Evaluating testing strategies for imaging software by means of Mutation Analysis

René Just & Franz Schweiggert

Ulm University, Germany

April 4, 2009
Outline

1. Motivation
2. Evaluating partial oracles
3. Example
4. Conclusion
Testing Strategy consists of two parts:
1. method to generate test inputs.
2. (partial) oracle.
Testing Process

Testing Strategy consists of two parts:

1. method to generate test inputs.
2. (partial) oracle.
Partial oracles

Properties:
- exploit necessary conditions of SUT.
- alleviate the oracle problem.
- sufficient conditions cannot be verified.
- may produce false negative results.
Partial oracles

Properties:
- exploit necessary conditions of SUT.
- alleviate the oracle problem.
- sufficient conditions cannot be verified.
- may produce false negative results.

Evaluation is necessary!
Mutation Analysis

- based on an arbitrary (defect-free) program $P: \mathcal{I} \rightarrow \mathcal{O}$.
- systematic error seeding $\Rightarrow$ mutants $M_i$.
- testing strategy generates $x \in \tilde{\mathcal{I}} \subseteq \mathcal{I}$.
- $M_i$ is equivalent to $P \iff M_i(x) = P(x) \ \forall \ x \in \mathcal{I}$.
- $M_i$ is killed $\iff \exists x \in \tilde{\mathcal{I}} : M_i(x) \neq P(x)$.

Mutation Score

$$S = \frac{M_k(\text{Number of killed mutants})}{M_t(\text{Number of non-equivalent mutants})}$$
Mutation Analysis

- based on an arbitrary (defect-free) program $P: \mathcal{I} \rightarrow \mathcal{O}$.
- systematic error seeding $\Rightarrow$ mutants $M_i$.
- testing strategy generates $x \in \tilde{\mathcal{I}} \subseteq \mathcal{I}$.
- $M_i$ is equivalent to $P \iff M_i(x) = P(x) \ \forall \ x \in \mathcal{I}$.
- $M_i$ is killed $\iff \exists x \in \tilde{\mathcal{I}}: M_i(x) \neq P(x)$.

**Mutation Score**

$$S = \frac{M_k(\text{Number of killed mutants})}{M_t(\text{Number of non-equivalent mutants})}$$
Overview

Objectives:
- assess quality of partial oracles.
- compare class based and traditional mutants.
- compare results with real faults.

Preconditions:
- complex and object-oriented implementation.
- dissociate non-terminating mutants.
Overview

Objectives:
- assess quality of partial oracles.
- compare class based and traditional mutants.
- compare results with real faults.

Preconditions:
- complex and object-oriented implementation.
- dissociate non-terminating mutants.
Methodology

Process is divided into two parts:
1. Ascertain suitable input values.
2. Assess the quality of applied partial oracles.
Methodology

Ascertain suitable inputs:

- generate images with different properties. (size, color depth, color offset, etc.)
- use original implementation as perfect oracle.
- determine mutation score for each image.
- classify categories of effectiveness.
- combine classes if necessary.
Methodology

Assess quality of partial oracle:

- use images derived from previous step.
- execute SUT and dissociate non-terminating mutants.
- verify only necessary conditions.
- determine mutation score for partial oracle.
Overview

Environment:
  - 413 traditional mutants.
  - 101 class based mutants.
- Random input generation.
- Metamorphic Relations as partial oracles.

Testing process
1. ascertained suitable input values.
2. assessed the quality of applied partial oracles.
Overview

Environment:
  - 413 traditional mutants.
  - 101 class based mutants.
- Random input generation.
- Metamorphic Relations as partial oracles.

Testing process
1. ascertain suitable input values.
2. assess the quality of applied partial oracles.
Input generation

Random image generation:
- use RGB color model.
- extend model for gray images.
Input generation

Random image generation:
- use RGB color model.
- extend model for gray images.
Input generation

Random image generation:
- use RGB color model.
- extend model for gray images.
Ascertain suitable input values

Categories of effectiveness (514 non-equivalent mutants)

<table>
<thead>
<tr>
<th>Image dimensions</th>
<th>$x &lt; y$</th>
<th>$x = y$</th>
<th>$x &gt; y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x &lt; y$</td>
<td>487</td>
<td>489</td>
<td>514</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
<td>465</td>
<td>489</td>
</tr>
<tr>
<td>$x &gt; y$</td>
<td></td>
<td></td>
<td>487</td>
</tr>
</tbody>
</table>
Metamorphic Relations

- input domain $I$.
- output range $O$.
- mapping $f : I \rightarrow O$.
- two relations $R_I \subseteq I^n$ and $R_O \subseteq I^n \times O^n$.

**Metamorphic Relation (Chen et al.)**

The pair $(R_I, R_O)$ is called Metamorphic Relation if and only if the following implication is fulfilled.

$$(i_1, \ldots, i_n) \in R_I \Rightarrow (i_1, \ldots, i_n, f(i_1), \ldots, f(i_n)) \in R_O$$
Metamorphic Relations

\[ R_i \rightarrow \text{FWT} \]

\[ \text{FWT} \rightarrow R_o \]
Applied relations

R1: Add an offset to the color values of each component.
R2: Multiply the color values of each component by a coefficient.
R3: Transpose the pixel array of each color component.
R4: Enlarge the image height (with zeros).
Assess Quality of partial oracles

Quality of applied relations (514 non-equivalent mutants)

<table>
<thead>
<tr>
<th>Relation</th>
<th>Total</th>
<th>Exceptions</th>
<th>Timeouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>484</td>
<td>312</td>
<td>8</td>
</tr>
<tr>
<td>R2</td>
<td>461</td>
<td>311</td>
<td>8</td>
</tr>
<tr>
<td>R3</td>
<td>480</td>
<td>312</td>
<td>8</td>
</tr>
<tr>
<td>R4</td>
<td>448</td>
<td>312</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metamorphic Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
</tr>
<tr>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
</tr>
<tr>
<td>R3</td>
</tr>
<tr>
<td>R4</td>
</tr>
</tbody>
</table>
Assess Quality of partial oracles

Quality of applied relations (413/101 non-equivalent mutants)

<table>
<thead>
<tr>
<th>Relation</th>
<th>Total</th>
<th>Exceptions</th>
<th>Timeouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>402</td>
<td>312</td>
<td>8</td>
</tr>
<tr>
<td>R2</td>
<td>381</td>
<td>311</td>
<td>8</td>
</tr>
<tr>
<td>R3</td>
<td>413</td>
<td>312</td>
<td>8</td>
</tr>
<tr>
<td>R4</td>
<td>398</td>
<td>312</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relation</th>
<th>Total</th>
<th>Exceptions</th>
<th>Timeouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>82</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>R2</td>
<td>80</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>R3</td>
<td>67</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>R4</td>
<td>50</td>
<td>30</td>
<td>0</td>
</tr>
</tbody>
</table>
Conclusion

- use class based and traditional mutation operators.
- partial oracles should be combined.
- effectiveness of smoke-tests may be language dependent.

Future work:
- confirm results with non-artificial faults.
- generalize approach.
- apply approach to other type of software.
Conclusion

- use class based and traditional mutation operators.
- partial oracles should be combined.
- effectiveness of smoke-tests may be language dependent.

Future work:
- confirm results with non-artificial faults.
- generalize approach.
- apply approach to other type of software.
Evaluating testing strategies for imaging software by means of Mutation Analysis

Thank you for your attention!