lab-soft
software engineering laboratory

Understanding Subsumption of First- and Second-Order Mutants

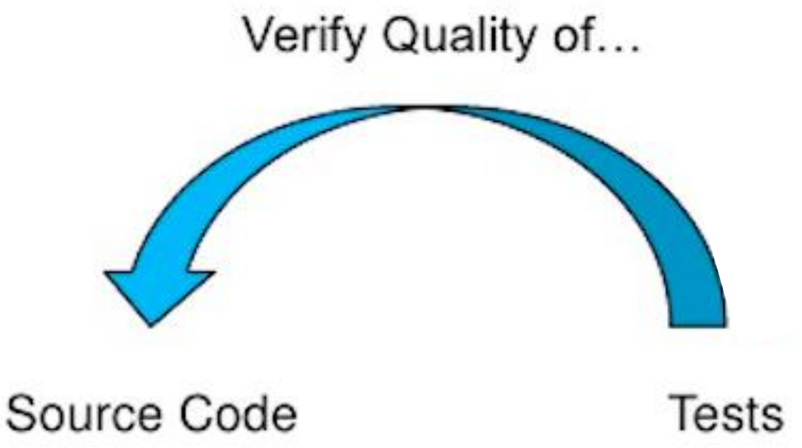
João Paulo de Freitas Diniz

LabSoft Seminar. Jun 21st, 2024

Outline

- Mutation testing
- Mutants subsumption
- Dynamic mutant subsumption graphs
- Study design
- Preliminary results
- Comparison with SS2OMs reduction
- Final remarks

Introduction



Mutation Testing

- Introducing **artificial syntactic changes** (**mutations**) into original source code
  - Intending to represent real common programming bugs
  - Changed programs are called **mutants**
- Running test cases on mutants
  - Result different from original: mutant **killed**
  - Otherwise: **alive**

Example of a mutant

Mutation place:

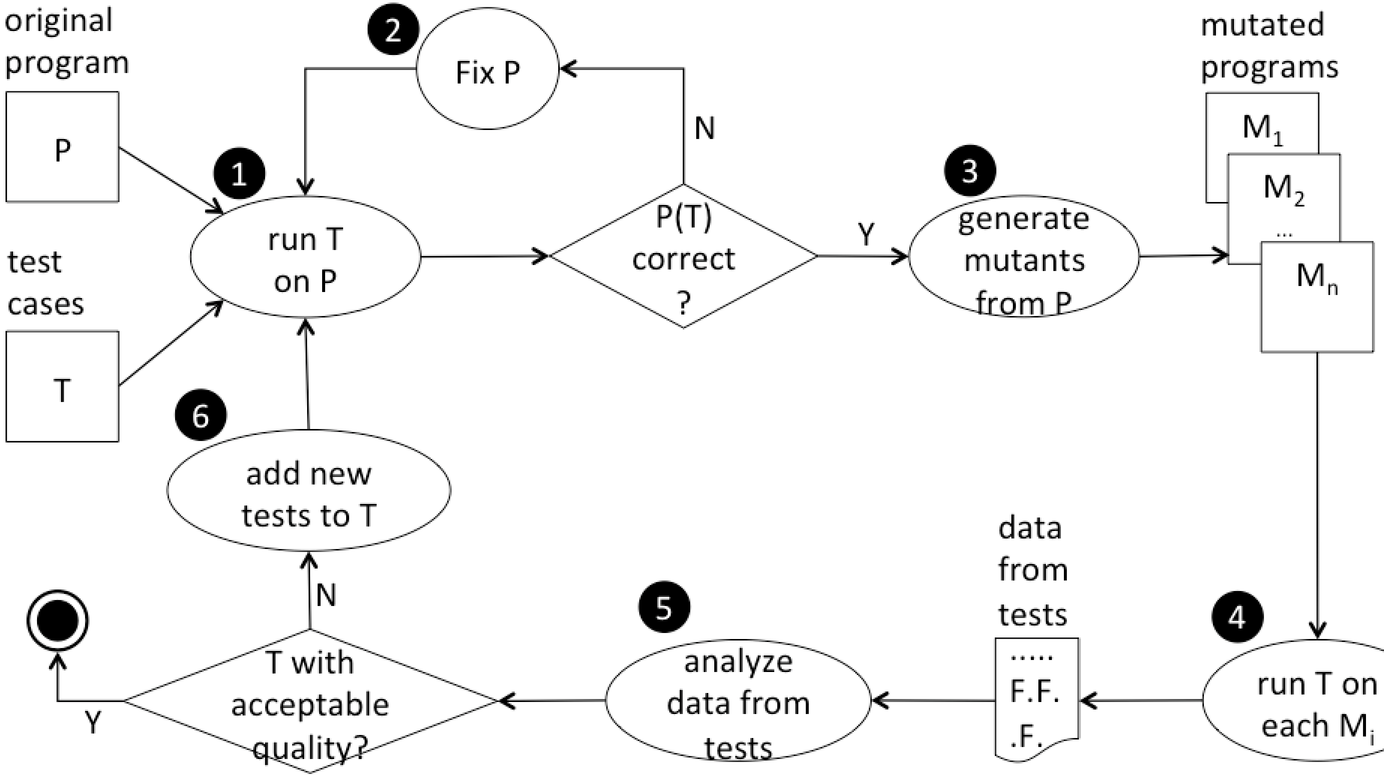
```
public class Taxes {

    double simpleTax(double amount) {

        return amount * 0.2;
    }
}
```

Example of a mutant

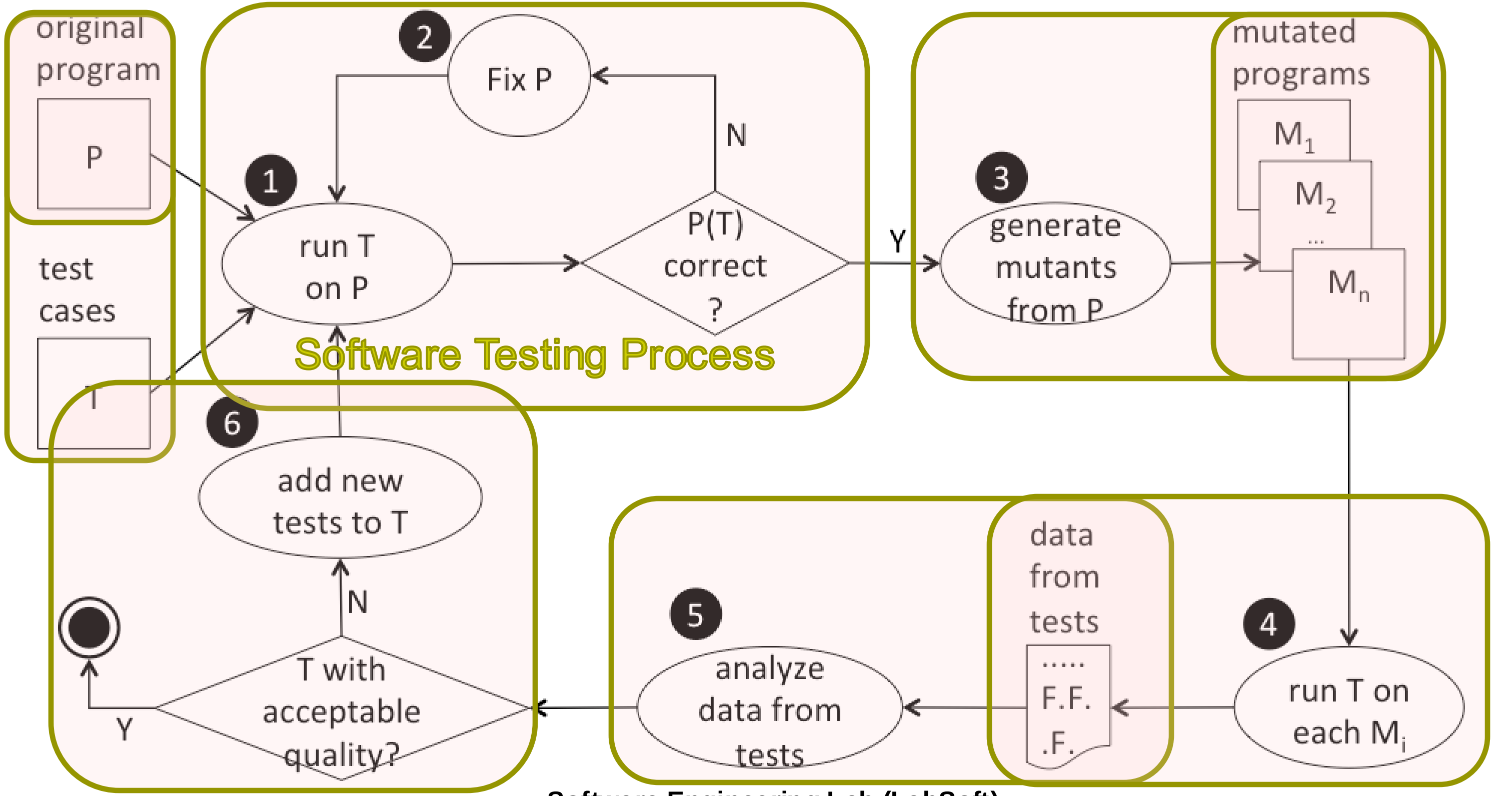
$* \rightarrow +$

```
public class Taxes {

      double simpleTax(double amount) {

            return amount + 0.2;
      }
}
```

Mutation testing process

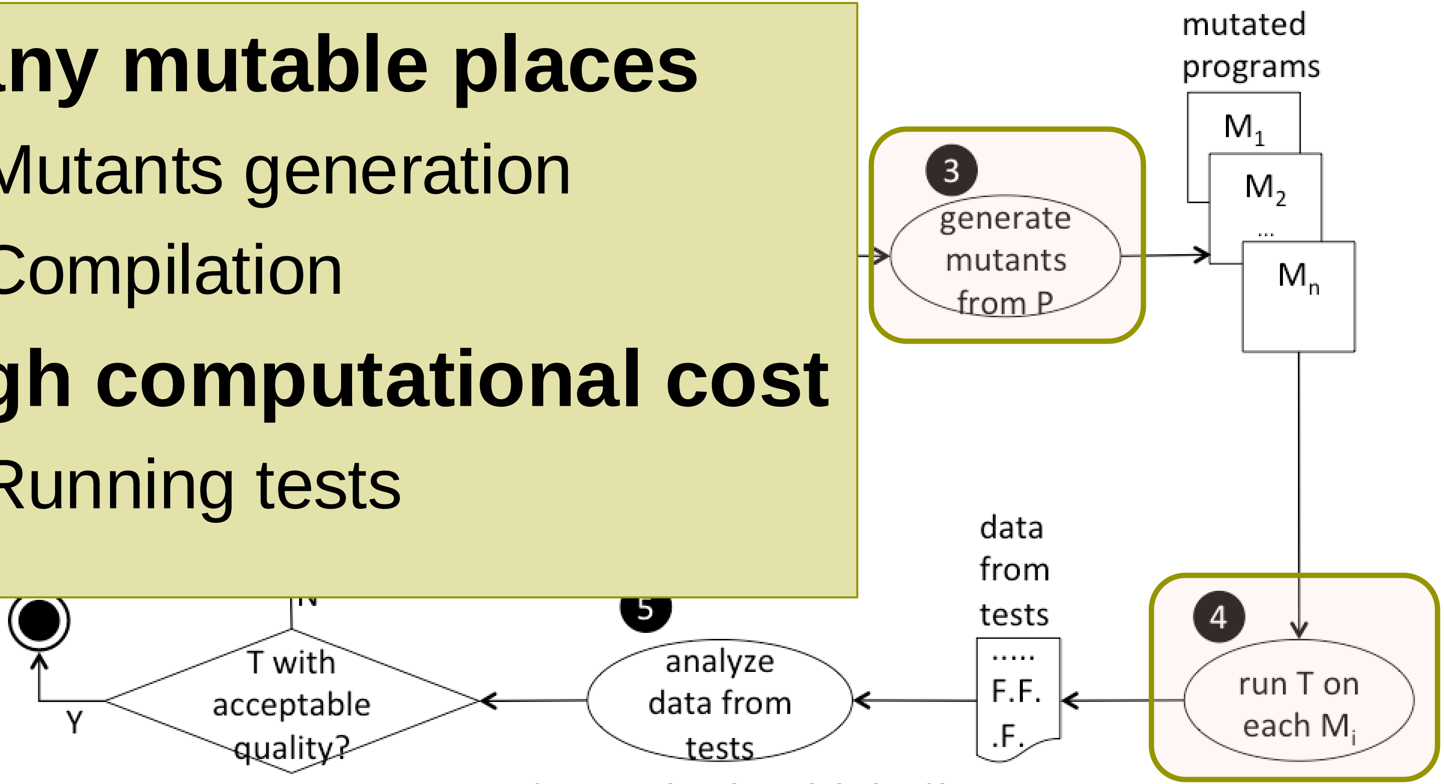


Mutation testing process



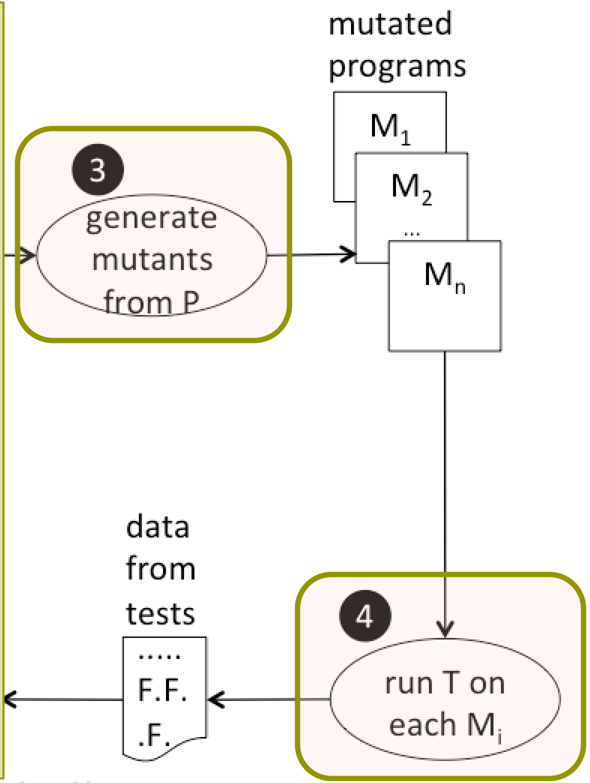
Mutation testing drawbacks

3. **Many mutable places**
   - Mutants generation
   - Compilation
4. **High computational cost**
   - Running tests



Mutation testing drawbacks

- Cost reduction techniques
  - Number of test cases
  - Test case prioritization
  - Number of mutants
    - **subsumption**



Mutants subsumption

Contextualization

```
def greaterThan(a, b):
    return a > b   # original
    return a >= b  # mutant 1
    return a <= b  # mutant 2
```

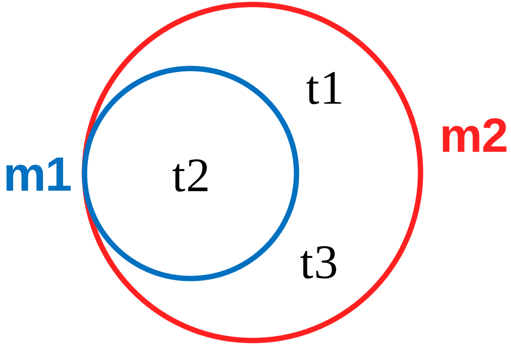
Contextualization

```
def greaterThan(a, b):
    return a > b    # original
    return a >= b # mutant 1
    return a <= b # mutant 2
```

	Test	orig	m1	m2
t1	`assertTrue(greaterThan(6, 5))`	✓	✓	✗
t2	`assertFalse(greaterThan(5, 5))`	✓	✗	✗
t3	`assertFalse(greaterThan(5, 6))`	✓	✓	✗

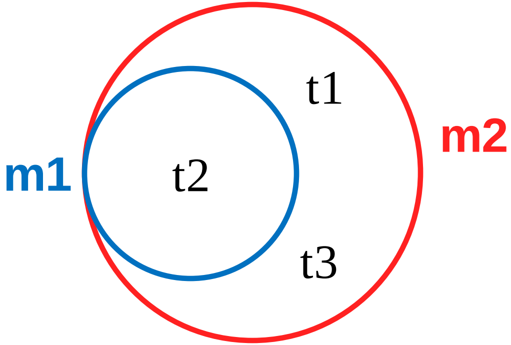
Contextualization

- Killing tests



Contextualization

- All test sets that kill **m1** also kill **m2**



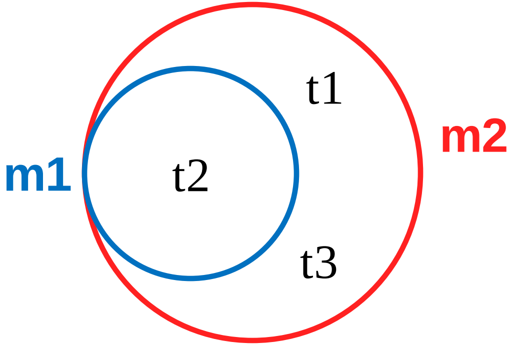
Definition

- The notion of **subsumption** is used to compare **test criteria**:

"a criterion C1 ***subsumes*** *C2 if every set of tests that satisfy C1 also satisfy C2"*

Conclusion

- **m1 subsumes m2**



Conclusion

- If we know beforehand that
  - **m1** subsumes **m2**
- Therefore,
  - **m2** should not have been generated

Cost reduction: fewer mutants to run the test suite against

Dynamic mutant subsumption graphs

Example

test	m1	m2	m3	m4	m5
t1	✗	✗		✗	✗
t2	✗		✗	✗	
t3				✗	
t4		✗		✗	✗

Subsumption relationships

test	m1	m2	m3	m4	m5
t1	✗	✗		✗	✗
t2	✗		✗	✗	
t3				✗	
t4		✗		✗	✗

$m1 \rightarrow m4$

$m2 \rightarrow m4$

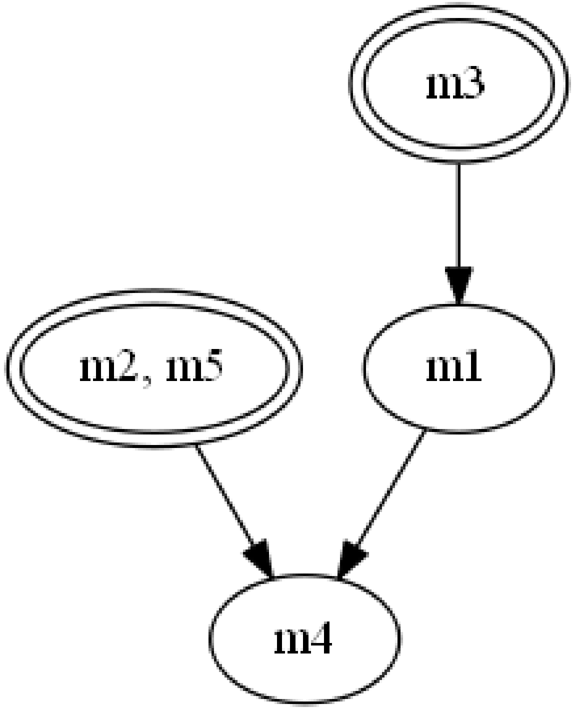
$m3 \rightarrow m1$

$m3 \rightarrow m4$

$m5 \rightarrow m4$

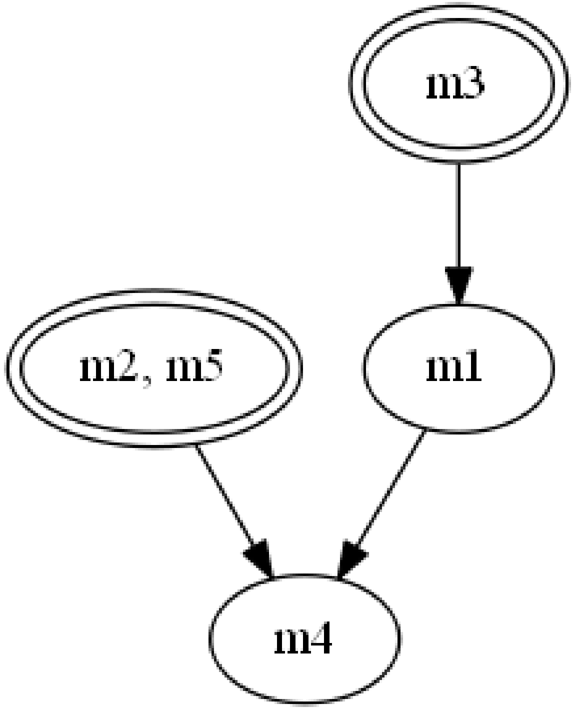
Subsumption graph

Test	m1	m2	m3	m4	m5
t1	✗	✗		✗	✗
t2	✗		✗	✗	
t3				✗	
t4		✗		✗	✗



Conclusion

- **Root nodes are kept**
  - 2 minimal
  - 3 mutants
- Remaining nodes
  - are disregarded
  - (redundants)



Study design

Dataset: 9 Java systems

System	Version	LOC	# Tests	JUnit	+16K mutants
Vending Machine	Exceptions	~100	35	4	**57**
Triangle	n/a	34	12	4	**138**
Monopoly	n/a	1,181	124	3	**866**
Commons CSV	1.8	~2k	325	4	**925**
Commons CLI	1.4	2,699	318	4	**1,082**
ECal	2003.10	3,626	224	3	**1,207**
Commons Validator	1.6	7,409	536	4	**3,197**
Gson	2.9.0	> 10k	1,089	3 and 4	**3,712**
Chess	n/a	4,924	930	3 and 4	**5,287**

Study steps

- Compute the killing tests for each mutant
- Generate the subsumption graph
- Retrieve the root (minimal) nodes

Preliminary results

Subsumption analysis

System	mutants	minimal nodes	remaining mutants
Vending Machine	57	8	19
Triangle	138	12	59
Monopoly	866	48	127
Commons CSV	925	79	260
Commons CLI	1,082	101	238
ECal	1,213	98	281
Commons Validator	3,197	137	858
Gson	3,712	288	876
Chess	5,319	344	1,018
Total	**16,471**	**1,115**	**3,376**

Highlight on Triangle

12 minimal nodes:

{91}

{65, 67, 68, 70, 75, 77, 80, 52, 85, 56, 57, 58, 59, 63}

{35, 37, 38, 39, 108, 112, 114, 115, 116, 118, 119}

{129, 130, 131, 46, 122, 123, 124, 127}

{11}

{62}

{79}

{5}

{96, 132, 136, 121, 106, 111}

{69}

{97, 99, 100, 101, 103, 104, 23, 24, 25, 26, 27, 28, 93}

{0}

Comparison with SS2OMs reduction

Isolated reductions

System	FOMs	Via subsumption graph	Via SS2OMs
Vending Machine	57	66,67%	14.04%
Triangle	138	57,25%	36.23%
Monopoli	866	85,33%	24.13%
Commons CSV	925	71,89%	20.32%
Commons CLI	1,082	78,00%	31.05%
ECal	1,213	76,83%	22.54%
Commons Validator	3,197	73,01%	24.52%
Gson	3,712	76,40%	20.42%
Chess	5,319	80,86%	20.38%
Overall	**16,471**	**77.35%**	**22.37%**

Isolated r

System			Ms
Vending Machine			
Triangle			
Monopoli			
Commons CSV			
Commons CLI			
ECal			
Commons Validator			
Gson			
Chess			
Overall	**16,471**	**77.35%**	**22.37%**

Final remarks

Investigate

- Can SS2OMs reduce even more the non-subsumed mutants?

- Is it correct keeping only one mutant from each minimal set?

Reference

B. Kurtz, P. Ammann, M. E. Delamaro, J. Offutt and L. Deng

Mutant Subsumption Graphs, *2014*

IEEE Seventh International Conference on Software Testing Verification and Validation Workshops (Mutation)

Cleveland, OH, USA, pp. 176-185

doi: 10.1109/ICSTW.2014.20.

lab-soft
software engineering laboratory

Software Engineering Lab (LabSoft)
http://labsoft.dcc.ufmg.br/

Questions?