Evaluating the effectiveness of BEN in localizing different types of software fault

Jaganmohan Chandrasekaran, UT Arlington

In collaboration with Laleh S. Gholamhosseing(UTA), Jeff Lei (UTA), Raghu Kacker (NIST), and D.Richard Kuhn (NIST)

April 10, 2016

Outline

- Introduction
- Three Fault Properties
- Experimental Design
- Experimental Results
- Conclusion

Overview of BEN

- BEN is a spectrum based fault localization tool.
  - Compares the spectra of failing and passing test executions
- BEN leverages the results obtained from combinatorial testing to perform fault localization.
- BEN locates the fault in two-phases:
  - Phase 1 : Identify failure-inducing combinations
  - Phase 2 : Produce a ranking of statements

Phase 1: Identify inducing combinations

- Identify suspicious combinations from the initial combinatorial test set with execution results
- Produce a ranking of suspicious combinations
- Add new tests to refine the ranking
- Repeat until a stopping condition is satisfied

Phase 2: Produce a ranking of statements

- Generate a small group of tests based on the failure-inducing combination
  - One core member (failing test) and several derived members (passing tests)
  - Core member (failing test) and derived members produce similar execution traces but have different outcomes.
- Compare the spectrum of core member to the spectrum of each derived member
- Statements are ranked in terms of their likelihood to be faulty

Effectiveness of BEN

- Measured in terms of the percentage of program statements (executable) the user has to inspect to locate the fault
  - The fewer statements to be inspected, the more effective
- Fault properties could be a significant factor that impacts the effectiveness of BEN

Fault Properties

- Accessibility
  - The degree of difficulty to reach (and execute) a fault during a program execution
- Input value sensitivity
  - Fault triggers a failure based on certain input values
- Control flow sensitivity
  - Fault triggers a failure while inducing a change of control flow in program execution

Problem Statement

- How do the three fault properties affect the effectiveness of BEN?

Outline

- Introduction
- Three Fault Properties
- Experimental Design
- Experimental Results
- Conclusion

Accessibility

- Accessibility score: The ratio of the number of tests that execute a faulty statement to the total number of tests
  - Example: if 9 out of 10 tests execute a faulty statement, accessibility score is 0.9.
- In practice, it is nearly always impossible to generate all possible tests.
  - A random test set can be used to estimate accessibility score

Input value sensitivity

- Fault executed by both passing and failing tests is considered as input value sensitive; otherwise, it is input value insensitive
- Generating all possible tests is not practical
  - A random test set is used to determine whether a fault is input value sensitive

Control flow sensitivity

- P: faulty program and P': error-free program
  - execute the failed tests (exhaustive test set) on P and P' and record their traces
  - compare the trace of each test from P and P'
  - at least one failed test trace from P is different from P', fault is control flow sensitive; otherwise, it is control flow insensitive
- Again, generating and executing all the failed tests is nearly impossible.
  - a practical option is to execute a random test set.

Example : Fault Properties

Program Statements

```
float applyDiscount(int totalPrice,int member, char type)
{
  float discount = 0.00;
//Fault 1 - correct :if(totalPrice>1000)
   if(totalPrice>100) {
    if(member == 1) {
       if(type == 'E')
           discount = (0.25)*totalPrice;
        if(type == 'G') {
          //Fault 2 - correct : discount = (0.10)*totalPrice;
           discount = (0.07)*totalPrice;
        }
     }
    else
       discount = (0.05)*totalPrice;
   }
   //Fault 3 – correct : totalPrice = totalPrice-discount;
   totalPrice = totalPrice+discount;
   return totalPrice;
}
```

Table II.. The Input Model

Parameter	Possible Values
totalPrice	{10, 500, 1500}
member	{0, 1}
type	{E, G}

Table III.. The Exhaustive Test Set

test: (totalPrice, member, type)	test: (totalPrice, member, type)
T1: (10, 0, E)	T7: (500, 1, E)
T2: (10, 0, G)	T8: (500, 1, G)
T3: (10, 1, E)	T9: (1500, 0, E)
T4: (10, 1, G)	T10: (1500, 0, G)
T5: (500, 0, E)	T11: (1500, 1, E)
T6: (500, 0, G)	T12: (1500, 1, G)

Example : Accessibility

Table I The Example Program, Test Execution Traces and Results

Program Statements		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
1.	float applyDiscount(int totalPrice,int member, char type)	1	1	1	1	1	1	1	1	1	1	1	1
2.	{	1	1	1	1	1	1	1	1	1	1	1	1
3.	float discount = 0.00;	1	1	1	1	1	1	1	1	1	1	1	1
4.	**//Fault 1 - correct :if(totalPrice>1000)**	-	-	-	-	-	-	-	-	-	-	-	-
5.	**if(totalPrice>100)** {	1	1	1	1	1	1	1	1	1	1	1	1
6.	if(member == 1) {	0	0	0	0	1	1	1	1	1	1	1	1
7.	if(type == 'E')	0	0	0	0	0	0	1	1	0	0	1	1
8.	discount = (0.25)*totalPrice;	0	0	0	0	0	0	1	0	0	0	1	0
9.	if(type == 'G') {	0	0	0	0	0	0	1	1	0	0	1	1
10.	**//Fault 2 - correct : discount = (0.10)*totalPrice;**	-	-	-	-	-	-	-	-	-	-	-	-
11.	**discount = (0.07)*totalPrice;**	0	0	0	0	0	0	0	1	0	0	0	1
12.	}	0	0	0	0	0	0	0	0	0	0	0	0
13.	}	0	0	0	0	0	0	0	0	0	0	0	0
14.	else	0	0	0	0	1	1	0	0	1	1	0	0
15.	discount = (0.05)*totalPrice;	0	0	0	0	1	1	0	0	1	1	0	0
16.	}	0	0	0	0	0	0	0	0	0	0	0	0
17.	**//Fault 3 – correct : totalPrice = totalPrice-discount;**	-	-	-	-	-	-	-	-	-	-	-	-
18.	**totalPrice = totalPrice+discount;**	1	1	1	1	1	1	1	1	1	1	1	1
19.	return totalPrice;	1	1	1	1	1	1	1	1	1	1	1	1
20.	}	0	0	0	0	0	0	0	0	0	0	0	0
	Test result	Pass	Pass	Pass	Pass	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail

Example : Input value sensitivity

Table I The Example Program, Test Execution Traces and Results

Program Statements		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
1.	float applyDiscount(int totalPrice,int member, char type)	1	1	1	1	1	1	1	1	1	1	1	1
2.	{	1	1	1	1	1	1	1	1	1	1	1	1
3.	float discount = 0.00;	1	1	1	1	1	1	1	1	1	1	1	1
4.	**//Fault 1 - correct :if(totalPrice>1000)**	-	-	-	-	-	-	-	-	-	-	-	-
5.	**if(totalPrice>100)** {	1	1	1	1	1	1	1	1	1	1	1	1
6.	if(member == 1) {	0	0	0	0	1	1	1	1	1	1	1	1
7.	if(type == 'E')	0	0	0	0	0	0	1	1	0	0	1	1
8.	discount = (0.25)*totalPrice;	0	0	0	0	0	0	1	0	0	0	1	0
9.	if(type == 'G') {	0	0	0	0	0	0	1	1	0	0	1	1
10.	**//Fault 2 - correct : discount = (0.10)*totalPrice;**	-	-	-	-	-	-	-	-	-	-	-	-
11.	**discount = (0.07)*totalPrice;**	0	0	0	0	0	0	0	1	0	0	0	1
12.	}	0	0	0	0	0	0	0	0	0	0	0	0
13.	}	0	0	0	0	0	0	0	0	0	0	0	0
14.	else	0	0	0	0	1	1	0	0	1	1	0	0
15.	discount = (0.05)*totalPrice;	0	0	0	0	1	1	0	0	1	1	0	0
16.	}	0	0	0	0	0	0	0	0	0	0	0	0
17.	**//Fault 3 – correct : totalPrice = totalPrice-discount;**	-	-	-	-	-	-	-	-	-	-	-	-
18.	**totalPrice = totalPrice+discount;**	1	1	1	1	1	1	1	1	1	1	1	1
19.	return totalPrice;	1	1	1	1	1	1	1	1	1	1	1	1
20.	}	0	0	0	0	0	0	0	0	0	0	0	0
	Test result	Pass	Pass	Pass	Pass	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail

Example : Control flow sensitivity

Table IV. The Error-Free Version of the Example Program, Test Execution Traces and Results

Program Statements		T5	T6	T7	T8	T9	T10	T11	T12
1.	float applyDiscount(int totalPrice,int member, char type)	1	1	1	1	1	1	1	1
2.	{	1	1	1	1	1	1	1	1
3.	float discount = 0.00;	1	1	1	1	1	1	1	1
4.	**//Fault 1 - correct :if(totalPrice>1000)**	-	-	-	-	-	-	-	-
5.	**if(totalPrice>1000) {**	1	1	1	1	1	1	1	1
6.	if(member == 1) {	0	0	0	0	1	1	1	1
7.	if(type == 'E')	0	0	0	0	0	0	1	1
8.	discount = (0.25)*totalPrice;	0	0	0	0	0	0	1	0
9.	if(type == 'G') {	0	0	0	0	0	0	1	1
10.	**//Fault 2 - correct : discount = (0.10)*totalPrice;**	-	-	-	-	-	-	-	-
11.	**discount = (0.10)*totalPrice;**	0	0	0	0	0	0	0	1
12.	}	0	0	0	0	0	0	0	0
13.	}	0	0	0	0	0	0	0	0
14.	else	0	0	0	0	1	1	0	0
15.	discount = (0.05)*totalPrice;	0	0	0	0	1	1	0	0
16.	}	0	0	0	0	0	0	0	0
17.	**//Fault 3 – correct : totalPrice = totalPrice-discount;**	-	-	-	-	-	-	-	-
18.	**totalPrice = totalPrice-discount;**	1	1	1	1	1	1	1	1
19.	return totalPrice;	1	1	1	1	1	1	1	1
20.	}	0	0	0	0	0	0	0	0
	Test result	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass

Example: Control flow sensitivity

Table I The Example Program, Test Execution Traces and Results (Faulty version)

Program Statements		T5	T6	T7	T8	T9	T10	T11	T12
1.	float applyDiscount(int totalPrice,int member, char type)	1	1	1	1	1	1	1	1
2.	{	1	1	1	1	1	1	1	1
3.	float discount = 0.00;	1	1	1	1	1	1	1	1
4.	**//Fault 1 - correct :if(totalPrice>1000)**	-	-	-	-	-	-	-	-
5.	**if(totalPrice>100)** {	1	1	1	1	1	1	1	1
6.	if(member == 1) {	1	1	1	1	1	1	1	1
7.	if(type == 'E')	0	0	1	1	0	0	1	1
8.	discount = (0.25)*totalPrice;	0	0	1	0	0	0	1	0
9.	if(type == 'G') {	0	0	1	1	0	0	1	1
10.	**//Fault 2 - correct : discount = (0.10)*totalPrice;**	-	-	-	-	-	-	-	-
11.	**discount = (0.07)*totalPrice;**	0	0	0	1	0	0	0	1
12.	}	0	0	0	0	0	0	0	0
13.	}	0	0	0	0	0	0	0	0
14.	else	1	1	0	0	1	1	0	0
15.	discount = (0.05)*totalPrice;	1	1	0	0	1	1	0	0
16.	}	0	0	0	0	0	0	0	0
17.	**//Fault 3 – correct : totalPrice = totalPrice-discount;**	-	-	-	-	-	-	-	-
18.	**totalPrice = totalPrice+discount;**	1	1	1	1	1	1	1	1
19.	return totalPrice;	1	1	1	1	1	1	1	1
20.	}	0	0	0	0	0	0	0	0
	Test result	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail

Outline

- Introduction
- Three Fault Properties
- Experimental Design
- Experimental Results
- Conclusion

Subjects: Siemens suite

	Subject Programs	# of lines of executable code	# of faulty versions	Model	Constraints
Siemens suite	printtokens	188	7	$(2^1 \times 3^1 \times 4^4 \times 5^1 \times 10^1 \times 13^2)$	8
	printtokens2	201	10		
	replace	242	32	$(2^4 \times 4^{16})$	36
	schedule	154	9	$(2^1 \times 3^8 \times 8^2)$	0
	schedule2	127	10		
	tcas	65	41	$(2^7 \times 3^2 \times 4^1 \times 10^2)$	0
	totinfo	123	23	$(3^3 \times 5^2 \times 6^1)$	0

Subjects: GREP

Subject Programs		# of lines of executable code	# of faulty versions	Model	Constraints
GREP	grep 1	3078	18	$(2^7 \times 4^1 \times 5^1 \times 6^3 \times 8^1 \times 9^1 \times 1^{31})$	1
	grep 2	3224	8		
	grep 3	3294	18		
	grep 4	3313	12		
	grep 5	3314	1		

Subjects: GZIP

Subject Programs		# of lines of executable code	# of faulty versions	Model	Constraints
GZIP	gzip 1	1705	16	$(2^{11} \times 4^{2})$	8
	gzip 2	2006	7		
	gzip 3	1866	10		
	gzip 4	1892	12		
	gzip 5	1993	14		

Fault localization results

Programs		# of faulty versions	# of killed versions
Siemens suite	printtokens	7	3
	printtokens2	10	9
	replace	32	32
	schedule	9	7
	schedule2	10	3
	tcas	41	36
	totinfo	23	12
GREP	grep1	18	3
	grep3	18	4
	grep4	12	2
GZIP	gzip1	16	6
	gzip2	7	3
	gzip4	12	1
	gzip5	14	3

Measurement of fault properties

- Randomly generate a set of 1000 tests
- Record the program execution trace using GCOV
- High accessibility faults: accessibility score>=0.50; Low accessibility faults: accessibility score< 0.50

Outline

- Introduction
- Three Fault Properties
- Experimental Design
- Experimental Results
- Conclusion

Impact of accessibility

Group	Input value Sensitivity	Control flow sensitivity	Accessibility	# of faults	Average % of code
1	Y	Y	**H**	56	20.93
	Y	Y	**L**	41	10.18
2	Y	N	**H**	3	29.51
	Y	N	**L**	2	3.66
3	N	Y	**H**	2	10.57
	N	Y	**L**	16	4.27
4	N	N	**H**	0	NA
	N	N	**L**	3	5.96

Impact of input value sensitivity

Group	Input value Sensitivity	Control flow sensitivity	Accessibility	# of faults	Average % of code
1	**Y**	Y	H	56	20.93
	N	Y	H	3	10.57
2	**Y**	Y	L	41	10.18
	N	Y	L	16	4.27
3	**Y**	N	H	3	29.51
	N	N	H	0	NA
4	**Y**	N	L	2	3.66
	N	N	L	3	5.96

Impact of control flow sensitivity

Group	Input value Sensitivity	Control flow sensitivity	Accessibility	# of faults	Average % of code
1	Y	**Y**	H	56	20.93
	Y	**N**	H	3	29.51
2	Y	**Y**	L	41	10.18
	Y	**N**	L	2	3.66
3	N	**Y**	H	3	10.57
	N	**N**	H	0	NA
4	N	**Y**	L	16	4.27
	N	**N**	L	3	5.96

Outline

- Introduction
- Three Fault Properties
- Experimental Design
- Experimental Results
- Conclusion

Conclusion

- Investigate the impact of three fault properties on the effectiveness of BEN
- A random test set-based approach was followed to determine the three fault properties.
- BEN is very effective in localizing
  - low accessibility faults
  - input value-insensitive (or control flow-insensitive) faults than input value-sensitive (or control flow-sensitive) faults

Future work

- Evaluate the impact of high accessibility, input value and control flow insensitive faults
- Use scalar measures for input value and control flow sensitivity and analyze the correlation
- Create different types of faults using a mutation tool and evaluate their impact

References

1. A. Bandyopadhyay, S. Ghosh. On the Effectiveness of the Tarantula Fault Localization Technique for Different Fault Classes. Proceedings of 13th International Symposium on High-Assurance Systems Engineering (HASE), 317-324, 2011.
2. L.S.Ghandehari, Y.Lei, D.Kung, R.Kacher and R.Kuhn. Fault localization based on failure-inducing combinations. Proceedings of IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), 168–177, 2013
3. L.S.Ghandehari, Y.Lei, T.Xie, R.Kuhn and R.Kacker. Identifying failure-inducing combinations in a combinatorial test set. Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST). 370-379,2012
4. L.S.Ghandehari, Y.Lei, R. Kacker and R.Kuhn. A Combinatorial testing based approach to fault localization. [under preparation]