FEP (fault does not affect internal state): 47.6%
  - External FEP: 6.1%
  - 2.4% Internal FEP

- **Closure Compiler**
  - No FEP (fault propagates to output): 42.2%
  - No FEP (fault does not affect internal state): 56.3%
  - External FEP: 1.5%
  - 0.7% Internal FEP

- **Commons Lang**
  - No FEP (fault propagates to output): 45.6%
  - No FEP (fault does not affect internal state): 51.4%
  - External FEP: 2.9%
  - 1.1% Internal FEP
RQ2: Executions

No FEP (fault propagates to output)  No FEP (fault does not affect internal state)  External FEP

- 45.7%
- 50.6%
- 1.6%
RQ3: Does the prevalence of failed error propagation with real faults change if it is measured at the system level instead of unit level?
RQ3: Executions

- No FEP (fault propagates to output)
- No FEP (fault does not affect internal state)
- System FEP

- 80.3% No FEP (fault does not affect internal state)
- 8.3% System FEP
- 11.4% No FEP (fault propagates to output)
- 1.5% Internal & External FEP
Qualitative Analysis

Manual analysis on 384 bugs from Defects4J and generated mutants

Bug fix affects output directly

State change resulting from the fix is such that it always propagates to output
public Complex divide(double divisor) {
    if (isNaN || Double.isNaN(divisor)) {
        return NaN;
    }
    if (divisor == 0d) {
        return NaN;
        //return isZero ? NaN : INF;
    }
    if (Double.isInfinite(divisor)) {
        return !isInfinite() ? ZERO : NaN;
    }
    return createComplex(real / divisor,
                          imaginary / divisor);
}
Bug fix affects output directly

```java
1  public static LocalDate fromDateFields(Date date) {
2      if (date == null) {
3          throw new IllegalArgumentException
4              ("The date must not be null");
5      }
6
7      //if (date.getTime() < 0) {
8          // GregorianCalendar cal = new GregorianCalendar();
9          // cal.setTime(date);
10         // return fromCalendarFields(cal);
11      // }
12
13      return new LocalDate(
14          date.getYear() + 1900,
15          date.getMonth() + 1,
16          date.getDate());
17  }
```
Fix propagates to output

```java
public double getChiSquare() {
    double chiSquare = 0;
    for (int i = 0; i < rows; ++i) {
        final double residual = residuals[i];
        chiSquare += residual * residual *
                    residualsWeights[i];
        //chiSquare += residual * residual /
        //residualsWeights[i];
    }
    return chiSquare;
}
```
Reasons for the absence of FEP

- Change in Return: 8.74%
- If addition: 11.71%
- Bug propagates to output: 19.93%
- Other: 59.62%

Change in Return
If addition
Bug propagates to output
Other
FEP in mutants

```java
protected double getInitialDomain(double p) {
    double ret = 0.0;
    //mut0: double ret = 1.0;
    //pp1
    double d = getDenominatorDegreesOfFreedom();
    if (d > 2.0) {
        ret = d / (d - 2.0);
    }
    //pp_ret
    return ret;
}
```
Implications

- Internal Oracles
- Post-conditions
- Subsystem Testing
- Mutants vs. Real Faults
Failed Error Propagation

System

Unit

public int test(int x)
   //pp0 statement
   //pp1 faulty statement
   //pp2 statement
   ...
   ...
   //pp_ret return x;
   ...

System FEP

External FEP

Internal FEP

RQ2: Executions

- Externally Detectable
- Externally Not Detectable
- External FEP (Internal)
- External FEP (Return points)

- 50.6%
- 45.7%
- 3.1%
- 1.6%

Experimental Procedure

Subjects
- 6 projects from Defects4J benchmark
- 384 real bugs, 528 methods

Trace Alignment
- Tree Edit Distance algorithm applied to ASTs of fixed and buggy methods

Measure Internal/External FEP

Test Case Generation
- 1000 executions that cover faulty statement

Reasons for the absence of FEP

- Change in Return: 8.74%
- If addition: 11.71%
- Bug propagates to output: 19.93%
- Other: 59.62%