Is Mutation an Appropriate Tool for Testing Experiments?

J.H.Andrews, L.C.Briand, Y.Labiche

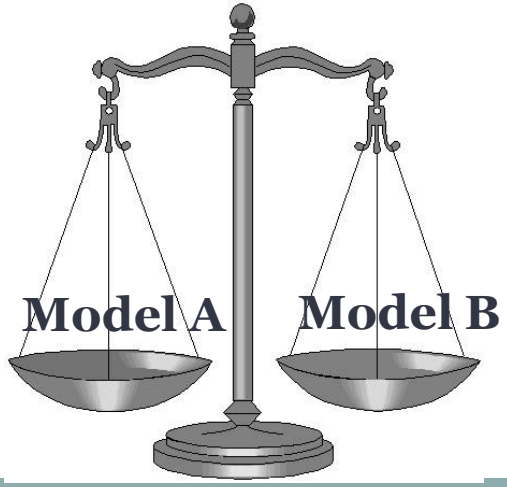
CMSC737

Srividya Ramaswamy

December 1, 2009

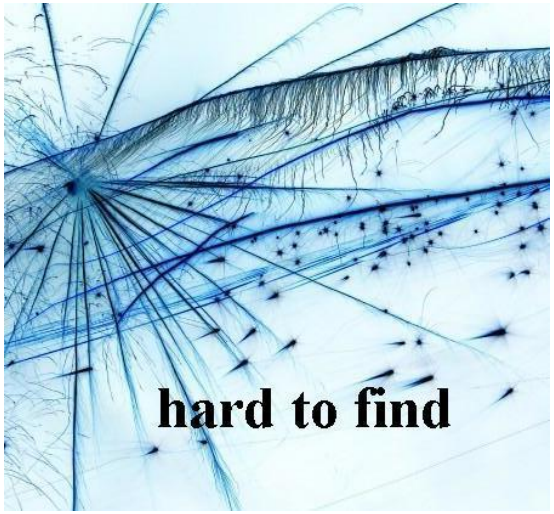
Experimentation

- Essential part of research in software testing
- Determine which model/technique is superior
- Require realistic subject programs



Realistic Programs

- Must have appropriate size
- Significant number of real faults



Problem

- Too hard to find such programs
- Often faults are not numerous enough

Solution

- Introduce faults by hand
  - Realistic
  - Difficult to replicate
- Generate variants of the code automatically
  - Various operators available to produce these variants
  - Well defined fault seeding process
  - Easy to replicate

Mutation operator:
Generates variants of code

Mutant:
Faulty version of code generated by a mutation operator

Mutation:
Process of mutant generation

Is the ability to detect mutants an accurate predictor of actual performance?

The paper has tried to answer this question

Related Work

Usage of mutants to measure test suite adequacy
- *DeMillo et al, Hamlet*

Test data that detects simple faults will detect complex faults
- *Offutt*

Method for generating faulty versions for experiments
- *Memon et al, Kim et al, …*

The Experiment

- Analyze the detection rates of test suites
- Eight subject programs
- Large comparable pools of test cases
  - Satisfying several structural criteria
- Test suites formed by random sampling of test pool
- Create mutant versions of those programs
- Execute test suites on faults and mutants
- Compare fault detection ratio

Null Hypothesis (H_0):

There is no difference in detection ratios between sets of faults and sets of mutants

Alternative Hypothesis (H_a):

There is a difference in detection ratios between sets of faults and sets of mutants

Subject programs

- Eight well known programs written in C

- Space
  - Developed at European Space Agency
  - Real faults
  - 38 faulty versions

- Siemens suite of programs
  - First used to compare control flow-based and data-flow based coverage criteria
  - Hand seeded faults
  - Faults seeded by eight different people

- Programs chosen due to historical significance

- Black-box testing techniques and structural test coverage criteria used to create test pool

Table 1. Description of Subject Programs

	Subject Programs							
Criterion	Space	Printtokens	Printtokens2	Replace	Schedule	Schedule2	Tcas	Totinfo
NLOC	5905	343	355	513	296	263	137	281
# conditionals	635	94	81	99	37	51	25	46
Test Pool Size (# test cases)	13585	4130	4115	5542	2650	2710	1608	1052
Number of faults (Versions)	38	7	10	32	9	10	41	23
Number of compiled mutants	11379	582	375	666	253	299	291	516

NLOC – Net lines of code
conditionals – Number of C conditional constructs and binary logical operators

Mutant Operators

- Each line considered in sequence
- Each of the four classes of “mutant operators” applied (whenever possible)
- “Sufficient” mutant operators

- Replace an integer constant C by 0, 1, -1, ((C) + 1), or ((C) – 1)
- Replace an arithmetic, relational, logical, bitwise logical, increment/decrement, or arithmetic-assignment operator by another
- Negate the decision in an *if* or *while* statement
- Delete a statement

Mutant Operators (contd..)

- Generate mutants using these operators
- 8.4% of resulting mutants did not compile
- Too many mutants generated for the Space program
- Test suite ran on every 10^{th} mutant generated
- Random selection of 10% of the mutants
- Whole source code seeded with faults

Analysis procedure

- Generate and compile the mutants
- 5000 test suites randomly formed by sampling the available pool
- Obtain sample distributions of fault/mutant detection rates that would approximate well the underlying theoretical distributions
- Random selection deemed to provide good variability as compared to a selection driven by coverage criteria
- Size of each test suite taken as 100

Analysis procedure (contd..)

For each test suite S, we calculate the following:

Dm(S) - Number of mutants detected by S

Df(S) - Number of faults detected by S

Nm - Number of non-equivalent mutants

Nf - Number of non-equivalent faults

Am(F) - Mutation detection ratio (Dm(S) / Nm)

Af(S) - Fault detection ratio (Df(S) / Nf)

Analysis procedure (contd..)

- For each program P, check if mean of Am and Af ratios for P were the same
- If the means are different, the next question is why?
- If the results are not consistent across programs, identify most most plausible explanations
  - Differences in characteristics of subject programs?
  - Test suites?
  - The way faults were seeded?

Analysis procedure (contd..)

For each mutant M and faulty version F, we calculate the following:

K(M) - Number of test cases that killed M

K(F) - Number of test cases that killed F

E(M) - Ease of killing M (K(M) / T)

K(F) - Ease of killing F (K(F) / T)

Threats to Validity

- *Internal*:
  - The programs, test pools, and faults used as is
  - No guarantee that test pools have the same detection power and coverage
  - Except Space, others have “realistic” hand seeded faults
  - Programs could be of varying complexity
  - Other mutation operators could produce varying results
  - Fixed size of test suite

- *External*:
  - Relates to our ability to generalize the results of the experiment to industrial practice
  - Only one program with real faults used

Threats to Validity

- *Construct*:
  - Concerns the way we defined our measurement
  - Does it measure the detection power of test sets and detectability of faults?
  - This was justified before

- *Conclusion*:
  - Relates to subject selection, data collection, validity of the statistical tests, and measurement reliability
  - Addressed during the design of the experiment

Analysis Results

- Compare detection distributions of mutants and faults
- Identify possible phenomena which could explain the trends observed from the above step
- Investigate most plausible explanations

Comparing detection distributions of mutants and faults

Table 2. Descriptive Statistics – Detection Ratios for Mutants and Faults (test suite size 100)

Subject Programs								
Statistic	Space	Replace	Printtokens	Printtokens2	Schedule	Schedule2	Tcas	Totinfo
Mutants – Am(S)								
Median	0.75	0.93	0.98	0.99	0.96	0.96	0.91	0.99
Mean	0.75	0.93	0.97	0.99	0.96	0.96	0.90	0.99
90%	0.78	0.95	0.99	0.99	0.98	0.97	0.94	0.99
75%	0.77	0.94	0.98	0.99	0.97	0.97	0.93	0.99
25%	0.74	0.93	0.97	0.98	0.94	0.95	0.89	0.98
10%	0.72	0.92	0.96	0.98	0.93	0.95	0.86	0.98
Min	0.65	0.88	0.94	0.93	0.91	0.91	0.77	0.94
Max	0.82	0.98	1	1	0.99	0.99	0.97	1
Faults – Af(S)								
Median	0.76	0.68	0.57	0.90	0.67	0.67	0.76	0.65
Mean	0.77	0.67	0.63	0.93	0.66	0.63	0.79	0.89
90%	0.82	0.77	0.86	1	0.78	0.78	1	0.96
75%	0.79	0.74	0.71	1	0.78	0.78	0.95	0.91
25%	0.74	0.61	0.57	0.90	0.56	0.56	0.68	0.87
10%	0.71	0.55	0.29	0.80	0.44	0.44	0.61	0.83
Min	0.53	0.35	0.14	0.60	0.33	0	0.34	0.65
Max	0.97	0.93	1	1	1	1	1	1

10%, 25%, 75%, 90% are all quantiles

Comparing detection distributions of mutants and faults

- From Table 2 we can observer that mutants tend to be easier to detect than faults except for Space

- Is the difference in the results between Space and the other programs due to higher difficultly of detectability of seeded faults in the Siemens suite as compared to the real faults in Space?

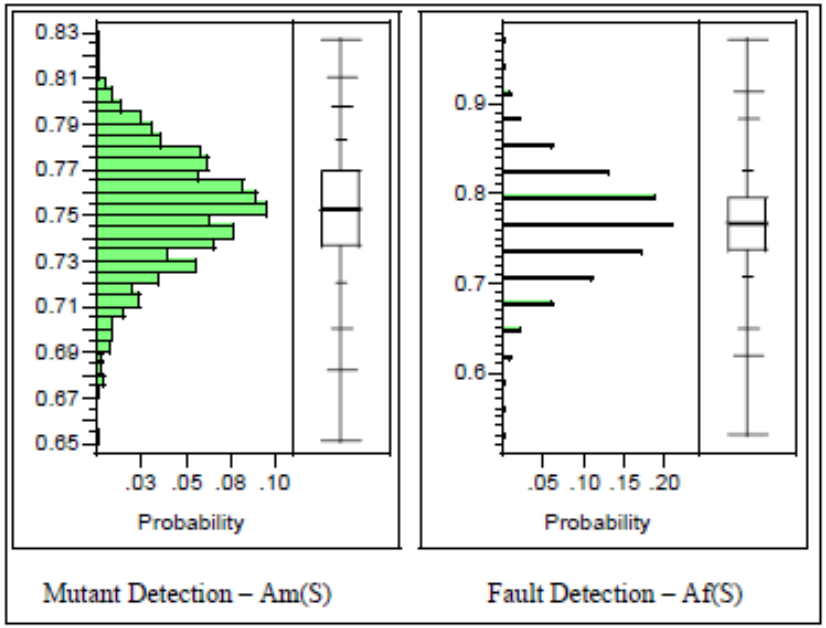


Figure 1. Subject Program Space: Distributions of Detection Ratios – test suite size = 100

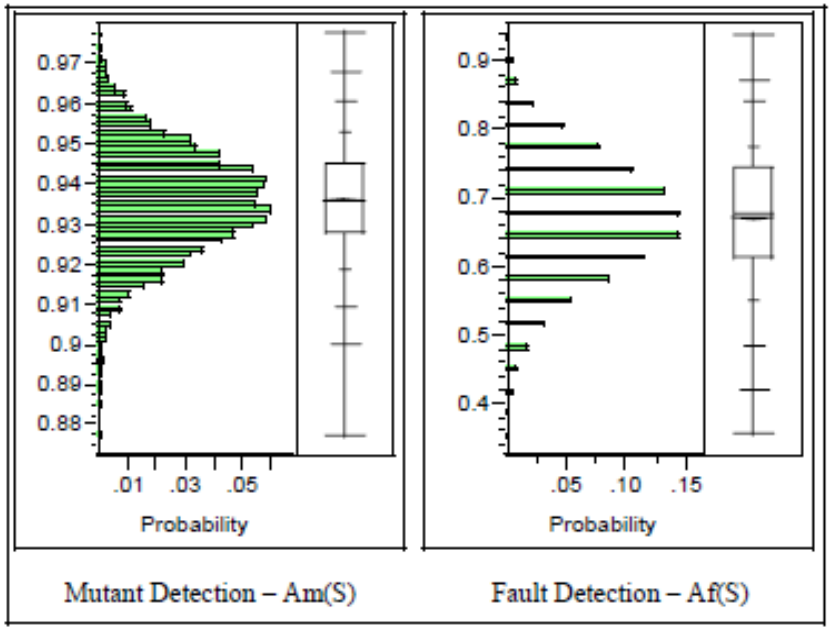


Figure 2. Subject Program Replace: Distributions of Detection Ratios – test suite size = 100

Alternative Explanations

- Relative detectability of real and seeded faults
- Test pool
- Test suite size compared to the size of the test pool and program
- Mutation process
- Difference between Am and Af increases as the test suite size increases compared to the size of the test pool or the size of the program
- Mutation could somehow be biased so that mutants are harder to kill on Space than on the Siemens programs
- Space could have lower mutant detection ratio as larger programs are more difficult to test

One other explanation:

- Space faults easier to detect for any individual test case as compared to the Siemens faults
- Space mutants should not be easier for any individual test case to kill than the Siemens program mutants

Comparing the detectability of faults and mutants across programs

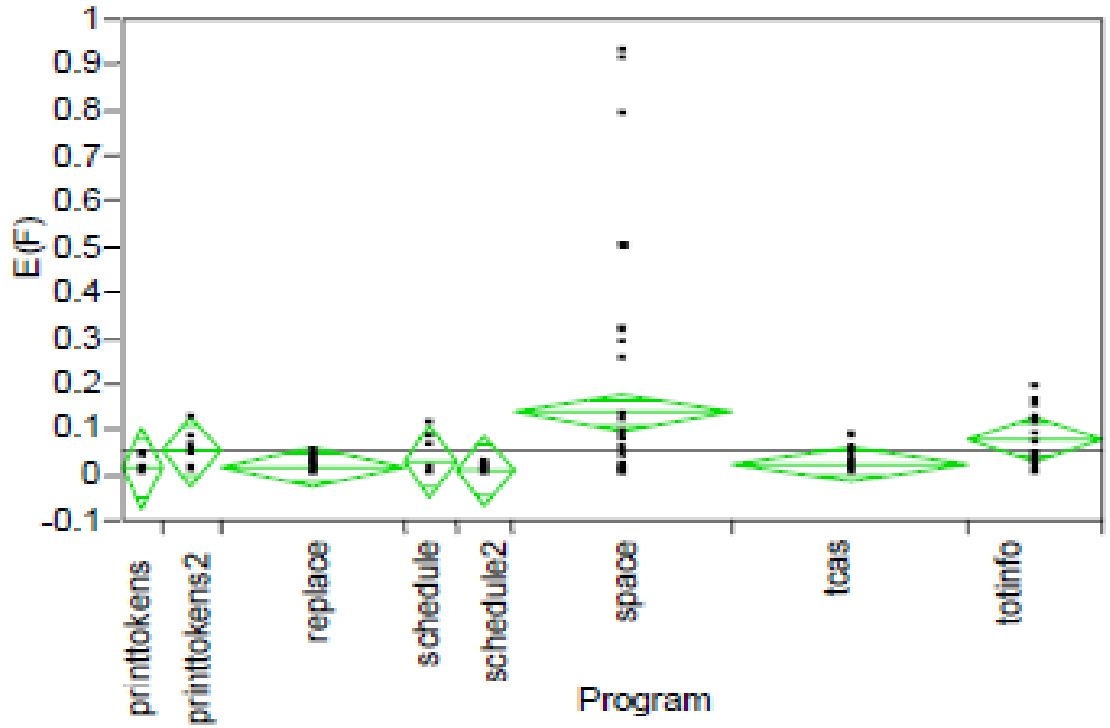
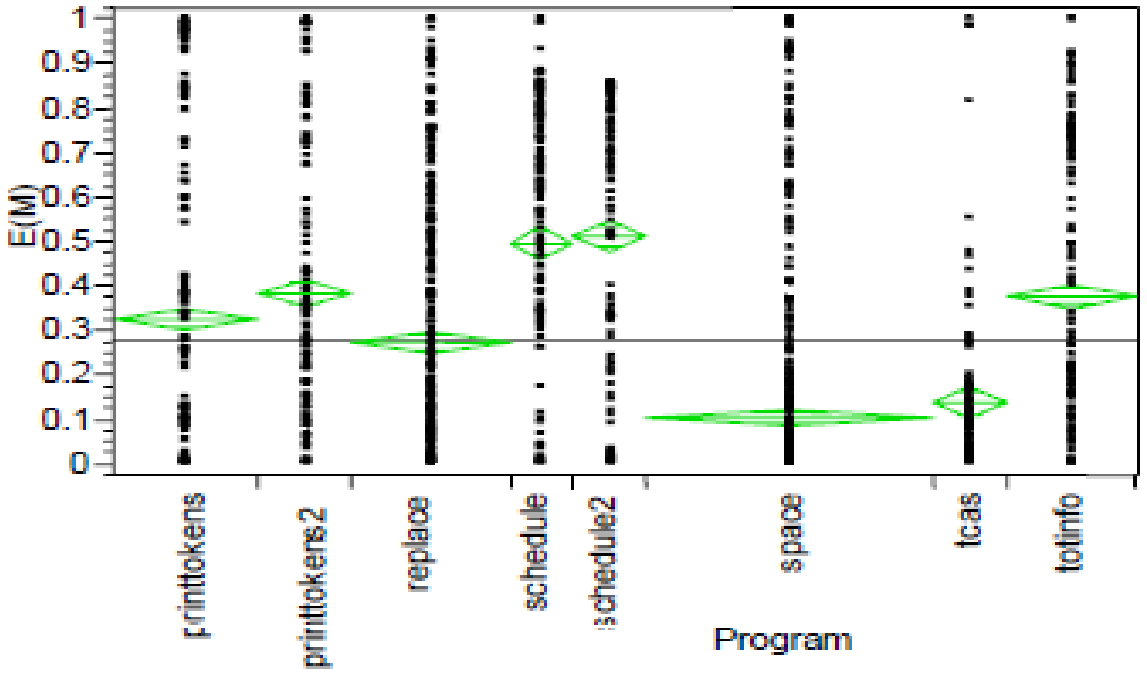


Figure 3. Distribution of Ease of Detection of Faults – E(F)

X-Axis: Subject program

Y-Axis: Computer ratio for each fault

- Horizontal line across all programs – *Overall average*
- Line across the diamond - *Each program specific average*
- The vertical span of the diamond – *The 95% confidence level*



X-Axis: Subject program

Y-Axis: Computer ratio for each fault

Figure 4. Distribution of Ease of Detection of Mutants – E(M)

Discussion

- The faults seeded in the programs other than Space are probably not representative in terms of ease of detection
- Hutchins et al. state that, faults that were detected by 350 or more test cases (from their original test pool) were discarded
- We can conclude that mutants, based on the mutation operators presented here, do provide test effectiveness results that are representative of real faults
- Using human-seeded faults could lead to underestimating the effectiveness of the techniques
- The faults seeded in the Siemens suite are preferable where we may want to concentrate on hard-to-detect faults
- Should be able to create difficult to kill mutants just as faults were selected in Siemens suite

Conclusion

- Mutants can provide a good indication of the fault detection ability of a test suite
- Certain faults, like the ones in the Siemens suite, could lead to under estimation
- Faults made by programmers will be detected by test suites that kill mutants

Future work

- Replicate the study report in this paper
- Perform similar studies in the context of object-oriented systems
- Test suites selected according to various criteria, such as code coverage criteria or operational profile criteria

Questions


THANK
YOU