How to Kill Them All: An Exploratory Study on the Impact of Code Observability on Mutation Testing

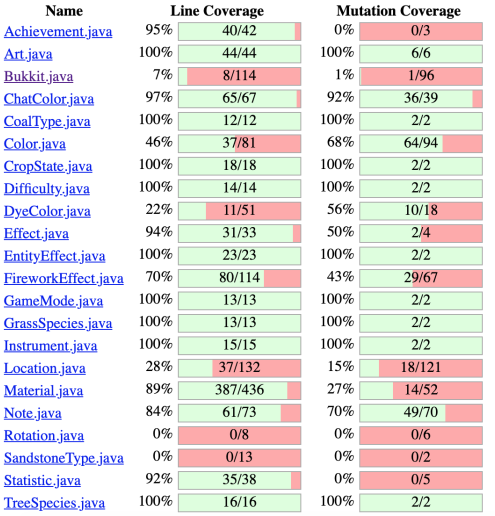
Qianqian Zhu
Ph.D. student

Co-author: Andy Zaidman, Annibale Panichella

Software Engineering Research Group,
Delft University of Technology, Netherlands

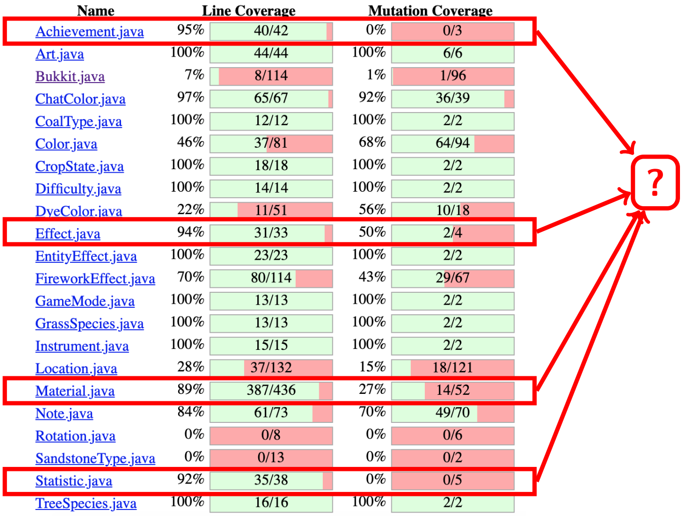
July 9, 2019

Confusion about mutation score



Name	Line Coverage		Mutation Coverage	
Achievement.java	95%	40/42	0%	0/3
Art.java	100%	44/44	100%	6/6
Bukkit.java	7%	8/114	1%	1/96
ChatColor.java	97%	65/67	92%	36/39
CoalType.java	100%	12/12	100%	2/2
Color.java	46%	37/81	68%	64/94
CropState.java	100%	18/18	100%	2/2
Difficulty.java	100%	14/14	100%	2/2
DyeColor.java	22%	11/51	56%	10/18
Effect.java	94%	31/33	50%	2/4
EntityEffect.java	100%	23/23	100%	2/2
FireworkEffect.java	70%	80/114	43%	29/67
GameMode.java	100%	13/13	100%	2/2
GrassSpecies.java	100%	13/13	100%	2/2
Instrument.java	100%	15/15	100%	2/2
Location.java	28%	37/132	15%	18/121
Material.java	89%	387/436	27%	14/52
Note.java	84%	61/73	70%	49/70
Rotation.java	0%	0/8	0%	0/6
SandstoneType.java	0%	0/13	0%	0/2
Statistic.java	92%	35/38	0%	0/5
TreeSpecies.java	100%	16/16	100%	2/2

Confusion about mutation score



Name	Line Coverage		Mutation Coverage	
Achievement.java	95%	40/42	0%	0/3
Art.java	100%	44/44	100%	6/6
Bukkit.java	7%	8/114	1%	1/96
ChatColor.java	97%	65/67	92%	36/39
CoalType.java	100%	12/12	100%	2/2
Color.java	46%	37/81	68%	64/94
CropState.java	100%	18/18	100%	2/2
Difficulty.java	100%	14/14	100%	2/2
DyeColor.java	22%	11/51	56%	10/18
Effect.java	94%	31/33	50%	2/4
EntityEffect.java	100%	23/23	100%	2/2
FireworkEffect.java	70%	80/114	43%	29/67
GameMode.java	100%	13/13	100%	2/2
GrassSpecies.java	100%	13/13	100%	2/2
Instrument.java	100%	15/15	100%	2/2
Location.java	28%	37/132	15%	18/121
Material.java	89%	387/436	27%	14/52
Note.java	84%	61/73	70%	49/70
Rotation.java	0%	0/8	0%	0/6
SandstoneType.java	0%	0/13	0%	0/2
Statistic.java	92%	35/38	0%	0/5
TreeSpecies.java	100%	16/16	100%	2/2

Research Questions

Our goal: to explore the relationship between code quality metrics and mutation testing

Research Questions

Our goal: to explore the relationship between code quality metrics and mutation testing

RQ1 What is the relation between *testability* metrics and the mutation score?

RQ2 What is the relation between *observability* metrics and the mutation score?

RQ3 What is the relation between the combination of *testability* and *observability* metrics and the mutation score?

RQ4 To what extent does the refactoring of anti-patterns based on testability and observability help in improving the mutation score?

Testability

64 existing code quality metrics in literature

Testability

64 existing code quality metrics in literature

27 method-level metrics

method-level			
COMP	Cyclomatic Complexity	HBUG	Halstead bugs
NOA	Number of Arguments	TDN	Total depth of nesting
NOCL	Number of Comments	CAST	Number of casts
NOC	Number of Comment Lines	LOOP	Number of loops
VDEC	Variable Declarations	NOPR	Number of operators
VREF	Variable References	NAND	Number of operands
NOS	Number of statements	CREF	Number of classes referenced
NEXP	Number of expressions	XMET	Number of external methods
MDN	Max depth of nesting	LMET	Number of local methods
HLTH	Halstead length	EXCR	Number of exceptions referenced
HVOC	Halstead vocabulary	EXCT	Number of exceptions thrown
HVOL	Halstead volume	MOD	Number of modifiers
HDIF	Halstead difficulty	NLOC	Lines of Code
HEFF	Halstead effort		

Testability

37 class-level metrics

class-level			
NOMT	Number of methods	NSUP	Number of Superclasses
LCOM	Lack of Cohesion of Methods	NSUB	Number of Subclasses
TCC	Total Cyclomatic Complexity	MI	Maintainability Index (including comments)
AVCC	Average Cyclomatic Complexity	MINC	Maintainability Index (not including comments)
MAXCC	Maximum Cyclomatic Complexity	COH	Cohesion
NOS	Number of statements	DIT	Depth of Inheritance Tree
HLTH	Cumulative Halstead length	LCOM2	Lack of Cohesion of Methods (2)
HVOL	Cumulative Halstead volume	CCOM	Number of Comments
HEFF	Cumulative Halstead effort	CCML	Number of Comment Lines
HBUG	Cumulative Halstead bugs	NLOC	Lines of Code
UWCS	Un Weighted class Size	RFC	Response for Class
NQU	Number of Queries	MPC	Message passing
NCO	Number of Commands	CBO	Coupling between objects
EXT	External method calls	FIN	Fan In
LMC	Local method calls	FOUT	Fan Out
HIER	Hierarchy method calls	R-R	Reuse Ratio
INST	Instance Variables	S-R	Specialization Ratio
MOD	Number of Modifiers	PACK	Number of Packages imported
INTR	Number of Interfaces		

Mutant Observability

“An expression in a program is *observable* in a *test case* if *the value of an expression* is changed, leaving the rest of the program intact, and the output of the system is changed correspondingly.”

Mutant Observability

"An expression in a program is *observable* in a $\boxed{\textit{test case}}$ if $\boxed{\textit{the value of an expression}}$ is changed, leaving the rest of the program intact, and the output of the system is changed correspondingly."

Mutant Observability comprises:

- **production code:** return type, access control modifiers, fault masking
- **test case:** test directness, assertion

Mutant Observability

19 newly-proposed metrics (1)

#	Name	Definition	Category
1	is_void	whether the return value of the method is void or not	return type
2	non_void_percent (class-level)	the percent of non-void methods in the class	
3	getter_percentage	the percentage of getter methods in the class	
4	is_public	whether the method is public or not	access control modifiers
5	is_static	whether the method is static or not	
6	is_nested (class-level)	whether the method is located in a nested class or not	fault masking (1)
7	nested_depth	the maximum number of nested depth	
8	(cond)	the number of conditions (if, if-else and switch) in the method	
9	(cond(cond))	the number of nested conditions (e.g.if{if{}}) in the method	

Mutant Observability

19 newly-proposed metrics (2)

#	Name	Definition	Category
10	(cond(loop))	the number of nested condition-loops (e.g. if{for{}}) in the method	fault masking (2)
11	(loop)	the number of loops (for, while and do-while) in the method	
12	(loop(cond))	the number of nested loop-conditions (e.g. for{if{}}) in the method.	
13	(loop(loop))	the number of nested loop-loops (e.g. for{for{}}) in the method.	
14	method_length	the number of lines of code in the method	
15	direct_test_no.	the number of test methods directly invoking the methods	test directness
16	test_distance	the shortest method call sequence required to invoke the method in test methods	
17	assertion_no.	the number of assertions in direct tests	assertion
18	assertion-McCabe_Ratio	the ratio between the total number of assertions in direct tests and the McCabe Cyclomatic complexity	
19	assertion_density	the ratio between the total number of assertions in direct tests and the lines of code in direct tests	

Experimental Study

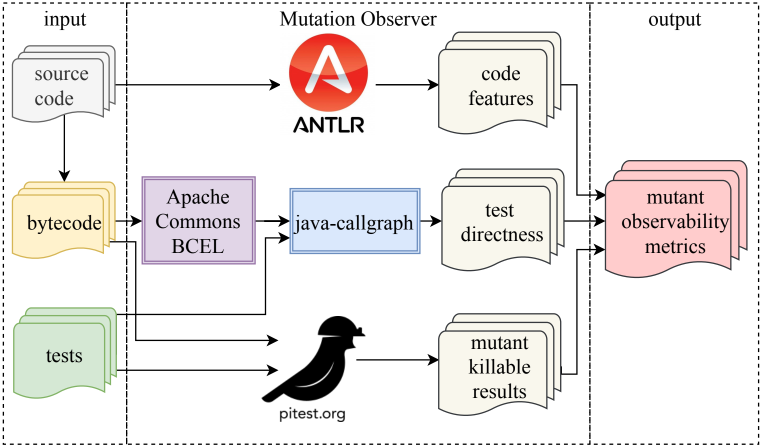
Six open-source projects from GitHub

pid	project	LOC	#Test	#Method	#Mutant	#Killed Mutant
1	Bukkit-1.7.9-R0.2	32373	432	2385	7325	947
2	commons-lang-LANG_3_7	77224	4068	2740	13052	11284
3	commons-math-MATH_3_6_1	208959	6523	6663	48524	38016
4	java-apns-apns-0.2.3	3418	91	150	429	247
5	jfreechart-1.5.0	134117	2175	7133	34488	11527
6	pysonar2-2.1	10926	269	719	3074	836
	overall	467017	13558	19790	106892	62857

Experimental Study

Tool:

- JHawk: existing metrics for testability
- MUTANT OBSERVER: mutant observability metrics



RQ1-RQ3 Testability vs. Observability vs. MS.

- **Pair-wise correlation:**
  Spearman's Rank Order (method-level)

```
for each metric $Metric_i$ for all methods:
    [rho,pval] = corr($Metric_i$, $MutScore$)
```

RQ1-RQ3 Testability vs. Observability vs. MS.

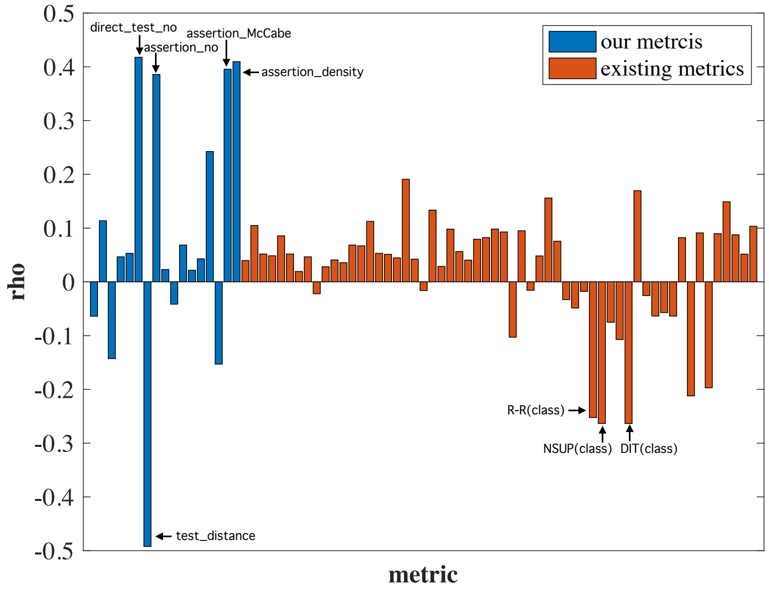
- **Pair-wise correlation:**
  Spearman's Rank Order (method-level)

```
for each metric $Metric_i$ for all methods:
  [rho,pval] = corr($Metric_i$, $MutScore$)
```

- **Interaction:** Random Forest (four models)

	testability	observability	combined
ZeroR (baseline)	1 (based on majority)		
Random Forest	2	3	4

Spearman's Results



Random Forest Results

- **classification effectiveness**

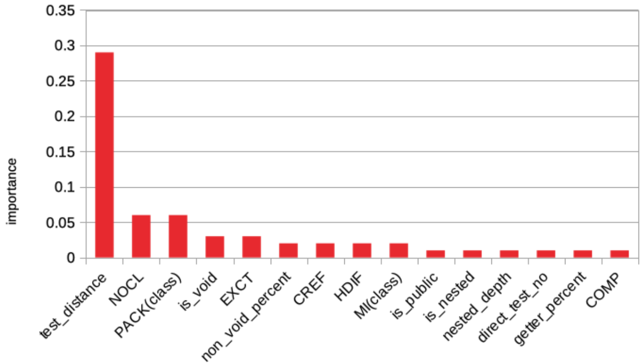
		precision	recall	AUC	error
ZeroR		0.569	0.569	0.5	0.4905
Random Forest	testability	0.862	0.862	0.928	0.2133
	observability	0.864	0.864	0.937	0.1846
	combined	0.905	0.905	0.963	0.1625

Random Forest Results

- **classification effectiveness**

		precision	recall	AUC	error
ZeroR		0.569	0.569	0.5	0.4905
Random Forest	testability	0.862	0.862	0.928	0.2133
	observability	0.864	0.864	0.937	0.1846
	combined	0.905	0.905	0.963	0.1625

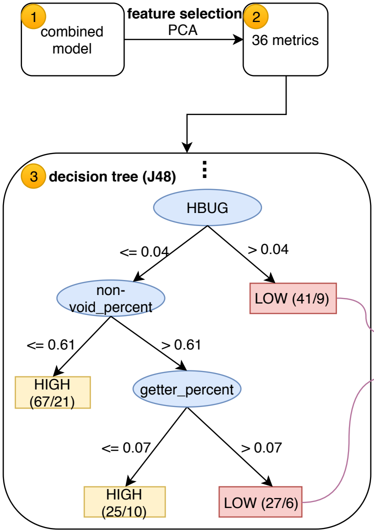
- **feature importance**



RQ4 Code Refactoring

RQ4 Code Refactoring

(1) anti-patterns/indicators



4 Anti-patterns	Correct instance	Incorrect instance	Accuracy
HBUG > 0.04	41	9	0.82
HBUG <=0.04 && non-void_percent > 0.61 && getter_percent > 0.07	27	6	0.818

RQ4 Code Refactoring

(2) case study on 16 code fragments (top 6 anti-patterns)

RQ4 Code Refactoring

(2) case study on 16 code fragments (top 6 anti-patterns)

- test_distance$>$5 $\rightarrow$ adding direct tests
- test_distance$\leq$5 $\rightarrow$ adding assertions
- is_public$=$0: private $\rightarrow$ public/protected
- three void methods $\rightarrow$ non-void
- one void method $\rightarrow$ adding a getter

Summary

- **What we have done**
    - 64 existing metrics for *testability*
    - 19 newly-proposed metrics for *mutant observability*
    - experimental study on 6 open-source projects (Java)
    - case study on 16 code fragments

Summary

- **What we have done**
  - 64 existing metrics for *testability*
  - 19 newly-proposed metrics for *mutant observability*
  - experimental study on 6 open-source projects (Java)
  - case study on 16 code fragments
- **What we have learned**
  - 64 existing metrics $\rightarrow$ not strongly correlated (rho$<$0.27)
  - 19 mutant observability metrics $\rightarrow$ stronger (rho$<$0.5)
  - anti-patterns $\rightarrow$ actionable insights