Definition and Evaluation of Mutation Operators for GUI-level Mutation Analysis

MUTATION 2015

Rafael Oliveira[1], Emil Alégroth[2],
Zebao Gao[3], Atif Memon[3]

[1]University of Sao Paulo – USP/ICMC
[2]Software Eng. and Tech. – Chalmers University
[3]Dep. of Computer Science – University of Maryland (UMD)

Graz/Austria April 12, 2015

Agenda

- Introduction and Conceptual Aspects
  - GUI(*Graphical User Interface*)-based applications;
  - Mutation testing;
  - Mutation operators for GUI-based applications;
  - GUI particularities and GUI testing.
- Mutation Operators for GUI-level
  - Removing, Adding, and Modifying.
- Empirical Evaluation
  - *Research Questions* (RQs);
  - Research strategy.
- Result Discussion and Final Remarks
  - Answers to Research questions;
  - Quantitative and qualitative analysis;
  - Discussion and threats to validity;
  - Final remarks.

Introduction – Context

- *Graphical User Interface* (GUI) based applications
  - Interaction between underlying code and users;
  - "Events": mouse clicks, mouse drags, keyboards shortcuts or commands, object manipulation, etc;
  - GUI testing: limited, non-generic strategies, specific tools.
- Mutation testing
  - Observing the program's behavior against slightly modifications;
  - May support the evaluation of new GUI testing strategies;

Introduction

- Traditional Mutation Operators are no effective for GUI-level testing.
- What is the necessity?
  - Special Mutation Operators;
  - MOs acting across widgets, properties (and values);
  - MOs changing GUI appearance;
  - MOs changing GUI states.
- This paper presents:
  - 18 “GUI-based” MOs;
  - A framework for automating the generation of these MOs;
  - Two empirical analysis on the fault-seeding effectiveness of the proposed MOs.

Conceptual aspects

- GUI-based applications
  - Event-based;
  - GUI States (*Ws, P, V*).
- GUI testing
  - First Generation tools;
  - Second Generation tools;
  - Third Generation tools;
  - Challenges: test cases modeled as a sequence of event;
  - Visual and underline-code faults.

MOs for GUI Level

- Three different classes:
  - removing;
  - adding; and
  - modifying code in the SUT.

MOs for GUI Level – Removing

#	Class	Mutant Operator	Acronym
1	Rem.	–Remove Existing Widget	REW
2		–Set Widget Invisible	SWI
3		–Remove Existing Listener	REL

MOs for GUI Level – Adding

#	Class	Mutant Operator	Acronym
4	Adding	–Add Identical Widget	AIW
5		–Add Similar Widget	ASW
6		–Add Different Widget	ADW
7		–Add Another Listener	AAL

MOs for GUI Level – Modifying

#	Class	Mutant Operator	Acronym
8	Modifying	–Expand/Reduce size of Windows and Widgets will Auto-adjust Their Sizes	EWWAR/ RWWAR
9	Modifying	–Expand size of windows and widgets will not auto-adjust their sizes	EWWNAR/ RWWNAR
10	Modifying	–Reduce size of windows to hide widgets	RWHW
11	Modifying	–Modify location of a widget to a proper location	MLWP
12	Modifying	–Modify location of a widget to edges of windows	MLWE
13	Modifying	–Modify location of a widget to overlap with another	MLWO
14	Modifying	–Modify size of widgets	MWS
15	Modifying	–Modify appearance of widgets	MWA
16	Modifying	–Modify type of widgets	MWT
17	Modifying	–Modify GUI library for widgets	MWL
18	Modifying	–Expand/Reduce size of Windows and Widgets will Adjust their Sizes	EWWAR/ RWWAS

Emp. Evaluation – Research Questions

The study is designed to answer the following *Research Questions* (RQ):

- **RQ1:** Are the defined GUI-based MOs more effective on seeding faults associated to the GUI level than traditional MOs?
- **RQ2:** Is it possible to automate the generation of GUI mutants?
- **RQ3:** Is it possible to use these proposed GUI operators on a real-world GUI application?

Emp. Evaluation – Research Strategy

We have divided our study into three parts:

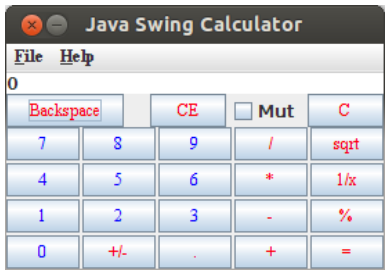
- A pilot experiment;
- A proof-of-concept;
- An analysis to evaluate the effect of the MOs on the GUI, regarding GUI testing.

Pilot Experiment

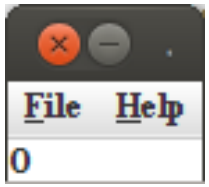
- Goals to support answering RQ1 and RQ2;
  - Evaluate the MOs' applicability and measure their efficacy;
  - compare the GUI effect of designed MOs and traditional mutation operators.
- Strategy:
  1. Choose a subject application (*Java Calculator*);
  2. Select three mutation operators from each class;
  3. Automate the generation of the mutants through the MOs;
  4. Generate mutants for each MO we have selected;
  5. Use MuJava to generate traditional method-level mutants for the same application (12 method-level MOs);
  6. Manually analysis of the effects of the MOs on the GUI-level.

Effects of MOs for GUI

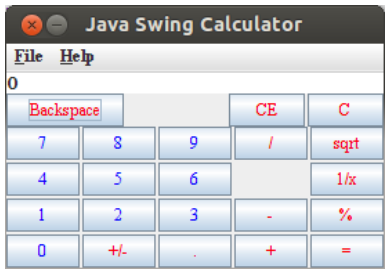
87 mutants were obtained (screenshots are available online)



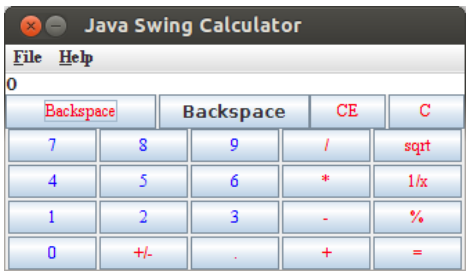
(a) ADW effect.



(b) REW effect.



(c) SWI effect.

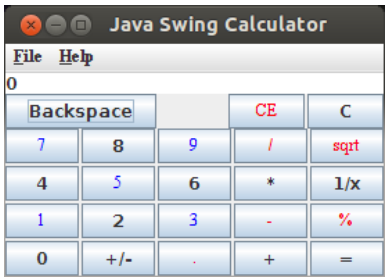


(d) ASW effect.

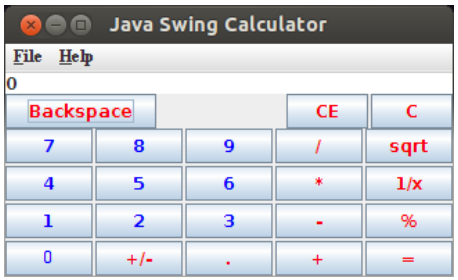
Figure: Four mutated instances of the *Java Calculator*.

Effects of tradition method-level MOs

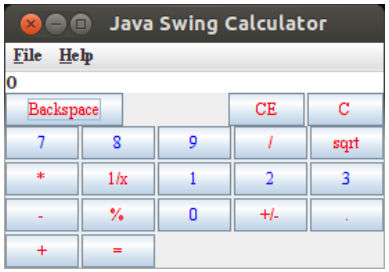
408 mutants were obtained (screenshots are available online)



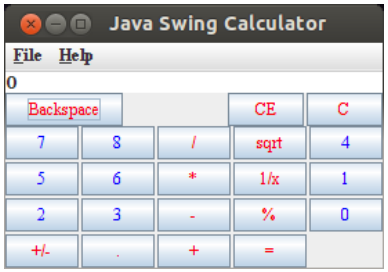
(a) AIOS effect.



(b) AOIU effect.



(c) COI effect.



(d) ROR effect.

Figure: Mutants of *Java Calculator* from traditional MuJava MOs.

Proof-of-concept: real-world application

- Goals to support answering RQ3;
  - Evaluate the MOs' applicability in real-world applications.
- Strategy:
  1. Choose a real-world application (WEKA);
  2. Studying its GUI code and selecting one of its GUIs to apply the proposed MOs;
  3. Automatically generates the mutants;
  4. Manually analyze the resulting mutants regarding their GUI effects and implication.

Original WEKA's GUI

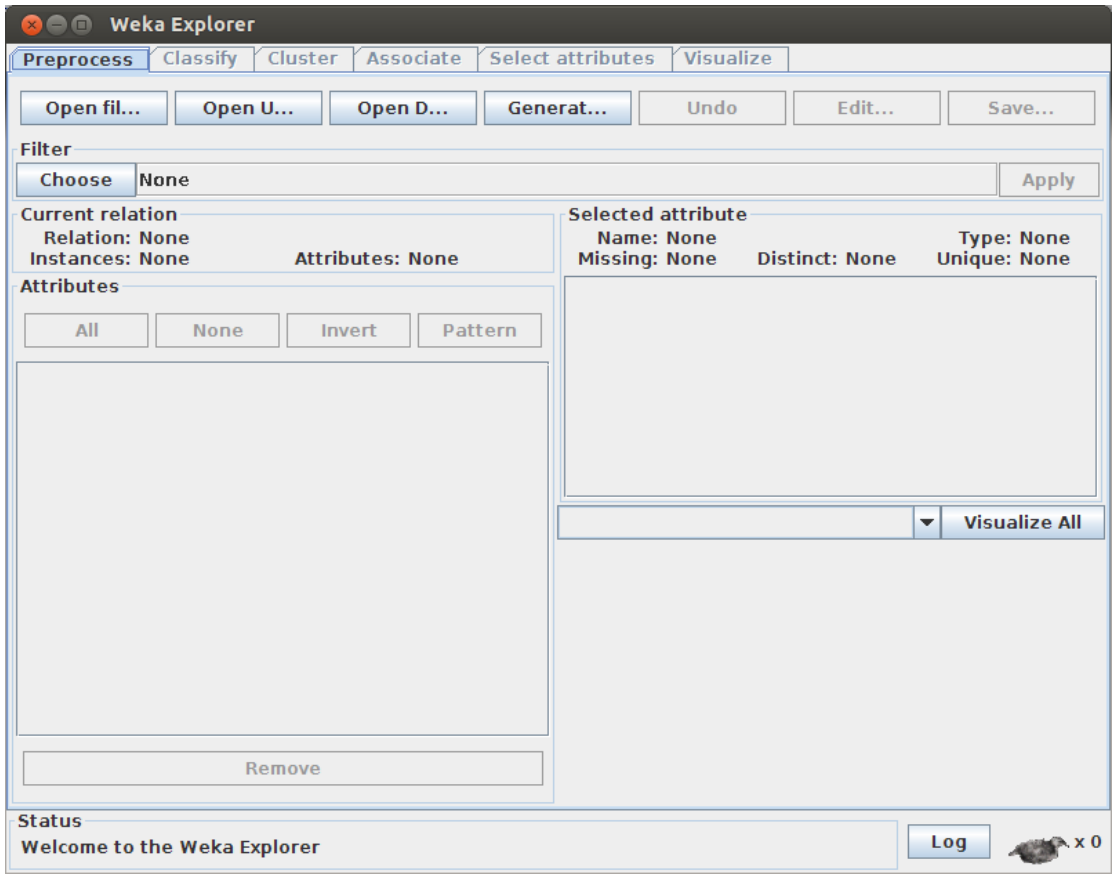


Figure: WEKA's original pre-processing GUI.

Mutants generated for WEKA

130 mutants were obtained (screenshots are available online)

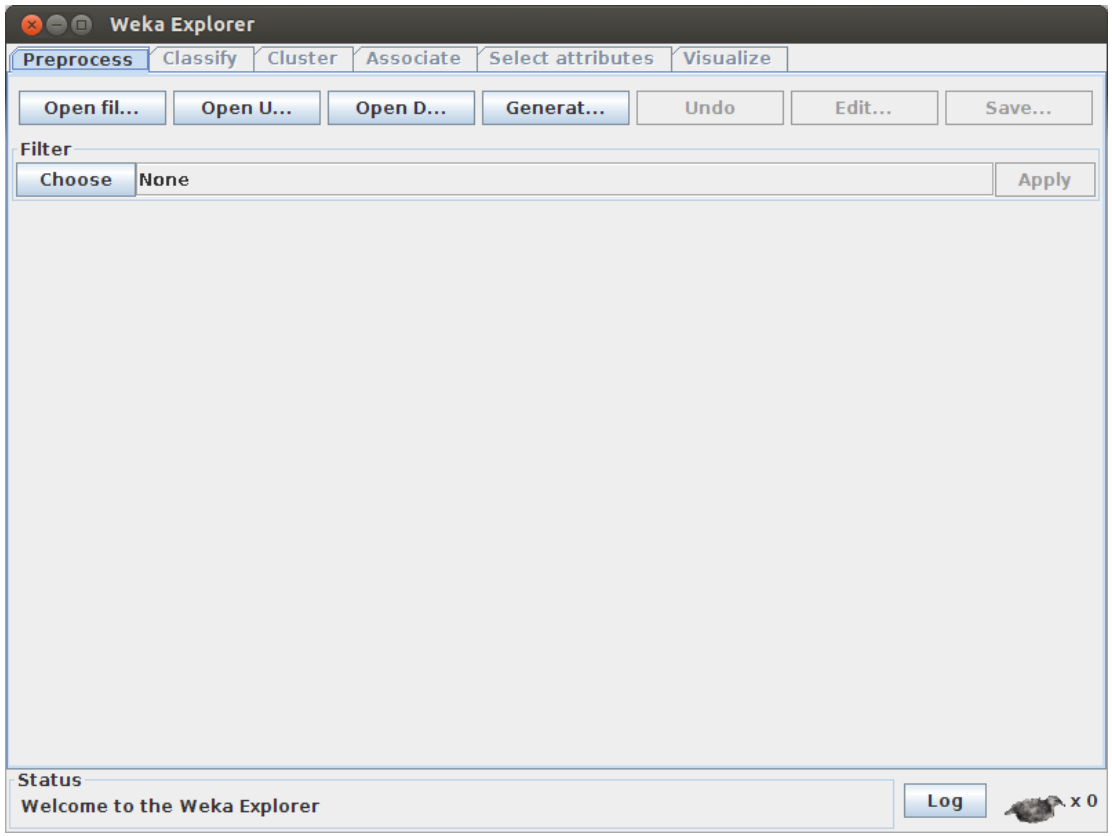


Figure: Mutant of *WEKA* generated from GUI-based MOs.

Quantitative analysis and of evaluations

- Numbers of mutants, visual equivalent mutants, structural equivalent mutants, capacity of generating mutants, etc;
- A yes/no questionnaire to identify properties associated with the GUI generated from the MOs.

Table: Questionnaire – Properties of GUI-based MOs.

	Yes/No – GUI-based MOs
P1	Does it change the GUI appearance?
P2	Does it change the GUI model/structure in testing tools?
P3	Can it change the SUT's input behavior?
P4	Can it change the SUT's output behavior
P5	Does it create equivalent mutants?
	P5.1 – Structural equivalent mutants?
	P5.2 – Visual equivalent mutants?

Results - Research Question 1

- RQ1: *"Traditional MOs presented a mean of 11.27% of GUI fault seeding effect, i.e. useful GUI mutants. In comparison, GUI-based MOs presented a mean of 83.90%"*
  - GUI-based MOs are more effective on seeding faults associated to the GUI level;
  - GUI-based MOs generate mutants able to represent more complete set of GUI-associated faults;
- Another issues:
  - Alive vs Dead mutants;
  - GUI effects;
  - Mutant equivalence;
  - Redundancy.

RQ1 - Statistics – Traditional vs GUI-based MOs

Statistics on using traditional mutation operators

trad. MO	# mut.	# dead/invalid	# alive	# alive + no GUI eff.	# alive + GUI eff.	# equiv.	"Good" Mut.	efficiency (%)
AOIS	183	17	166	17	149	135	14	**7.65**
AOIU	33	6	27	0	27	22	5	**15.15**
AORS	6	5	1	0	1	1	0	**0.00**
COI	25	1	24	0	24	19	5	**20.00**
LOI	44	11	33	0	33	26	7	**15.91**
ROR	92	5	87	0	87	72	15	**16.30**
AORB	16	0	16	0	16	16	0	**0.00**
COD	3	0	3	0	3	3	0	**0.00**
COR	6	0	6	0	6	6	0	**0.00**
total	408	45	363	17	346	300 (73%)	46	**11.27**

Statistics on using GUI-based mutation operators

trad. MO	# mut.	# dead/invalid	# alive	# alive + no GUI eff.	# alive + GUI eff.	# equiv.	"Good" Mut.	efficiency (%)
REW	13	0	13	0	13	0	13	**100**
SWI	11	0	11	0	11	0	11	**100**
REL	13	0	13	0	13	13	0	**0.00**
AIW	11	0	11	0	11	0	11	**100**
ASW	11	0	11	0	11	0	11	**100**
ADW	11	0	11	0	11	0	11	**100**
MWS	10	0	10	0	10	0	10	**100**
RWHW	5	0	4	0	4	0	4	**80**
EWWAR / RWWAR	2	0	2	0	2	0	2	**100**
total	87	0	86	0	86	13 (14.94%)	73	**83.90**

Results - Research Question 2

- RQ2: *"We were able to generate semi-automated scripts to generate mutants for seven different MOs for the Java Swing GUI library ..."*
  - Semi-automated: do not execute the mutant;
- Script's workflow:
  1. Read the original;
  2. Find some target command;;
  3. Replace, comment, or rejoin with other pre-defined code;
  4. Save the modified version;
  5. Repeat steps 2 to 5 until reach the end of the file.

Results - Research Question 3

- RQ3: *"The GUI-based MOs behaved properly in a real-world complex GUI-based application ..."*
  - 87 mutants generated for a GUI of WEKA;
  - We had no previously contact with WEKA's code;
  - WEKA has a highly documented source-code and well organized;
- The GUI properties were kept in more than 90% of the cases;

Discussion

- Benefits on using GUI-based MOs:
  - Mutants reflect directly on the GUI-level;
  - Avoid equivalent mutants;
  - Productivity in the GUI testing context.
- Threats to validity
  - Internal validity, external validity, construct validity, and conclusion validity.
- Mutation analysis and GUI testing: Future directions:
  - Need for standardization of approaches and strategies;
  - Implementation of effective tools.

Final Remarks

- Traditional method-level MOs are not effective for GUI testing;
- This study presents an empirical analysis of the fault seeding-effectiveness of generic GUI-based MOs;
- 18 MOs specially designed for the GUI level of abstractions were proposed (GUI-based MOs);
- The main findings are:
  - GUI-based MOs more effective than traditional MOs for the fault-seeding process;
  - It is possible to implement supporting tools for GUI-based MOs;
  - GUI-based MOs are useful on real-world applications.