Field study and classification of faults in Eiffel

Raluka Borca-Mureșan
ETHZ
June, 14, 2006
Project description

Field study and classification of faults in Eiffel

ETHZ - June 2006, Raluca Borca-Mureșan - p. 2
Problem:

- **Testing** - an important step in the software development process
Problem:

- Testing - an important step in the software development process

- BUT testing is:
  - time consuming
  - tiresome
  - boring
  - ...
Problem:

- Testing - an important step in the software development process

- BUT testing is:
  - time consuming
  - tiresome
  - boring
  - ...

- use automatic testing tools (AutoTest) that:
  - provide a list of test cases that generate bugs
  - NO debugging and interpretation of results
Problem:

- Testing - an important step in the software development process

- BUT testing is:
  - time consuming
  - tiresome
  - boring
  - …

- use automatic testing tools (AutoTest) that:
  - provide a list of test cases that generate bugs
  - NO debugging and interpretation of results

- HOW TO BUILD A CLASSIFICATION SCHEME?
Solution

- AutoTest
- KIKS
- Software
- Bugs

Field of study and classification of faults in Eiffel

ETHZ - June 2006, Raluca Borca-Mureșan - p. 5
Solution

Fault classification:

Category 1:
  Subcategory 1.1
  Subcategory 1.2
  ...

Category 2:
  Subcategory 2.1
  Subcategory 2.2
  ...
  ...

Category n:
  Subcategory n.1
  Subcategory n.2
  ...

Field study and classification of faults in Eiffel ETHZ - June 2006, Raluca Borca-Mureșan - p. 6
Solution

**Fault classification:**

- **Category 1:**
  - Subcategory 1.1
  - Subcategory 1.2
  - ...

- **Category 2:**
  - Subcategory 2.1
  - Subcategory 2.2
  - ...

- **Category n:**
  - Subcategory n.1
  - Subcategory n.2
  - ...

*Analysis and Interpretation*

*Debugging*

*Large field case study*

**FINAL PRODUCT**

**Field study and classification of faults in Eiffel**
AutoTest was run on the following libraries and applications:

- EiffelBase
- Gobo
- PerfectDeveloper - mathematical library
- DrC
- EWG
Classification scheme

- Bug Types
- Implementation/Specification bug
- Implementation/Specification bug
- Implementation/Specification bug
- Implementation/Specification bug
- Specification bug-example
- Implementation bugs
- Implementation bug-example
- Dunno bug example
- Supplier-induced (SI) bug
- Inheritance contract bug
- Invariant
Bug Types

Field study and classification of faults in Eiffel

**Project description**

**Classification scheme**

- Bug Types
- Implementation/Specification bug
- Implementation/Specification bug
- Implementation/Specification bug
- Specification bug-example
- Implementation bugs
- Implementation bug-example
- Dunno bug example
- Supplier-induced (SI) bug
- Inheritance contract bug
- Invariant

**AutoTest**

**KIKS**

**Software**

**Bugs**

**FINAL PRODUCT**

Fault classification:

**Category 1:**
- Implementation
- Specification
- Dunno

**Category 2:**
- Faulty supplier
- Inheritance
- Wrong export status
- External fault
- Void target
- Other

*Analysis and Interpretation
*Debugging
*Large field case study

Field study and classification of faults in Eiffel

ETHZ - June 2006, Raluca Borca-Mureșan - p. 10
Implementation/ Specification bug

- Bug Types
  - Implementation/Specification bug
  - Implementation bug-example
  - Dunno bug example
  - Supplier-induced (SI) bug
  - Inheritance contract bug
  - Invariant

Field study and classification of faults in Eiffel

ETHZ - June 2006, Raluca Borca-Mureşan - p. 11
Definitions:

- A specification bug appears because of the discrepancy between the intended specification and the real specification.
Definitions:

- A specification bug appears because of the discrepancy between the intended specification and the real specification.

- An implementation bug appears because the implementation of the routine does not fulfill the real specification of the routine.
Definitions:

- A specification bug appears because of the discrepancy between the intended specification and the real specification.

- An implementation bug appears because the implementation of the routine does not fulfill the real specification of the routine.

- Dunno: prove that a bug can be interpreted as being a specification and an interpretation bug.
## Specification bug - example

### ARRAYED_TREE

<table>
<thead>
<tr>
<th>fill</th>
<th>(other: TREE [G]) is</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>−− Fill with as many</td>
</tr>
<tr>
<td></td>
<td>−− items of ‘other’</td>
</tr>
<tr>
<td></td>
<td>−− as possible. The</td>
</tr>
<tr>
<td></td>
<td>−− representations</td>
</tr>
<tr>
<td></td>
<td>−− of ‘other’ and</td>
</tr>
<tr>
<td></td>
<td>−− current node</td>
</tr>
<tr>
<td></td>
<td>−− need not be the</td>
</tr>
<tr>
<td></td>
<td>−− same. (from TREE)</td>
</tr>
</tbody>
</table>

### Test case

```eiffel
create {ARRAYED_TREE[STRING]} v_1.make (0, Void )
v_1. fill (Void)
```

In EiffelBase: 28 specification bugs.
Implementation bugs

An implementation bug is the result of an implementation of a routine not fulfilling the real specification of the routine.

In EiffelBase: 27 implementation bugs.
BASIC_ROUTINES

abs (n: INTEGER): INTEGER

is

−− Absolute value of ‘n’

do

if n < 0 then

Result := −n

else

Result := n

end

ensure

non_negative_result.

Result >= 0

end

Test case

create {BASIC_ROUTINES} v_15

v_16 := v_15.abs(−2147483648)
Dunno bug example

```eiffel
STRING

adapt (s: STRING): like Current is
    -- Object of a type conforming to the type of ‘s’, initialized with attributes from ‘s’
    do
        Result := new_string (0)
        Result.share (s)
    ensure
        adapt_not_void:
            Result /= Void
        shared_implementation:
            Result.shared_with (s)
    end

Test case

create {STRING} v_1.make_empty
v_3 := v_1.adapt (Void)
```

In EiffelBase: 2 dunno bugs
## Supplier-induced (SI) bug

A supplier-induced bug appears in the following cases: suppose routine \( r_1 \) calls routine \( r_2 \) (which is faulty). One can have:

### Specification SI bug

\[
\begin{align*}
\text{\textbf{r1 is}} \\
\text{\textbf{require}} \\
\text{r2 \hfill \textbf{!!! buggy}} \\
\text{\textbf{do}} \\
\text{...} \\
\text{\textbf{end}}
\end{align*}
\]

### Implementation SI bug

\[
\begin{align*}
\text{\textbf{r1 is}} \\
\text{\textbf{do}} \\
\text{r2 \hfill \textbf{!!! buggy}} \\
\text{\textbf{end}}
\end{align*}
\]

In EiffelBase: 9 specification SI bugs

In EiffelBase: 30 implementation SI bugs
Field study and classification of faults in Eiffel

Inheritance contract bug

Test case

```
create { LINKED_QUEUE[ANY] v_1.make }
v_1.append(Void)
```
### Invariant inheritance bug

#### class ML_SORT_SET

**default_create** is

--- Process instances of
--- classes with no creation
--- clause.
--- (Default: do nothing.)
--- (from ANY)
--- (export status {NONE})

**do**

**end**

#### The invariant

**set_size_not_negative:**

\[
\# \text{ Current} \geq 0
\]

**prefix "#"**

**prefix "#":** `INTEGER` is

--- how many elements are currently in the set
--- (i.e. cardinality)? (from ML_SET)

**do**

**Result := # implementation**

**ensure** --- from ML_MODEL

\[
is_{\text{empty}}_{\text{means\_count\_is\_zero}}: \text{is\_empty} \implies (\text{Result} = 0)
\]

**end**

#### Test case

**create** {ML_SORTED_SET[STRING]} \_v\_10
Other bugs in Category 2

- Wrong export status: creation procedures exported to ANY, but they should be exported to NONE!

- External fault

- Void target
Results
Results - EiffelBase

Category 1: Implementation, Specification, Dunno

- Impl: 46.38%
- Spec: 52.17%
- Dunno: 1.45%
Category 2: Supplier-induced, Inheritance, Export Status, External fault, Void target, other

- Impl.Si: 22.06%
- Spec.Si: 6.62%
- Inherit: 8.82%
- Export: 13.97%
- Extern: 4.41%
- Void target: 3.68%
- Other: 40.44%
Faulty Supplier:

- Impl: 45.65%
- Impl.Sl: 21.74%
- Spec.Sl: 6.52%
- Spec: 24.64%
- Dunno: 1.45%
Results - EiffelBase - Overlapping

Inheritance:

- Impl: 44.20%
- Imp Inherit: 1.45%
- Spec Inherit: 0.72%
- Spec: 7.97%
Results - EiffelBase - Overlapping

Export status:

- Impl: 1.45%
- Spec: 13.77%
- Export: 38.41%
- Overlapping: 16.38%
Results - EiffelBase - Overlapping

External fault:

- Impl: 43.48%
- Impl.External: 1.45%
- Spec: 50.72%
- Spec.External: 2.90%
- Dunno: 1.45%

Field study and classification of faults in Eiffel

ETHZ - June 2006, Raluca Borca-Mureșan - p. 29
Results - EiffelBase - Overlapping

Void target:

- Impl: 50.00%
- Impl.Void target: 44.93%
- Spec.Void target: 1.45%
- Spec: 1.45%
- Dunno: 2.17%
### Results

<table>
<thead>
<tr>
<th>Library/Application</th>
<th>Bugs</th>
<th>Buggy routines/total tested</th>
<th>fail tests/total tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>EiffelBase</td>
<td>127</td>
<td>6.40(127/1984)</td>
<td>3.81%(1513/39615)</td>
</tr>
<tr>
<td>base.kernel</td>
<td>16</td>
<td>4.64%(16/343)</td>
<td>1.37%(204/15140)</td>
</tr>
<tr>
<td>base.support</td>
<td>23</td>
<td>10.74%(23/214)</td>
<td>5.13%(166/3233)</td>
</tr>
<tr>
<td>base.structures</td>
<td>88</td>
<td>6.28(88/1400)</td>
<td>5.38%(1143/21242)</td>
</tr>
<tr>
<td>Gobo</td>
<td>26</td>
<td>4.44%(26/585)</td>
<td>3.66%(2928/79886)</td>
</tr>
<tr>
<td>Gobo -xml</td>
<td>17</td>
<td>3.85%(17/441)</td>
<td>3.71%(2912/78347)</td>
</tr>
<tr>
<td>Gobo - math</td>
<td>9</td>
<td>6.25(9/144)</td>
<td>1.03%(16/1539)</td>
</tr>
<tr>
<td>Perfect Devel - math</td>
<td>72</td>
<td>14.11%(72/510)</td>
<td>49.56%(12860/25946)</td>
</tr>
<tr>
<td>DoctorC</td>
<td>15</td>
<td>45.45%(15/33)</td>
<td>14.30%(1283/8972)</td>
</tr>
<tr>
<td>EWG</td>
<td>8</td>
<td>10.38%(8/77)</td>
<td>1.32%(43/3245)</td>
</tr>
</tbody>
</table>