COMP5111 – Fundamentals of Software Testing and Analysis

Automated Fault Localization

Shing-Chi Cheung

Computer Science & Engineering

HKUST

https://hackthology.com/how-to-evaluate-statistical-fault-localization.html

Software maintenance

- Testing
  - Detect a fault
- Fault localization
  - Locate the fault
- Patching
  - Fix the fault

Fault localization is tedious

- One of the most frustrated processes
- Require high concentration
- Familiar with program logic.

Traditional approaches

- Insert print statements
- Use debuggers and set breakpoints
- Add assertions
- Examine core dump or stack trace

Rely on thorough program understanding & expert knowledge

May we have faults be automatically located?

http://www.youtube.com/watch?v=JkimgY0NGSc

- A demo of GZoltar

Fault Localization

mid( ) {	Runs					
int x, y, z, m;	1	2	3	4	5	6
read(“Enter 3 numbers:”, x, y, z);						
m = z;						
if (y < z) {						
if (x < y)						
m = y;						
else if (x < z)						
m = y;						
} else {						
if (x > y)						
m = y;						
else if (x > z)						
m = x;						
}						
print(“Middle number is:”, m);						
}	✓	✓	✓	✓	✗	✓

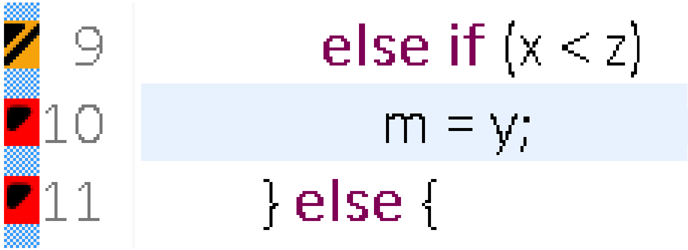
- Runs
  - 1: (1,1,2)
  - 2: (0,1,2)
  - 3: (2,1,0)
  - 4: (0,2,1)
  - 5: (1,0,2)
  - 6: (2,0,1)

GZoltar

- Likely faults are colored in red.
- Less likely faults are colored in orange.
- More less likely faults are

MyClass.java

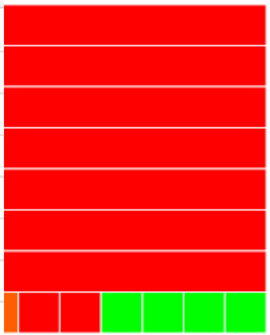
```
3  public class MyClass {
4      public static int mid(int x, int y, int z) {
5          int m = z;
6          if (y < z) {
7              if (x < y)
8                  m = y;
9              else if (x < z)
10                 m = y;
11         } else {
12             if (x > y)
13                 m = y;
14             else if (x > z)
15                 m = x;
16         }
```



Problems | Javadoc | Declaration | Coverage

0 errors, 7 warnings, 0 others

Description	Resource	Path	Location	Type
Warnings (7 items)				
Fault likelihood: 0.40824828	MyClass.java	/FaultLocalization/...	line 5	GZoltar Warni...
Fault likelihood: 0.40824828	MyClass.java	/FaultLocalization/...	line 6	GZoltar Warni...
Fault likelihood: 0.40824828	MyClass.java	/FaultLocalization/...	line 17	GZoltar Warni...
Fault likelihood: 0.5	MyClass.java	/FaultLocalization/...	line 7	GZoltar Orange
Fault likelihood: 0.57735026	MyClass.java	/FaultLocalization/...	line 9	GZoltar Orange
Fault likelihood: 0.70710677	MyClass.java	/FaultLocalization/...	line 10	GZoltar Error
Fault likelihood: 0.70710677	MyClass.java	/FaultLocalization/...	line 11	GZoltar Error



Fault Localization

mid( ) {	Runs					
int x, y, z, m;	1	2	3	4	5	6
read(“Enter 3 numbers:”, x, y, z);	●	●	●	●	●	●
m = z;	●	●	●	●	●	●
if (y < z) {	●	●	●	●	●	●
if (x < y)	●	●			●	●
m = y;		●				
else if (x < z)	●				●	●
m = y;	●				●	
} else {			●	●		
if (x > y)			●	●		
m = y;			●			
else if (x > z)				●		
m = x;						
}						
print(“Middle number is:”, m);	●	●	●	●	●	●
}	✓	✓	✓	✓	✗	✓

- Runs
  - 1: (1,1,2)
  - 2: (0,1,2)
  - 3: (2,1,0)
  - 4: (0,2,1)
  - 5: (1,0,2)
  - 6: (2,0,1)

Fault Localization

mid( ) {	Runs					
	1	2	3	4	5	6
int x, y, z, m;						
read(“Enter 3 numbers:”, x, y, z);	●	●	●	●	●	●
m = z;	●	●	●	●	●	●
if (y < z) {	●	●	●	●	●	●
if (x < y)	●	●			●	●
m = y;		●				
else if (x < z)	●				●	●
m = y; // *** BUG ***	●				●	
} else {			●	●		
if (x > y)			●	●		
m = y;			●			
else if (x > z)				●		
m = x;						
}						
print(“Middle number is:”, m);	●	●	●	●	●	●
}	✓	✓	✓	✓	✗	✓

- Runs
  - 1: (1,1,2)
  - 2: (0,1,2)
  - 3: (2,1,0)
  - 4: (0,2,1)
  - 5: (1,0,2)
  - 6: (2,0,1)

Fault Localization

mid( ) {	Runs					
	1	2	3	4	5	6
int x, y, z, m;						
read("Enter 3 numbers:", x, y, z);	●	●	●	●	●	●
m = z;	●	●	●	●	●	●
if (y < z) {	●	●	●	●	●	●
if (x < y)	●	●			●	●
m = y;		●				
else if (x < z)	●				●	●
m = y; // *** BUG ***	●				●	
} else {			●	●		
if (x > y)			●	●		
m = y;			●			
else if (x > z)				●		
m = x;						
}						
print("Middle number is:", m);	●	●	●	●	●	●
}	✓	✓	✓	✓	✗	✓

- Premise
  - Bugs participate more often in failing tests than passing tests
  - RIP model
    - ⇨ Reachability
    - Infection
    - Propagation

Coincidental Correctness

mid( ) {	Runs					
int x, y, z, m;	1	2	3	4	5	6
read("Enter 3 numbers:", x, y, z);	●	●	●	●	●	●
m = z;	●	●	●	●	●	●
if (y < z) {	●	●	●	●	●	●
if (x < y)	●	●			●	●
m = y;		●				
else if (x < z)	●				●	●
m = y; // *** BUG ***	●				●	
} else {			●	●		
if (x > y)			●	●		
m = y;			●			
else if (x > z)				●		
m = x;						
}						
print("Middle number is:", m);	●	●	●	●	●	●
}	✓	✓	✓	✓	✗	✓

- Occurs when a faulty statement is executed but does not lead to a failure.

Ranking function - Tarantula

- J. A. Jones and M. J. Harrold, "Empirical evaluation of the Tarantula automatic fault-localization technique," in *Proc. of the 20th IEEE/ACM Conference on Automated Software Engineering*, pp. 273-282, Long Beach, California, USA, December, 2005

$$X/X+Y,\ X=(N_{EF}/N_F)\ \&\ Y=(N_{ES}/N_S)$$

X: Participation in failing tests
Y: Participation in passing tests

N_{EF} : Number of failing tests executing the statement

N_{ES} : Number of passing tests executing the statement

N_F : Number of failing tests

N_S : Number of passing tests

Fault Localization

$X/X+Y$, $X=(N_{EF}/N_F)$ & $Y=(N_{ES}/N_S)$

$N_F: 1, N_S: 5$

mid( ) {	Runs						Tarantula
int x, y, z, m;	1	2	3	4	5	6	
read("Enter 3 numbers:", x, y, z);	●	●	●	●	●	●	0.5
m = z;	●	●	●	●	●	●	0.5
if (y < z) {	●	●	●	●	●	●	0.5
if (x < y)	●	●			●	●	0.625
m = y;		●					0.0
else if (x < z)	●				●	●	0.714
m = y; // *** BUG ***	●				●		**0.833**
} else {			●	●			0.0
if (x > y)			●	●			0.0
m = y;			●				0.0
else if (x > z)				●			0.0
m = x;							0.0
}							0.0
print("Middle number is:", m);	●	●	●	●	●	●	0.5
}	✓	✓	✓	✓	✗	✓	

Can we further improve the accuracy?

Fault Localization

$X/X+Y$, $X=(N_{EF}/N_F)$ & $Y=(N_{ES}/N_S)$

$N_F: 1$, $N_S: 4$

- Ignore successful runs identical to failing tests

mid( ) {	**Runs**					**Tarantula**	
	2	3	4	5	6		
int x, y, z, m;							
read(“Enter 3 numbers:”, x, y, z);	●	●	●	●	●	0.5	
m = z;	●	●	●	●	●	0.5	
if (y < z) {	●	●	●	●	●	0.5	
if (x < y)	●			●	●	0.625	
m = y;	●					0.0	
else if (x < z)				●	●	0.714	→ 0.8
m = y; // * BUG *****				●		**0.833**	→ **1.0**
} else {		●	●			0.0	
if (x > y)		●	●			0.0	
m = y;		●				0.0	
else if (x > z)			●			0.0	
m = x;						0.0	
}						0.0	
print(“Middle number is:”, m);	●	●	●	●	●	0.5	
}	✓	✓	✓	✗	✓		

Fault Localization

$X/X+Y$, $X=(N_{EF}/N_F)$ & $Y=(N_{ES}/N_S)$

$N_F: 1$, $N_S: 1$

mid( ) {	Runs 2	Runs 3	Runs 4	Runs 5	Runs 6	Tarantula
int x, y, z, m;						
read(“Enter 3 numbers:”, x, y, z);	●	●	●	●	●	0.5
m = z;	●	●	●	●	●	0.5
if (y < z) {	●	●	●	●	●	0.5
if (x < y)	●			●	●	0.625
m = y;	●					0.0
else if (x < z)				●	●	0.714 → 0.5
m = y; **// *** BUG *** **				●		**0.833** → **1.0**
} else {		●	●			0.0
if (x > y)		●	●			0.0
m = y;		●				0.0
else if (x > z)			●			0.0
m = x;						0.0
}						0.0
print(“Middle number is:”, m);	●	●	●	●	●	0.5
}	✓	✓	✓	✗	✓	

- Successful runs do not equally contribute to fault localization.
- Use successful runs most similar to the failing tests.

Use a better ranking function - Ochiai

- The formula consists of two components
  - N_{EF}/N_F: The chances that a statement E is executed in a failure
  - $N_{EF}/(N_{EF}+N_{ES})$: The chances of failure whenever E is executed
  - It ignores the total number of passing tests

$$\frac{N_{EF}}{\sqrt{N_F \times \underbrace{(N_{EF} + N_{ES})}_{\text{Total number of tests executing the statement E}}}}$$

N_{EF} : Number of failing tests executing the statement
N_{ES} : Number of passing tests executing the statement
N_F : Number of failing tests
N_S : Number of passing tests

ignore this factor?

Use a better ranking function - Ochiai

- A. Ochiai, "Zoogeographic studies on the soleoid fishes found in Japan and its neighboring regions," *Bull. Japan Soc. Sci. Fish* 22, 526–530, 1957
- R. Abreu, P. Zoeteweij, R. Golsteijn, and A.J.C. van Gemund, "A Practical Evaluation of Spectrum-based Fault Localization," *Journal of Systems and Software*, 82(11):1780 - 1792, 2009
- https://hackthology.com/how-to-evaluate-statistical-fault-localization.html

$$\frac{N_{EF}}{\sqrt{N_F \times \underbrace{(N_{EF} + N_{ES})}_{\text{Total number of runs executing the statement E}}}}$$

N_{EF} : Number of failing tests executing the statement
N_{ES} : Number of passing tests executing the statement
N_F : Number of failing tests
N_S : Number of passing tests

ignore this factor?

Fault Localization

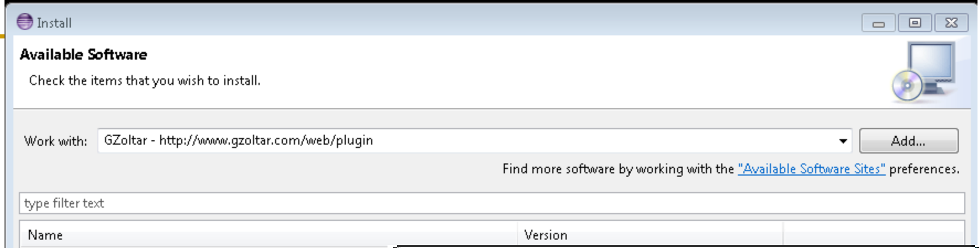
$$\frac{N_{EF}}{\sqrt{N_F \times (N_{EF} + N_{ES})}}$$

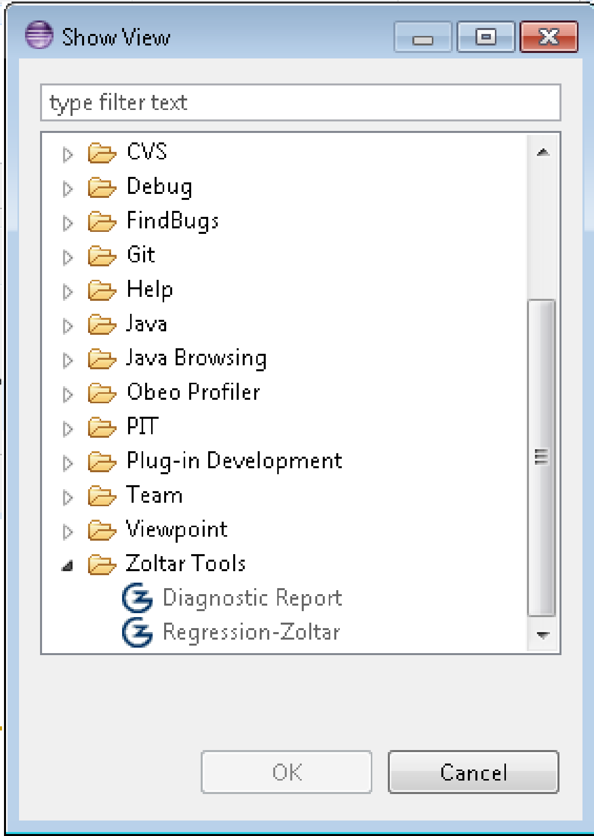
mid( ) {	Runs						Tarantula	Ochiai
int x, y, z, m;	1	2	3	4	5	6		
read(“Enter 3 numbers:”, x, y, z);	●	●	●	●	●	●	0.5	0.408
m = z;	●	●	●	●	●	●	0.5	0.408
if (y < z) {	●	●	●	●	●	●	0.5	0.408
if (x < y)	●	●			●	●	0.625	0.5
m = y;		●					0.0	0.0
else if (x < z)	●				●	●	0.714	0.577
m = y; // *** BUG ***	●				●		**0.833**	**0.707**
} else {			●	●			0.0	0.0
if (x > y)			●	●			0.0	0.0
m = y;			●				0.0	0.0
else if (x > z)				●			0.0	0.0
m = x;							0.0	0.0
}							0.0	0.0
print(“Middle number is:”, m);	●	●	●	●	●	●	0.5	0.408
}	✓	✓	✓	✓	✗	✓		

Empirically, **Ochiai** outperforms **Tarantula** in ranking suspicious statements at the top

Gzoltar – Eclipse Plugin

- Ranking function
  - Ochiai
- Eclipse Plugin
  - http://www.gzoltar.com/web/eclipse-plugin
  - Window->Show View->Other…
  - The software has not been maintained for a couple of years, the plugin may not work on the latest Eclipse version. You may run GZoltar using its standalone library with Java 8 in command line mode. It generates a report.
- Select the Java Project -> CTL-F5
- Standalone library
  - http://gzoltar.com/lib/
- API Documentation
  - http://gzoltar.com/api/
- Video
  - http://www.youtube.com/user/GZoltarDebugging



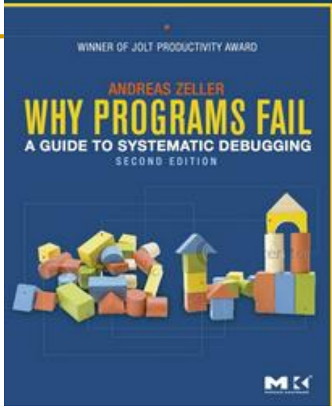


Generalizing the Concept

A test requirement is more suspicious if it participates in more failing than passing tests. The test requirement can be:

- Statement // *adopted so far in prior discussion*
- Statement sequence // *e.g., f.open() … f.close() … f.read()*
- Branch // *i.e., a specific evaluation of a predicate*
- Active boolean clause
- Prime path
- DU-path
- Mutants

Further readings



- Andreas Zeller, *Why Programs Fail: A Guide to Systematic Debugging (2nd Edition)*, Morgan Kaufmann, 2009.
  - https://www.udacity.com/course/cs259
  - https://www.st.cs.uni-saarland.de/whyprogramsfail/toc.php
- Xinming Wang, Shing-Chi Cheung, W.K. Chan, Zhenyu Zhang, Taming Coincidental Correctness: Coverage Refinement with Context Pattern to Improve Fault Localization, in *Proceedings of the 31st International Conference on Software Engineering (ICSE 2009)*, Vancouver, Canada, May 2009, pp. 45-55.
- Shay Artzi, Julian Dolby, Frank Tip, Marco Pistoia, Fault Localization for Dynamic Web Applications, *IEEE Transactions on Software Engineering 38(2)*, Mar/Apr 2012, pp. 314-335.
- Shin Yoo, Mark Harman, David Clark, Fault Localization Prioritization: Comparing Information-Theoretic and Coverage-Based Approaches, *ACM Transactions on Software Engineering and Methodology 22(3)*, July 2013.

Further readings

- Ming Wen, Rongxin Wu, Shing-Chi Cheung. How Well Do Change Sequences Predict Defects? Sequence Learning from Software Changes. In IEEE Transactions on Software Engineering 2018. To Appear.
- Ming Wen, Rongxin Wu, and Shing-Chi Cheung. Locus: Locating Bugs from Software Changes. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016), Singapore, Sept 2016, pp. 262-273.
- Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, and Sunghun Kim. CrashLocator: Locating Crashing Faults based on Crash Stacks . In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2014), San Jose, California, USA, July 2014, pp. 204-214. **ACM SIGSOFT Distinguished Paper Award.**
- Daming Zou, Jingjing Liang, Yingfei Xiong, Michael Ernst, Lu Zhang. An Empirical Study of Fault Localization Families and Their Combinations. IEEE Transactions on Software Engineering, Online First, January 2019.