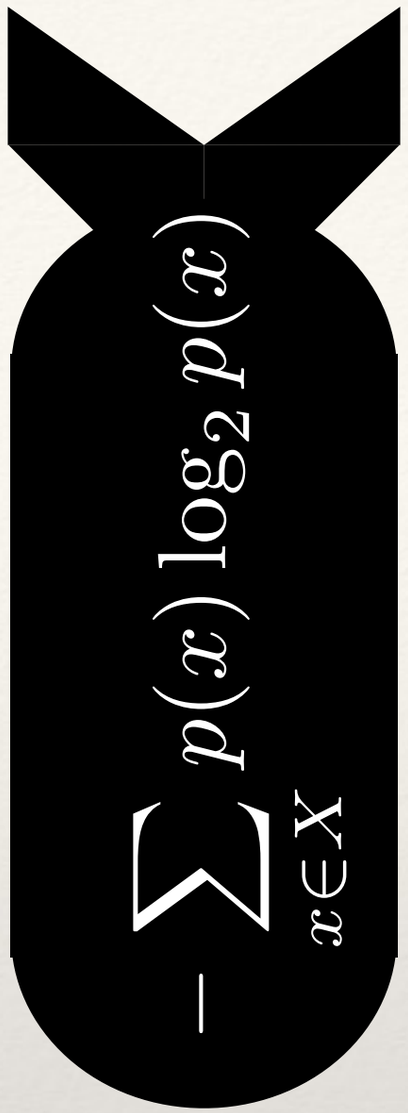


CREST Open Workshop #41

Application of Information Theory to Fault Localisation

or
"How I learnt to stop worrying and love the ~~bomb~~ entropy"
by Shin Yoo

This talk is…

- Not a theoretical masterclass on application of Shannon Entropy to software engineering, unfortunately
- Rather a story of a clueless software engineer who learnt to appreciate the power of information theory

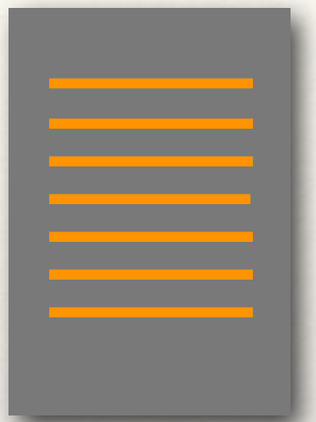
The Problem Domain

- Fault Localisation: given observations from test execution (which includes both passing and failing test cases), identify where the faulty statement lies.

```
160             return ret;
161         }
162
163         private SolutionSet spreadBetIr
164         {
165             SolutionSet pop = new Solut
166             DecisionVariables dv;
167
168             //empty selection
169             dv = new DecisionVariables(
170             for (int k = 0; k < problem
171             {
172                 dv.variables_[k] = new
173             }
174             pop.add(new Solution(proble
175
176             //full selection
177             dv = new DecisionVariables(
178             for (int k = 0; k < problem
179             {
180                 dv.variables_[k] = new
181             }
182             pop.add(new Solution(proble
183
```

Spectra Based Fault Localisation

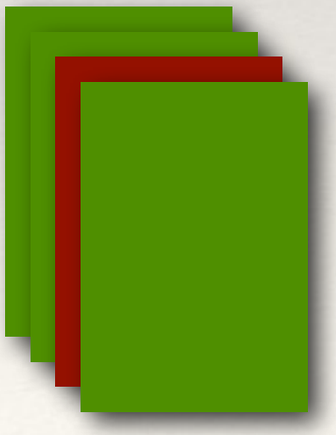
Program



Spectrum

$$e_f - \frac{e_p}{e_p + n_p + 1}$$

Formula (Suspiciousness)



Tests

Higher ranking

=

Fewer statements

to check

Ranking

Spectra-Based Fault Localisation

Structural Elements	Test t_1	Test t_2	Test t_3	Spectrum e_p	e_f	n_p	n_f	Tarantula	Rank
s_1	•			1	0	0	2	0.00	9
s_2	•			1	0	0	2	0.00	9
s_3	•			1	0	0	2	0.00	9
s_4	•			1	0	0	2	0.00	9
s_5	•			1	0	0	2	0.00	9
s_6	•	•		1	1	0	1	0.33	4
s_7 (faulty)	•	•		0	2	1	0	1.00	1
s_8	•	•		1	1	0	1	0.33	4
s_9	•	•	•	1	2	0	0	0.50	2
Result	P	F	F						

$$\text{Tarantula} = \frac{\frac{e_f}{e_f+n_f}}{\frac{e_p}{e_p+n_p} + \frac{e_f}{e_f+n_f}}$$

How do we evaluate these?

$$Wong2 = e_f - e_p$$

$$Op1 = \begin{cases} -1 & \text{if } n_f > 0 \\ n_p & \text{otherwise} \end{cases}$$

$$Op2 = e_f - \frac{e_p}{e_p + n_p + 1}$$

$$\frac{2e_f}{e_f + n_f + e_p}$$

$$\frac{e_f}{e_f + n_p + 2(e_p + n_f)}$$

$$\frac{e_f}{e_f + n_f + e_p}$$

$$Jaccard = \frac{e_f}{e_f + n_f + e_p}$$

$$Tarantula = \frac{\frac{e_f}{e_f + n_f}}{\frac{e_p}{e_p + n_p} + \frac{e_f}{e_f + n_f}}$$

$$\frac{2(e_f + n_p)}{2(e_f + n_p) + e_p + n_f}$$

$$\frac{e_f + n_p}{n_f + e_p}$$

$$AMPLE = |\frac{e_f}{e_f + n_f} - \frac{e_p}{e_p + n_p}|$$

$$\frac{e_f}{n_f + e_p}$$

$$Ochiai = \frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}}$$

$$\frac{e_f}{e_f + n_f + e_p + n_p}$$

$$\frac{e_f + n_p}{e_f + n_f + e_p + n_p}$$

$$Wong1 = e_f$$

$$\frac{2e_f}{2e_f + n_f + e_p}$$

$$\frac{\frac{e_f}{e_f + n_f}}{\frac{e_p}{e_p + n_p} + \frac{e_f}{e_f + n_f}}$$

$$\frac{1}{2}(\frac{e_f}{e_f + n_f} + \frac{e_f}{e_f + e_p})$$

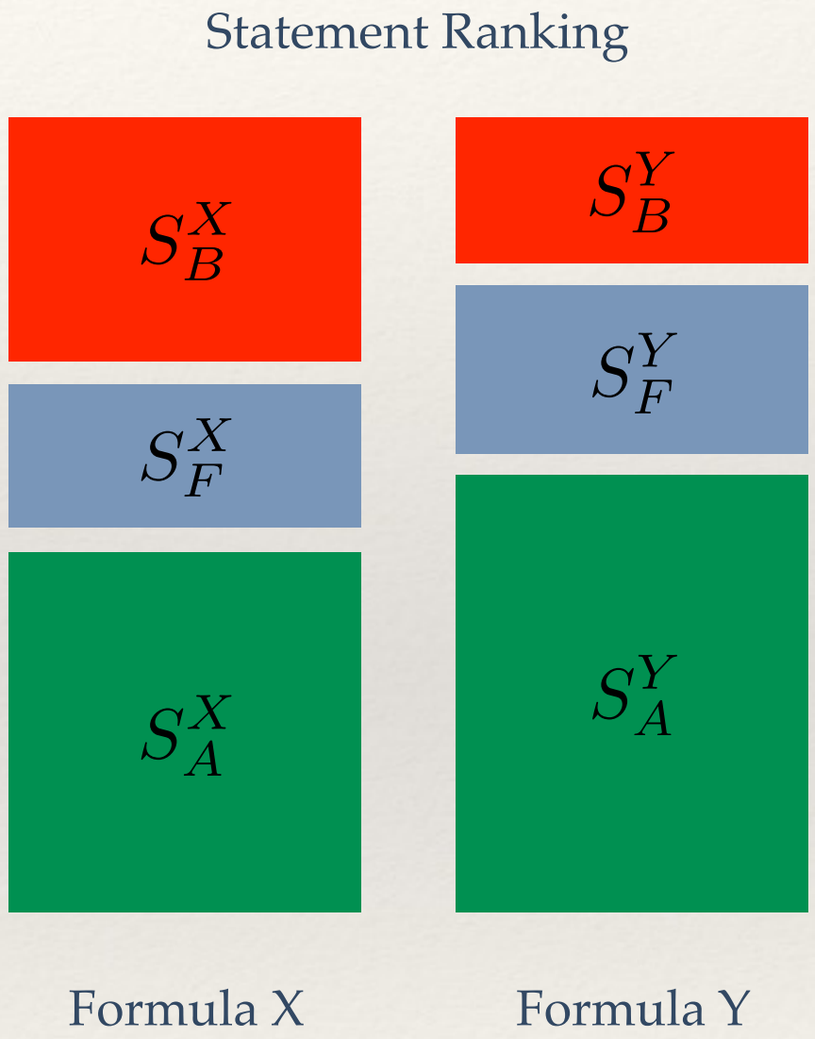
$$\frac{e_f + n_p - n_f - e_p}{e_f + n_f + e_p + n_p}$$

$$Wong3 = e_f - h, h = \begin{cases} e_p & \text{if } e_p \leq 2 \\ 2 + 0.1(e_p - 2) & \text{if } 2 < e_p \leq 10 \\ 2.8 + 0.001(e_p - 10) & \text{if } e_p > 10 \end{cases}$$

Expense Metric

$$E(\tau, p, b) = \frac{\text{Ranking of } b \text{ according to } \tau}{\text{Number of statements in } p} * 100$$

- Assumes that the developer checks the ranking from top to bottom
- The higher the faulty statement is ranked, the earlier the fault is found



Does every test execution help you?

- When a statement is executed by a failing test, we suspect it more; by a passing test, we suspect it less.
- *Ideally*, we want the failing test to only execute the faulty statement, which is not possible of course.
- *Practically*, we want the subset of test runs that gives us the most distinguishing power, and we want this as early as possible.

What is the information gain of executing one more test?

Convert suspiciousness into probability

$$\mathbf{P}_{T_i}(B(s_j)) = \frac{\tau(s_j|T_i)}{\sum_{j=1}^{m} \tau(s_j|T_i)}$$

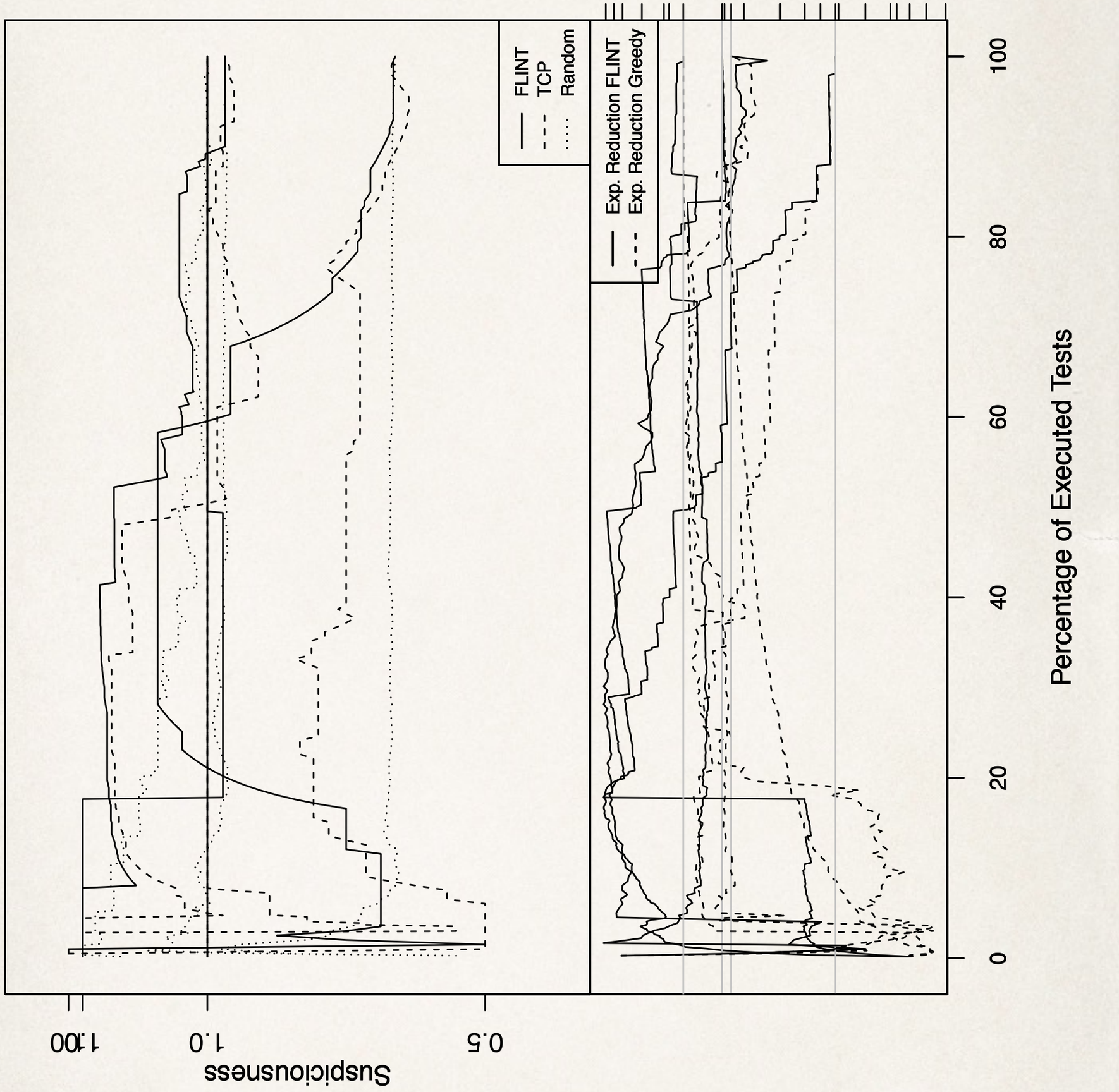
Compute the Shannon Entropy of Fault Locality

$$\mathbf{H}_{T_i}(S) = -\sum_{j=1}^{m} \mathbf{P}_{T_i}(B(s_j)) \cdot \log \mathbf{P}_{T_i}(B(s_j))$$

Assuming the failure rate observed so far, compute lookahead P

$$\mathbf{P}_{T_{i+1}}(B(s_j)) = \begin{array}{c} \mathbf{P}_{T_{i+1}}(B(s_j)|F(t_{i+1})) \cdot \alpha + \\ \mathbf{P}_{T_{i+1}}(B(s_j)|\neg F(t_{i+1})) \cdot (1-\alpha) \end{array}$$

We can predict the information gain of a test case!


Percentage of Executed Tests
0
20
40
60
80
100
Expense Reduction
Exp. Reduction FLINT
Exp. Reduction Greedy
FLINT
TCP
Random
Suspiciousness
0.5
1.0

Lessons Learned #1

- Probabilistic view works! Even when there are some wrinkles in your formulations.
- Software artefacts tend to exhibit continuity (e.g. coverage of a test case does not change dramatically between versions, etc). This helps the point 1.

Problem Solved…?

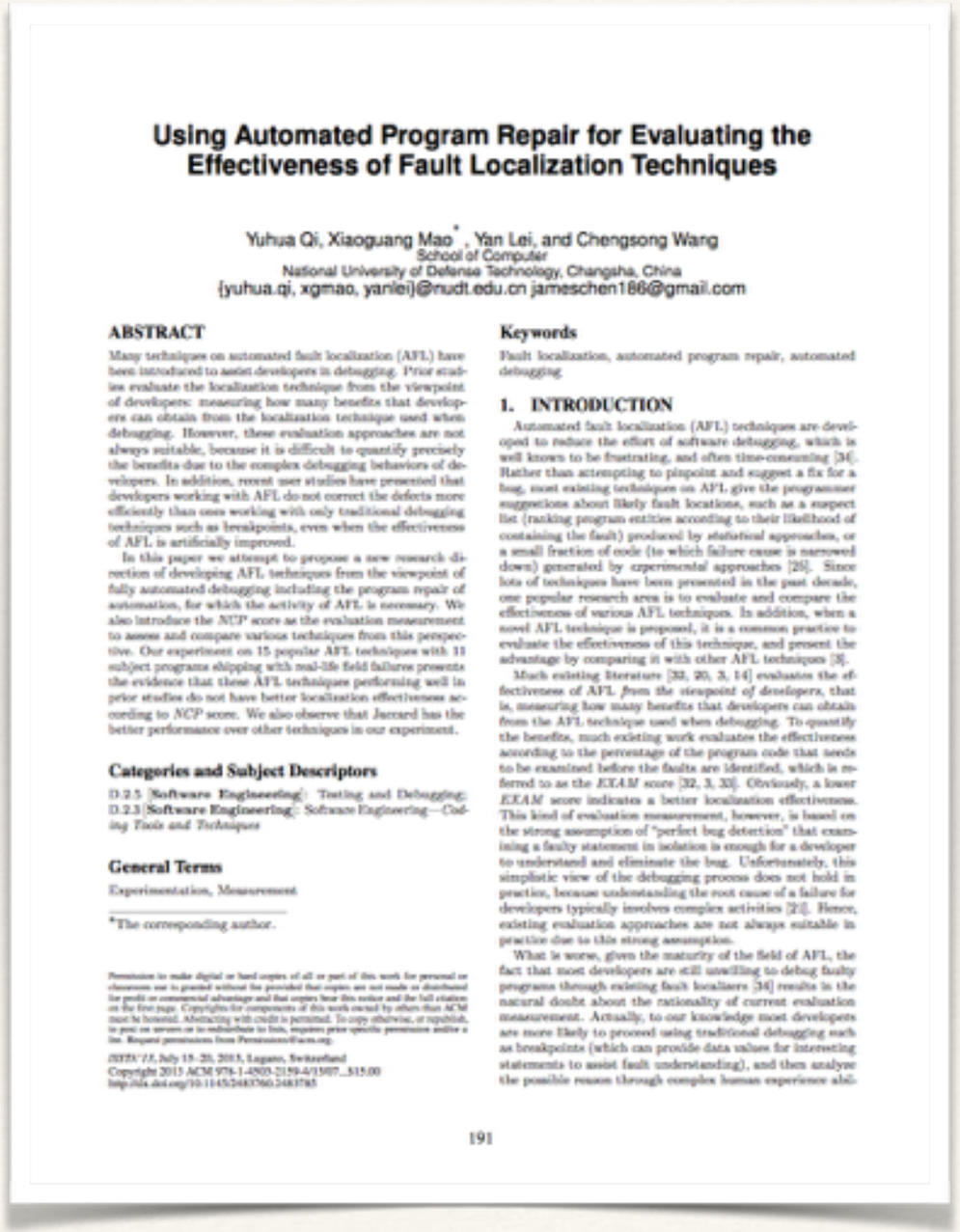
- Various empirical study established partial rankings between formulas at first.
- Then a theoretical study *proved* the dominance between formulas and their performance in Expense metrics.

But then machines arrived.

Aside: we also automatically evolved formulas using GP, which we then proved cannot be bettered by humans. So technically machines arrived *twice*.

Machine Based Evaluation

- Qi et al. took a backward approach
- Use suspicious score as weights to mutate program states until Genetic Programming can repair the fault.
- The better the localisation, the quicker the repair will be found.



Using Automated Program Repair for Evaluating the Effectiveness of Fault Localization Techniques

Yuhua Qi, Xiaoguang Mao*, Yan Lei, and Chengsong Wang
School of Computer
National University of Defense Technology, Changsha, China
{yuhua.qi, xgmao, yanlei}@nudt.edu.cn jameschen186@gmail.com

ABSTRACT

Many techniques on automated fault localization (AFL) have been introduced to assist developers in debugging. Prior studies evaluate the localization technique from the viewpoint of developers: measuring how many benefits that developers can obtain from the localization technique used when debugging. However, these evaluation approaches are not always suitable, because it is difficult to quantify precisely the benefits due to the complex debugging behaviors of developers. In addition, recent user studies have presented that developers working with AFL do not correct the defects more efficiently than ones working with only traditional debugging techniques such as breakpoints, even when the effectiveness of AFL is artificially improved.

In this paper we attempt to propose a new research direction of developing AFL techniques from the viewpoint of fully automated debugging including the program repair of automation, for which the activity of AFL is necessary. We also introduce the *NCP* score as the evaluation measurement to assess and compare various techniques from this perspective. Our experiment on 15 popular AFL techniques with 11 subject programs shipping with real-life field failures presents the evidence that these AFL techniques performing well in prior studies do not have better localization effectiveness according to *NCP* score. We also observe that Jaccard has the better performance over other techniques in our experiment.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.3 [**Software Engineering**]: Software Engineering—*Coding Tools and Techniques*

General Terms

Experimentation, Measurement

*The corresponding author.

ISSTA'15, July 15–20, 2015, Lugano, Switzerland

http://dx.doi.org/10.1145/2483760.2483785

Strange Results

- Theory says Jaccard formula is worse than Op2.
- But machines found it much easier to repair programs when using the localisation from Jaccard.
- Why?

Abstraction destroys Information

- Expense metric assumes linear consumption of the result (i.e. developer checks statements following the ranking).
- GP consumes raw suspiciousness numbers, which is a much richer source of information.

Same ranking, completely different amount of information.

New Evaluation Metric

- Following the way we predicted information yield, we should be able to describe the true fault locality as a probability distribution.
- Subsequently, measure the cross-entropy between the true distribution and one generated by any technique.

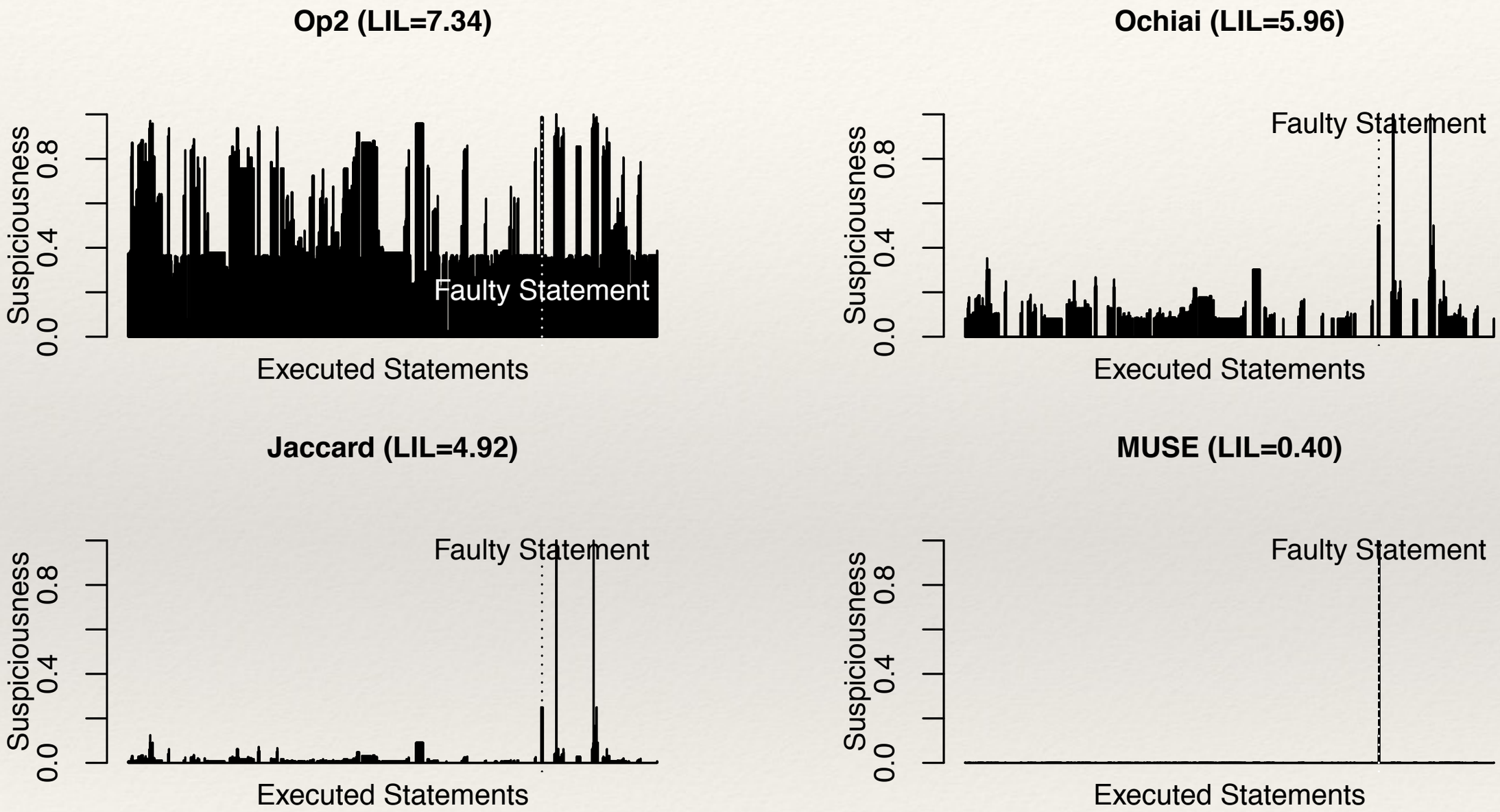
$$\mathcal{L}(s_i) = \begin{cases} 1 & (s_i = s_f) \\ \epsilon & (0 < \epsilon \ll 1, s_i \in S, s_i \neq s_f) \end{cases}$$

$$P_\tau(s_i) = \frac{\tau(s_i)}{\sum_{i=1}^{n} \tau(s_i)}, (1 \leq i \leq n)$$

$$D_{KL}(P_\mathcal{L} || P_\tau) = \sum_i \ln \frac{P_\mathcal{L}(s_i)}{P_\tau(s_i)} P_\mathcal{L}(s_i)$$

Locality Information Loss (LIL)
defined with Kullback-Leibler divergence

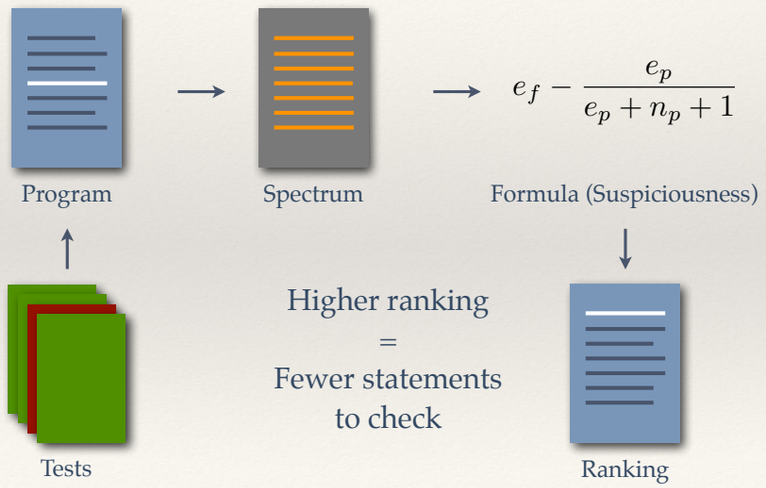
Worth a thousand words.

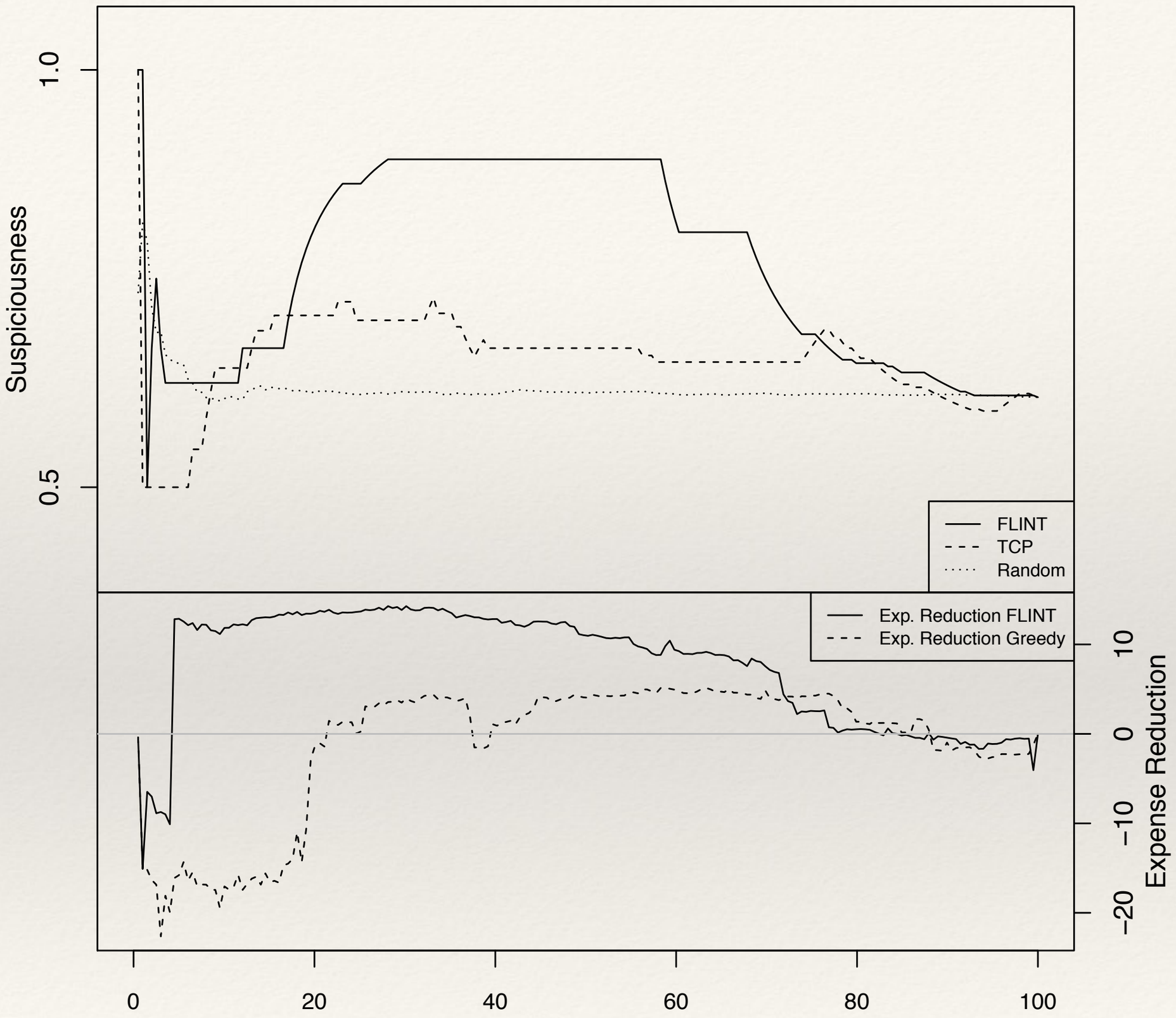


Lessons Learned #2

- Entropy measures are much richer than simply counting something: it gives you a holistic view.
- Cross-entropy is a vastly underused tool in software engineering in general.

Spectra Based Fault Localisation




grep, v3, F_KP_3
Suspiciousness
1.0
0.5
FLINT
TCP
Random
Exp. Reduction FLINT
Exp. Reduction Greedy
10
0
−10
−20
Expense Reduction
0
20
40
60
80
100
Percentage of Executed Tests