On Reducibility of Developer-Written Unit Tests in C#

Authors: Arpit Christi, David Weber

Presenter: Arpit Christi, School of Computing, Weber State University

Email: arpitchristi@weber.edu

Author Bio

Arpit Christi

- Assistant Professor at Weber State University
- PhD Computer Science – Oregon State University
- Research Interests – Program Debugging, Self Adaptive Software, Software Testing

David Weber

- Embedded Software Engineer at Northrop Grumman
- MS Computer Science – Weber State University
- Research Interests – Adaptive Programming, High Performance Computing, Embedded Systems

Introduction

- Program Debugging and Test Case Reduction
- Background and Motivation
- Reduction Process and Outcome
- Experiments and Results
- Conclusion and Future Work

Program Debugging

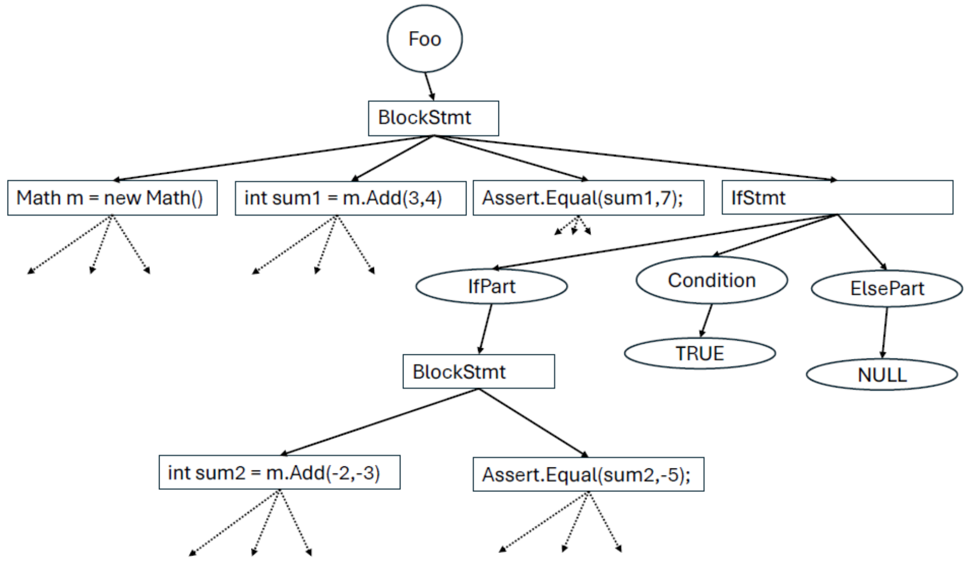
- Debugging is difficult and time consuming.
- Developer time is spent on locating the fault.
- Test case reduction is useful.
- Reduces the test while keeping failure-inducing input. [1]
- Entities not important to inducing the failure are removed.
- Keep developer focus on faulty aspects of the program.
- Test reduction improves Automatic Fault Localization [2,3]

Background and Motivation

- Delta Debugging [1]
  - Test reduction for flat, array or list like structures
- Hierarchical Delta Debugging [4]
  - Test reduction for hierarchical and tree like structures
- Modern Tools, Techniques and Algorithms
  - Mostly use DD, HDD algorithms
  - Slow algorithms: DD ($O(n^2)$), HDD ($O(n^3)$)
  - All elements are processed indiscriminately
  - No priority is assigned to an element based on category
  - What are the elements?: Can ne characters, words, lines of a test

Tests written as program and Abstract Syntax Tree

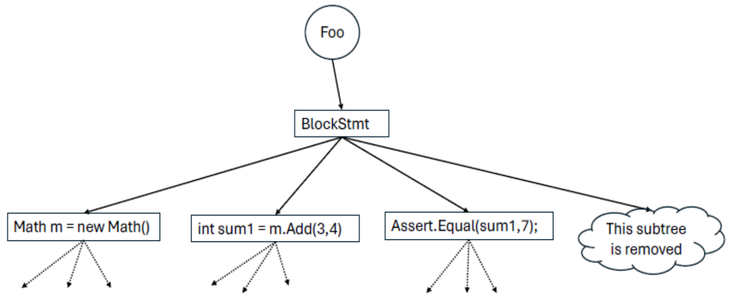
```
[Fact]
public void Foo(Test)
{
  Math m = new Math();
  int sum1 = m.Add(3,4)
  // Assumption: Add method is written in a
  // peculiar way and cannot add 3 and 4
  // correctly.
  Assert.Equal(sum1,7); //suppose sum1 is 8,
   hence the test is failing here.
   if(true){
      int sum2 = m.Add(-2,-3)
     Assert.Equal(sum2,-5); // This assert
   passes.
     }
}
```



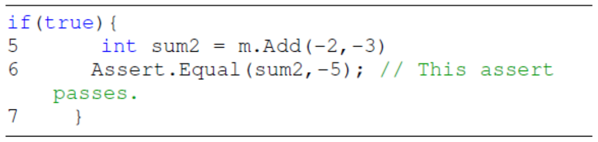
- Considering characters or words or lines of a program as reduction unit can produce non compilable results
- Nodes are the reduction units or elements – reduces the possibility of non compilable test
- Only statement level nodes are considered for reduction

Reduction Process and Outcome

- Outcome



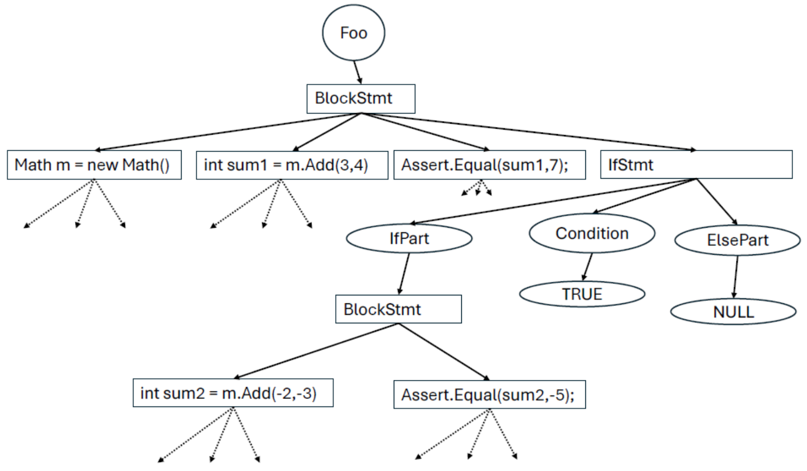
- Reduced entities
- Three statements are reduced
- One IfStmt
- Two other statements
  - Int sum2 = m.Add(-2,-3)
  - Assert.Equal(sum2,-5)



```
if(true){
     int sum2 = m.Add(-2,-3)
    Assert.Equal(sum2,-5); // This assert
  passes.
   }
```

Categorizing statements

- Based on the location of a statement within the AST
- Previously researchers found heuristics on program reduction based on similar categorization [6]
- Tree Statement – A statement that has one or more statement nodes below it.
- NonTree Statement – A statement that has no statement node below it.



Research Questions

- RQ1: What kind of statements are reduced and in what numbers? Based on the category.
- RQ2: What is the probability of a reduction of a statement based on the category?

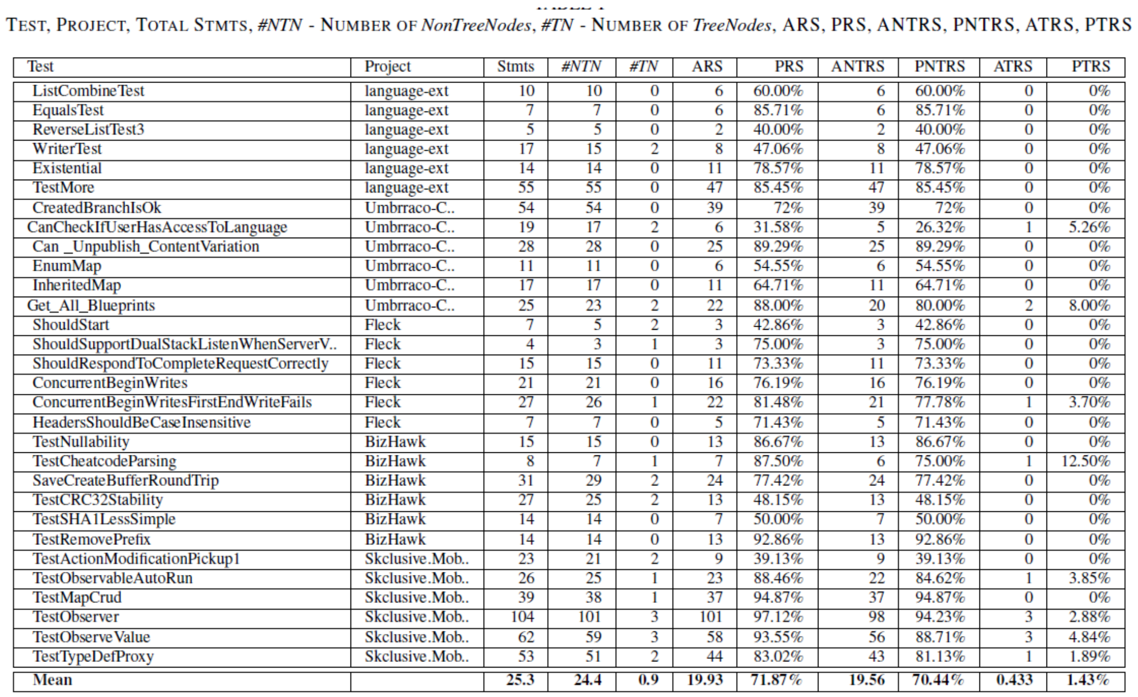
Experiment - Subjects

- 30 real-world bugs across five open-source projects, each bug has a failing test [5]
- Process 759 statements for failing tests
- 732 non tree statements and 27 tree statements
- Each test is reduced using ReduSharptor [5]
- Are we reducing tests accurately?
  - Redusharptor has 96% precision and 96% recall

Experiment - Measurement

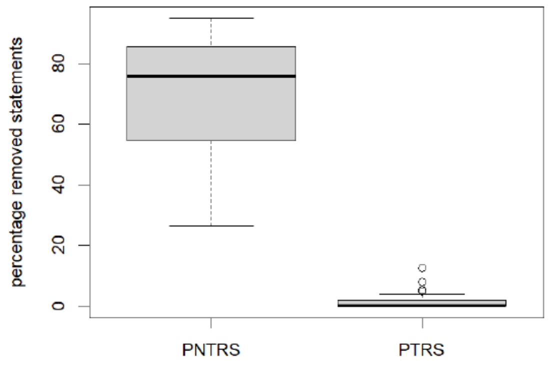
- ARS (Absolute Reduction Size): The number of statements reduced.
- PRS (Percentage Reduction Size): The percent of total statements reduced.
- ATRS (Absolute Tree Statement Reduction Size): The number of tree statements reduced.
- PTRS (Percent Tree Statement Reduction Size): The percentage of tree statements reduced.
- ANTRS and PNTRS: Same as ATRS and PTRS but for non tree statements.
- Percentage numbers are more meaningful than absolute numbers.

Results: RQ1



TEST, PROJECT, TOTAL STMTS, *#NTN* - NUMBER OF *NonTreeNodes*, *#TN* - NUMBER OF *TreeNodes*, ARS, PRS, ANTRS, PNTRS, ATRS, PTRS

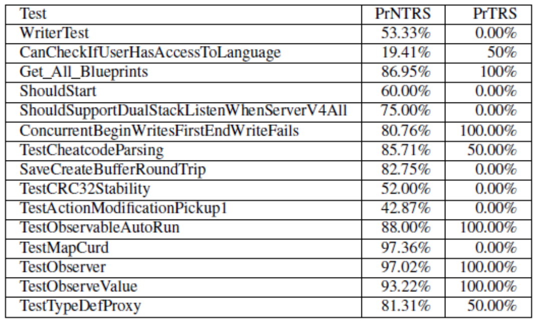
Test	Project	Stmts	*#NTN*	*#TN*	ARS	PRS	ANTRS	PNTRS	ATRS	PTRS
ListCombineTest	language-ext	10	10	0	6	60.00%	6	60.00%	0	0%
EqualsTest	language-ext	7	7	0	6	85.71%	6	85.71%	0	0%
ReverseListTest3	language-ext	5	5	0	2	40.00%	2	40.00%	0	0%
WriterTest	language-ext	17	15	2	8	47.06%	8	47.06%	0	0%
Existential	language-ext	14	14	0	11	78.57%	11	78.57%	0	0%
TestMore	language-ext	55	55	0	47	85.45%	47	85.45%	0	0%
CreatedBranchIsOk	Umbrraco-C..	54	54	0	39	72%	39	72%	0	0%
CanCheckIfUserHasAccessToLanguage	Umbrraco-C..	19	17	2	6	31.58%	5	26.32%	1	5.26%
Can _Unpublish_ContentVariation	Umbrraco-C..	28	28	0	25	89.29%	25	89.29%	0	0%
EnumMap	Umbrraco-C..	11	11	0	6	54.55%	6	54.55%	0	0%
InheritedMap	Umbrraco-C..	17	17	0	11	64.71%	11	64.71%	0	0%
Get_All_Blueprints	Umbrraco-C..	25	23	2	22	88.00%	20	80.00%	2	8.00%
ShouldStart	Fleck	7	5	2	3	42.86%	3	42.86%	0	0%
ShouldSupportDualStackListenWhenServerV..	Fleck	4	3	1	3	75.00%	3	75.00%	0	0%
ShouldRespondToCompleteRequestCorrectly	Fleck	15	15	0	11	73.33%	11	73.33%	0	0%
ConcurrentBeginWrites	Fleck	21	21	0	16	76.19%	16	76.19%	0	0%
ConcurrentBeginWritesFirstEndWriteFails	Fleck	27	26	1	22	81.48%	21	77.78%	1	3.70%
HeadersShouldBeCaseInsensitive	Fleck	7	7	0	5	71.43%	5	71.43%	0	0%
TestNullability	BizHawk	15	15	0	13	86.67%	13	86.67%	0	0%
TestCheatcodeParsing	BizHawk	8	7	1	7	87.50%	6	75.00%	1	12.50%
SaveCreateBufferRoundTrip	BizHawk	31	29	2	24	77.42%	24	77.42%	0	0%
TestCRC32Stability	BizHawk	27	25	2	13	48.15%	13	48.15%	0	0%
TestSHA1LessSimple	BizHawk	14	14	0	7	50.00%	7	50.00%	0	0%
TestRemovePrefix	BizHawk	14	14	0	13	92.86%	13	92.86%	0	0%
TestActionModificationPickup1	Skclusive.Mob..	23	21	2	9	39.13%	9	39.13%	0	0%
TestObservableAutoRun	Skclusive.Mob..	26	25	1	23	88.46%	22	84.62%	1	3.85%
TestMapCrud	Skclusive.Mob..	39	38	1	37	94.87%	37	94.87%	0	0%
TestObserver	Skclusive.Mob..	104	101	3	101	97.12%	98	94.23%	3	2.88%
TestObserveValue	Skclusive.Mob..	62	59	3	58	93.55%	56	88.71%	3	4.84%
TestTypeDefProxy	Skclusive.Mob..	53	51	2	44	83.02%	43	81.13%	1	1.89%
Mean		**25.3**	**24.4**	**0.9**	**19.93**	**71.87%**	**19.56**	**70.44%**	**0.433**	**1.43%**



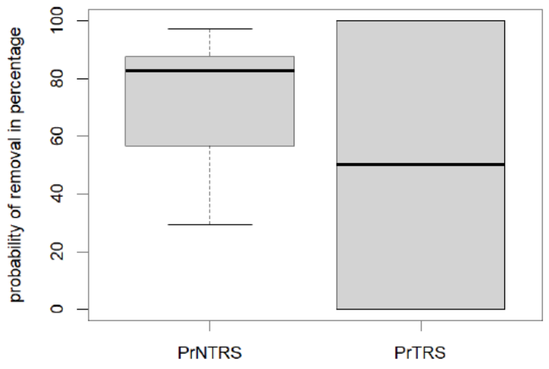
- Non tree nodes are reduced in large numbers.
- Wilcoxon signed rank test on PNTRS vs PTRS: p value < 0.0005 and V=465
- The boxplot suggests that non tree nodes are reduced approximately 50 times more.

Results: RQ2

- The number of tree statements are less in numbers than non tree statements.
- PrNTRS: Probability of removal of a non tree statement (ANTRS / NTN) * 100
- PrTRS: Probabiity of removal of a tree statement (ATRS/TN) * 100
- Contains undefined results due to TN=0 for a few tests. Those results are neglected for further evaluations.



Test	PrNTRS	PrTRS
WriterTest	53.33%	0.00%
CanCheckIfUserHasAccessToLanguage	19.41%	50%
Get_All_Blueprints	86.95%	100%
ShouldStart	60.00%	0.00%
ShouldSupportDualStackListenWhenServerV4All	75.00%	0.00%
ConcurrentBeginWritesFirstEndWriteFails	80.76%	100.00%
TestCheatcodeParsing	85.71%	50.00%
SaveCreateBufferRoundTrip	82.75%	0.00%
TestCRC32Stability	52.00%	0.00%
TestActionModificationPickup1	42.87%	0.00%
TestObservableAutoRun	88.00%	100.00%
TestMapCurd	97.36%	0.00%
TestObserver	97.02%	100.00%
TestObserveValue	93.22%	100.00%
TestTypeDefProxy	81.31%	50.00%



- Wilcoxon Signed Rank Test on PrNTRS vs PrTRS: p-value < 0.05 and V= 99.
- Boxplot of PrNTRS and PrTRS suggests that probability of removal of a non tree statement is 1.7 times higher than that of tree statement

Conclusion and Future Work

- DD/HDD implementations don’t assign priority to an entity based on the category.
- We came up with broad generic category for tests written as program (1) Tree statement (2) Non tree statement.
- We study the effect of a statement category on reduction outcome and removal process.
- We conclude (1) non tree statements are removed in larger numbers (2) non tree statements have slightly higher chance of removal.
- Extend the work for tests written in other programming languages.
- Extend the work by defining other categories.
- Extend the work for test inputs not written as programs.

Refernces

[1] A. Zeller and R. Hildebrandt, “Simplifying and isolating failure-inducing input,” IEEE Trans. Softw. Eng., vol. 28, no. 2, pp. 183–200, Feb. 2002.

[2] A. Christi, M. L. Olson, M. A. Alipour, and A. Groce, “Reduce beforeyou localize: Delta-debugging and spectrum-based fault localization,” in 2018 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Memphis, TN, USA, October 15-18,2018, pp. 184–191.

[3] D. Vince, R. Hodova´n, and A´ . Kiss, “Reduction-assisted fault localization: Don’t throw away the by-products!” in ICSOFT, 2021, pp. 196–206.

[4] G. Misherghi and Z. Su, “HDD: Hierarchical delta debugging,” in Proceedings of the 28th International Conference on Software Engineering, ser. ICSE ’06, 2006, pp. 142–151.

[5] D. Weber and A. Christi, “Redusharptor: A tool to simplify developer-written c# unit tests,” International Journal of Software Engineering & Applications, vol. 14, pp. 29–40, 09 2023.

[6] A. Christi and A. Groce, “Target selection for test-based resource adaptation,” in 2018 IEEE International Conference on Software Quality,Reliability and Security (QRS), July 2018, pp. 458–469.