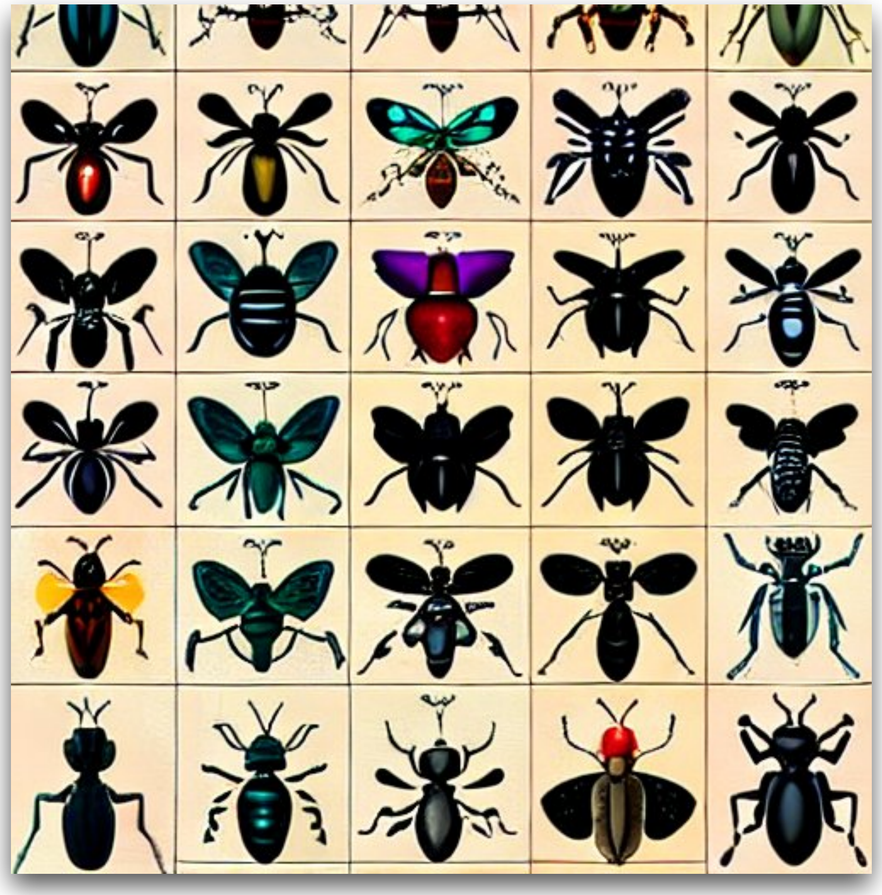
STAAR Summer Workshop 2023



Lessons from 10 Years of Automated Debugging Research

Shin Yoo

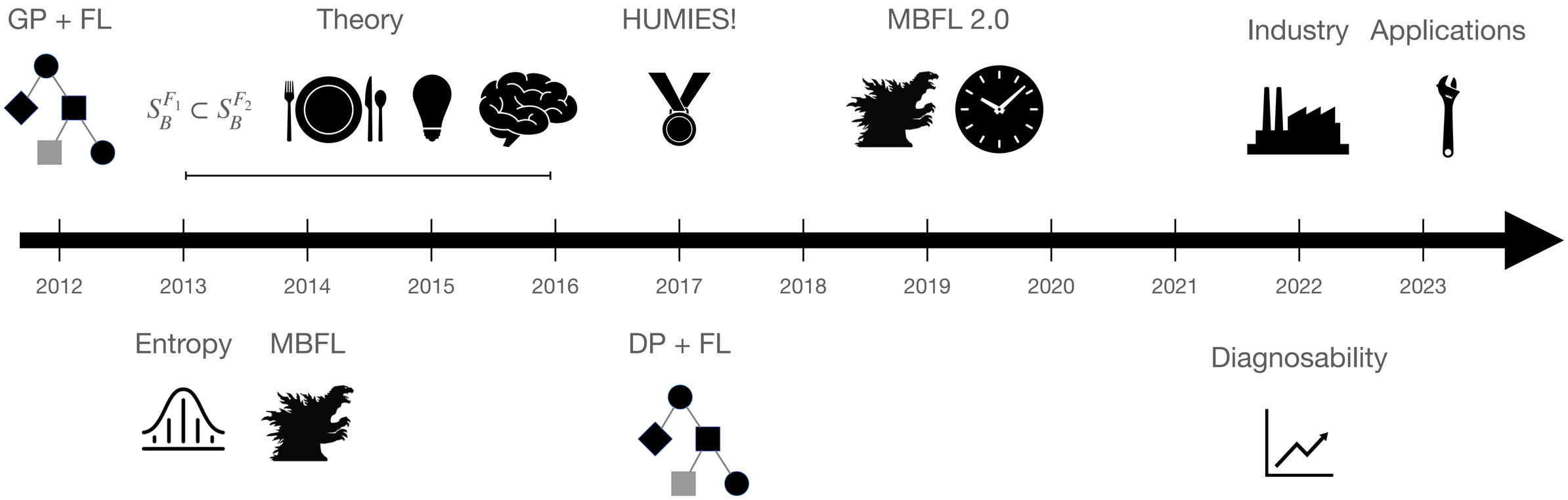
Who am I?

Shin Yoo

- Associate Prof@KAIST / Republic of Korea
- Leads Computational Intelligence for Software Engineering Group (https://coinse.github.io)
- Search Based Software Engineering, Automated Debugging, Software Testing…

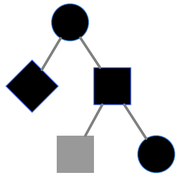
When I say “lessons”, I mean that ***I*** learnt these things. I do not mean to claim that these are some golden rules in research or anything else. Treat this talk just as a story of how certain techniques came to be… 🤠

Retrospective on My FL Work



Retrospective on My FL Work

GP + FL

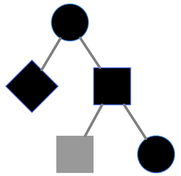


Theory

$S_B^{F_1} \subset S_B^{F_2}$

HUMIES!

DP + FL



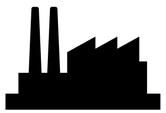
MBFL

MBFL 2.0

Entropy

Diagnosability

Industry



Applications

Everything started with people.

Prof. Mark Harman

Dr. Bill Langdon

Prof. T. Y. Chen

Prof. Moonzoo Kim

Dr. Jeongju Sohn

Prof. Robert Feldt

Dr. David Clark

Prof. Xiaoyuan Xie

Prof. Yunho Kim

Dr. Jinhan Kim

Dongwon Hwang
SAP Labs Korea

Jingun Hong
SAP Labs Korea

Gabin An
(PhD Candidate)

Sungmin Kang
(PhD Candidate)

Juyeon Yoon
(PhD Candidate)

Year 2008

- Bill joins CREST (Centre for Research on Evolution, Search, & Testing) at King's College London.
- He immediately gets on with GPGPU for genetic programming, Higher-order Mutation, and what will later become Genetic Improvement.

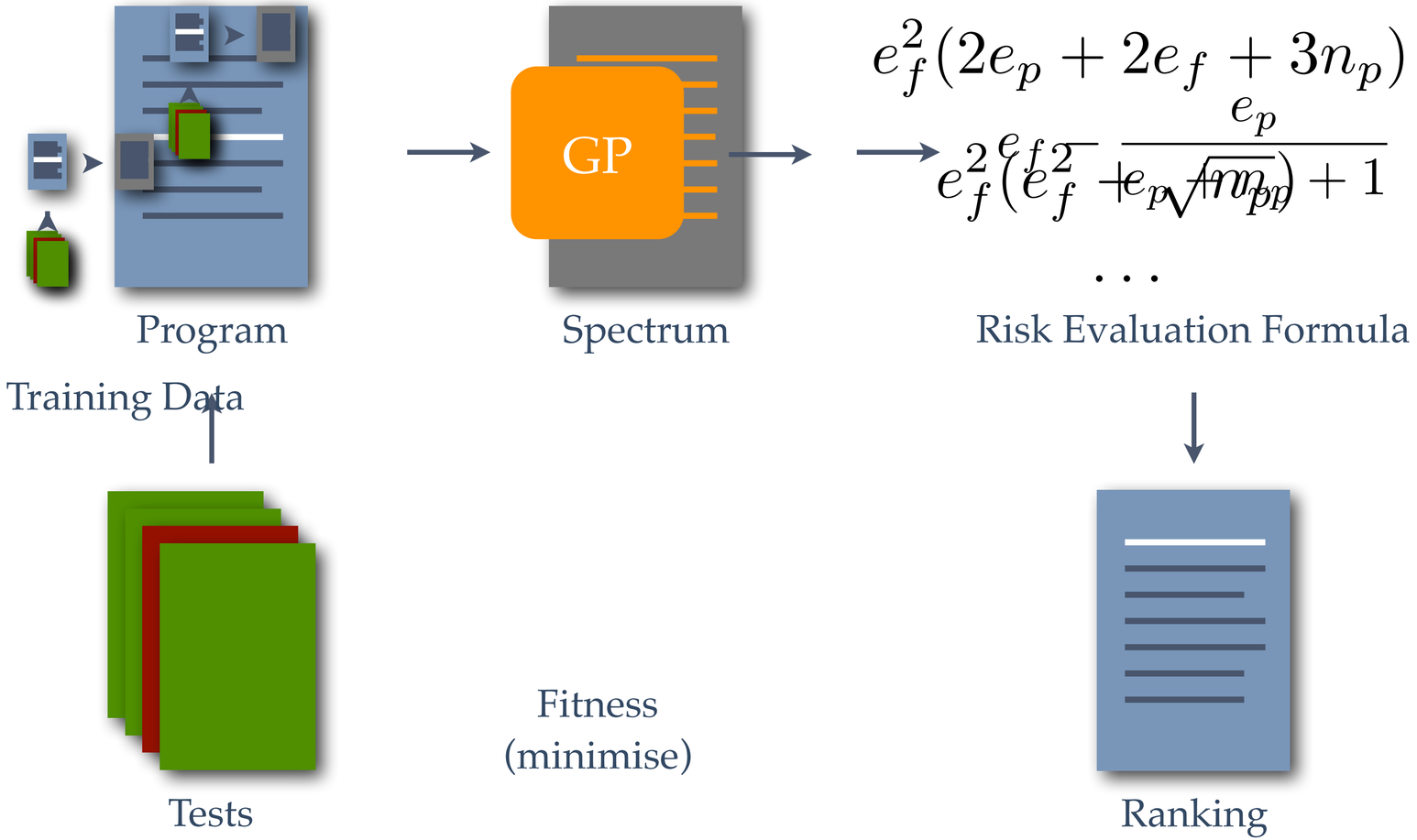
Dr. Bill Langdon

‘That stuff looks incredibly cool - what can I do with GP and GPGPU…?’

Me circa 2010

Evolving SBFL Formulas using GP

Yoo, SSBSE 2012



Evolved Formulas

Little did I know that these IDs would be their names…

ID	Refined Formula
GP01	$e_f(n_p + e_f(1+\sqrt{e_f}))$
GP02	$2(e_f+\sqrt{n_p})+\sqrt{e_p}$
GP03	$\sqrt{\lvert e_f^2-\sqrt{e_p}\rvert}$
GP04	$\sqrt{\lvert\frac{n_p}{e_p-n_p}-e_f\rvert}$
GP05	$\frac{(e_f+n_p)\sqrt{e_f}}{(e_f+e_p)(n_pn_f+\sqrt{e_p})(e_p+n_p)\sqrt{\lvert e_p-n_p\rvert}}$
GP06	e_fn_p
GP07	$2e_f(1+e_f+\frac{1}{2n_p})+(1+\sqrt{2})\sqrt{n_p}$
GP08	$e_f^2(2e_p+2e_f+3n_p)$
GP09	$\frac{e_f\sqrt{n_p}}{n_p+n_p}+n_p+e_f+e_f^3$
GP10	$\sqrt{\lvert e_f-\frac{1}{n_p}\rvert}$
GP11	$e_f^2(e_f^2+\sqrt{n_p})$
GP12	$\sqrt{e_p+e_f+n_p-\sqrt{e_p}}$
GP13	$e_f(1+\frac{1}{2e_p+e_f})$
GP14	$e_f+\sqrt{n_p}$
GP15	$e_f+\sqrt{n_f+\sqrt{n_p}}$

ID	Refined Formula
GP16	$\sqrt{e_f^{\frac{3}{2}}+n_p}$
GP17	$\frac{2e_f+n_f}{e_f-n_p}+\frac{n_p}{\sqrt{e_f}}-e_f-$
GP18	$e_f^3+2n_p$
GP19	$e_f\sqrt{\lvert e_p-e_f+n_f-n$
GP20	$2(e_f+\frac{n_p}{e_p+n_p})$
GP21	$\sqrt{e_f+\sqrt{e_f+n_p}}$
GP22	$e_f^2+e_f+\sqrt{n_p}$
GP23	$\sqrt{e_f}(e_f^2+\frac{n_p}{e_f}+\sqrt{n_p}+$
GP24	$e_f+\sqrt{n_p}$
GP25	$e_f^2+\sqrt{n_p}+\frac{\sqrt{e_f}}{\sqrt{\lvert e_p-n_p\rvert}}$
GP26	$2e_f^2+\sqrt{n_p}$
GP27	$\frac{n_p\sqrt{(n_pn_f-e_f)}}{e_f+n_pn_f}$
GP28	$e_f(e_f+\sqrt{n_p}+1)$
GP29	$e_f(2e_f^2+e_f+n_p)+\frac{(e_f$
GP30	$\sqrt{\lvert e_f-\frac{n_f-n_p}{e_f+n_f}\rvert}$

tiveness by training and evaluating models on each set of features independently.

RQ4: How much training data does *Savant* need to work effectively? In the default setting, *Savant* uses $n-1$ out of n bugs as training data. We investigate the effectiveness of *Savant* with reduced amount of training data. We do this by evaluating *Savant* in a k-fold cross validation setting, for k ranging from 2–10.

RQ5: How efficient is *Savant*? In this research question, we measure the average running time needed for *Savant* to output a ranked list of methods for a given bug.

5.3 Findings

RQ1: *Savant*'s Effectiveness. Table 3 shows the effectiveness of *Savant* on bugs from the five Defects4J projects. *Savant* successfully localizes 63.03, 101.72, and 122.00 out of 357 bugs in terms of average acc@1, acc@3, and acc@5 score, respectively. The wasted effort across all projects is 288.97, 811.14, and 1277.14 (wef@1, wef@3, and wef@5, respectively). The overall average MAP score is 0.221. Among the five projects, *Savant* is most effective on Commons Lang, achieving the highest *acc@k* (29, 36, and 41, respectively), and a MAP score of 0.535. Over these experiments, *Savant* terminated Daikon twice on a bug from the Math project due to the time limit. In total, we invoked Daikon 129,798 times for the 357 faulty versions.

RQ2: *Savant* vs. Previous work.
Table 4 shows the effectiveness of the 12 baseline approaches on our dataset. Among the baselines, the top four

	Lang	22.00	32.00	38.09	43.00	112.50	168.38	0.37
	OA	39.96	64.92	85.24	317.04	911.42	1464.26	0.15
GP19	Chart	4.00	4.00	6.00	22.00	66.00	108.00	0.15
	Closure	2.60	6.11	9.22	130.40	385.42	634.37	0.04
	Math	8.35	19.77	28.89	97.65	275.57	433.66	0.16
	Time	3.00	3.00	3.00	24.00	72.00	120.00	0.05
	Lang	22.00	32.00	38.09	43.00	112.50	168.38	0.37
	OA	39.95	64.88	85.20	317.05	911.49	1464.41	0.15
Ochiai	Chart	2.00	4.00	6.00	24.00	69.00	109.00	0.12
	Closure	1.95	4.70	8.77	131.05	388.77	639.29	0.04
	Math	8.27	19.71	28.82	97.73	275.71	433.99	0.16
	Time	5.50	7.00	9.00	21.50	62.50	99.50	0.12
	Lang	18.84	28.50	35.09	46.16	125.47	187.35	0.34
	OA	36.56	63.91	87.68	320.44	921.45	1469.13	0.14
MUL	Chart	4.35	6.44	8.18	21.65	62.14	98.35	0.15
	Closure	1.83	5.14	7.84	131.17	388.24	639.73	0.03
	Math	6.33	16.88	25.36	99.67	283.34	448.52	0.14
	Time	3.71	5.47	6.52	23.29	67.33	108.65	0.10
	Lang	22.80	30.04	34.06	42.20	113.80	177.57	0.36
	OA	39.02	63.97	81.96	317.98	914.85	1472.82	0.14
Carot$^+$	Chart	2.05	3.86	5.16	22.95	65.95	104.24	0.11
	Closure	0.35	1.01	1.88	132.65	396.99	659.67	0.01
	Math	5.81	13.84	20.48	99.19	284.99	457.32	0.10
	Time	0.90	1.86	2.36	26.10	76.69	126.25	0.06
	Lang	20.48	33.84	39.58	41.52	102.21	146.49	0.41
	OA	29.59	54.41	69.46	322.41	926.83	1493.97	0.12

Savant's improvement

Legend 10% ≤ improvement <20% | 20% ≤ improvement <50% | 50% ≤ improvement <100% | 100% ≤ improvement

performers are ER1^b, *GP*13, *GP*19 and Multric, and they achieve more or less the same score for many metrics. The absolute best performers are ER1^b and *GP*13.

Savant outperforms these baselines in all metrics. It outperforms ER1^b and *GP*13 by 57.73%, 56.69%, and 43.13% in terms of average acc@1, acc@3, and acc@5 scores. The wasted effort of *Savant* is lower than those of ER1^b and *GP*13 by 8.85%, 10.94%, and 12.78% (wef@1, wef@3, and wef@5 respectively). In terms of average MAP score, our approach is more effective than ER1^b and *GP*13 by 51.37%.

184

An Offer that I could not turn down…

(despite it being full of maths)

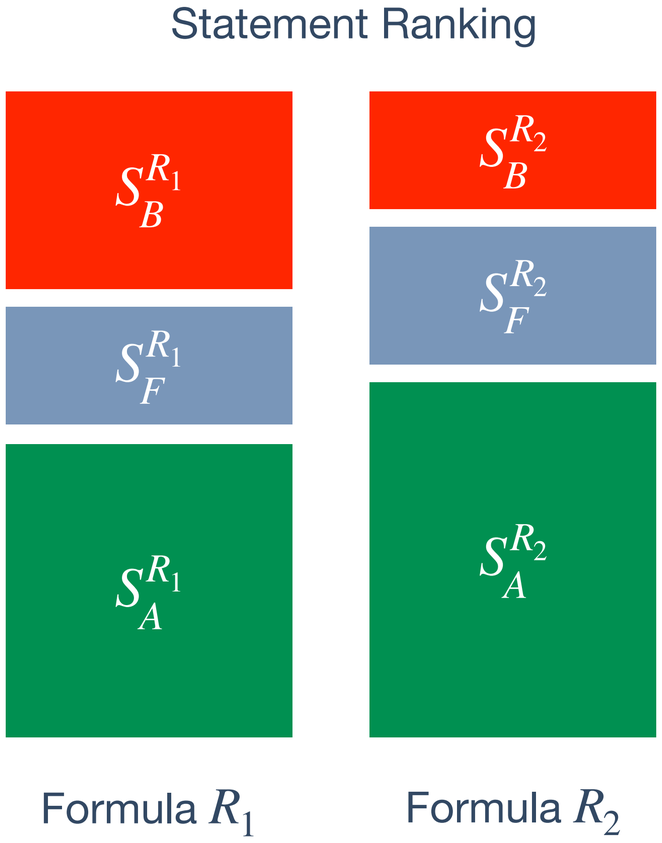
Prof. T. Y. Chen

Prof. Xiaoyuan Xie

$$S_B^R = \{s_i \in S | R(s_i) > R(s_f), 1 \leq i \leq n\}$$
$$S_F^R = \{s_i \in S | R(s_i) = R(s_f), 1 \leq i \leq n\}$$
$$S_A^R = \{s_i \in S | R(s_i) < R(s_f), 1 \leq i \leq n\}$$

Xiaoyuan and T. Y. essentially formulated the comparison between SBFL formulas as algebraic proofs about set membership.

A Theoretical Analysis of the Risk Evaluation Formulas for Spectrum-Based Fault Localisation, X. Xie, T.Y. Chen, F.C. Kuo, B. Xu,
ACM Transactions on Software Engineering and Methodology 22(4):1-40

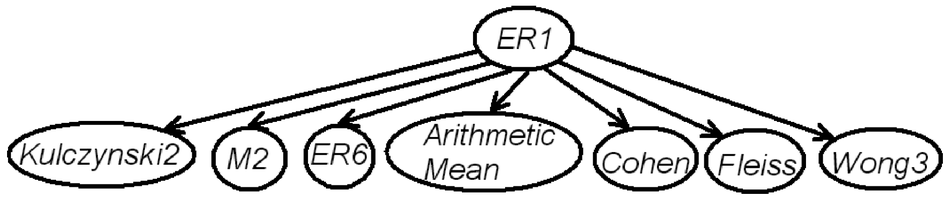


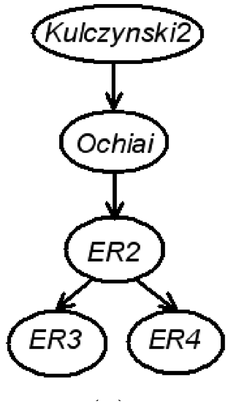
An Offer that I could not turn down…

(despite it being full of maths)

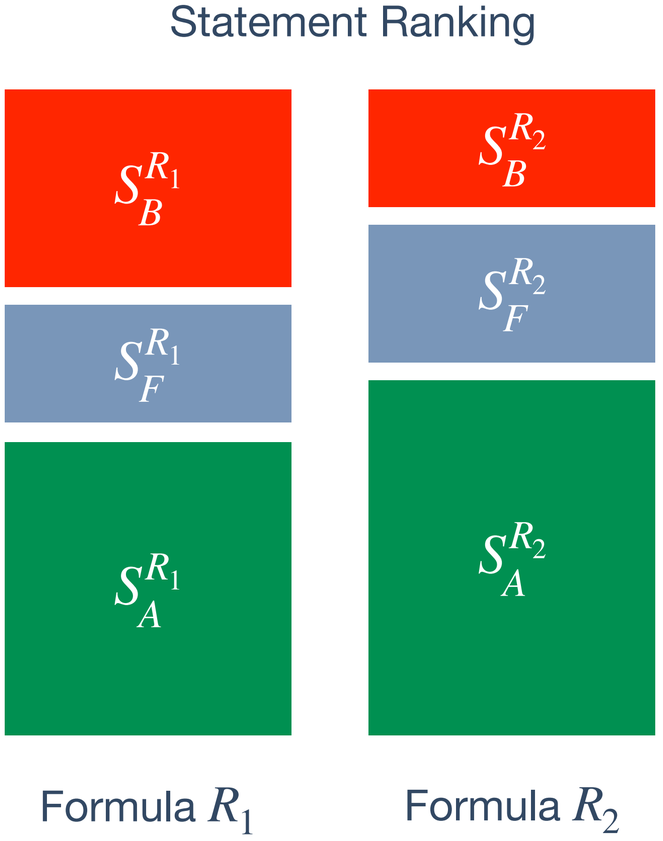
Prof. T. Y. Chen

Prof. Xiaoyuan Xie





- Formula R_1 *dominates* formula R_2 ($R_1 \rightarrow R_2$) if $S_B^{R_1} \subseteq S_B^{R_2} \wedge S_A^{R_2} \subseteq S_A^{R_1}$
- Formula R_1 is *equivalent* to formula R_2 ($R_1 \leftrightarrow R_2$) if $S_B^{R_1} = S_B^{R_2} \wedge S_F^{R_1} = S_F^{R_2} \wedge S_A^{R_1} = S_A^{R_2}$
- “Do you want to see where the evolved formulas are ranked in the hierarchy?” (i.e., the offer)



Proof that GP was as good as humans

Xie et al., SSBSE 2013

$ER1$ has been permanently expanded to $ER1'$ because $GP13$ joined the club.

Table 1. Investigated formulæ

	Name	Formula expression
ER1'	Naish1	$\begin{cases} -1 & \text{if } e_f < F \\ P - e_p & \text{if } e_f = F \end{cases}$
	Naish2	$e_f - \frac{e_p}{e_p + n_p + 1}$
	GP13	$e_f(1 + \frac{1}{2e_p + e_f})$
ER5	Wong1	e_f
	Russel & Rao	$\frac{e_f}{e_f + n_f + e_p + n_p}$
	Binary	$\begin{cases} 0 \text{ if } e_f < F \\ 1 \text{ if } e_f = F \end{cases}$
GP02		$2(e_f + \sqrt{n_p}) + \sqrt{e_p}$
GP03		$\sqrt{\lvert e_f^2 - \sqrt{e_p} \rvert}$
GP19		$e_f \sqrt{\lvert e_p - e_f + n_f - n_p \rvert}$

Three GP formulas formed their own maximal groups.


(Scene: T.Y., Mark, and Shin having lunch in the basement canteen of LSHTM, one day in 2014)
M: So, what next?
TY: Perhaps we can use search to find the greatest formula…?
S: Hmmmmm….

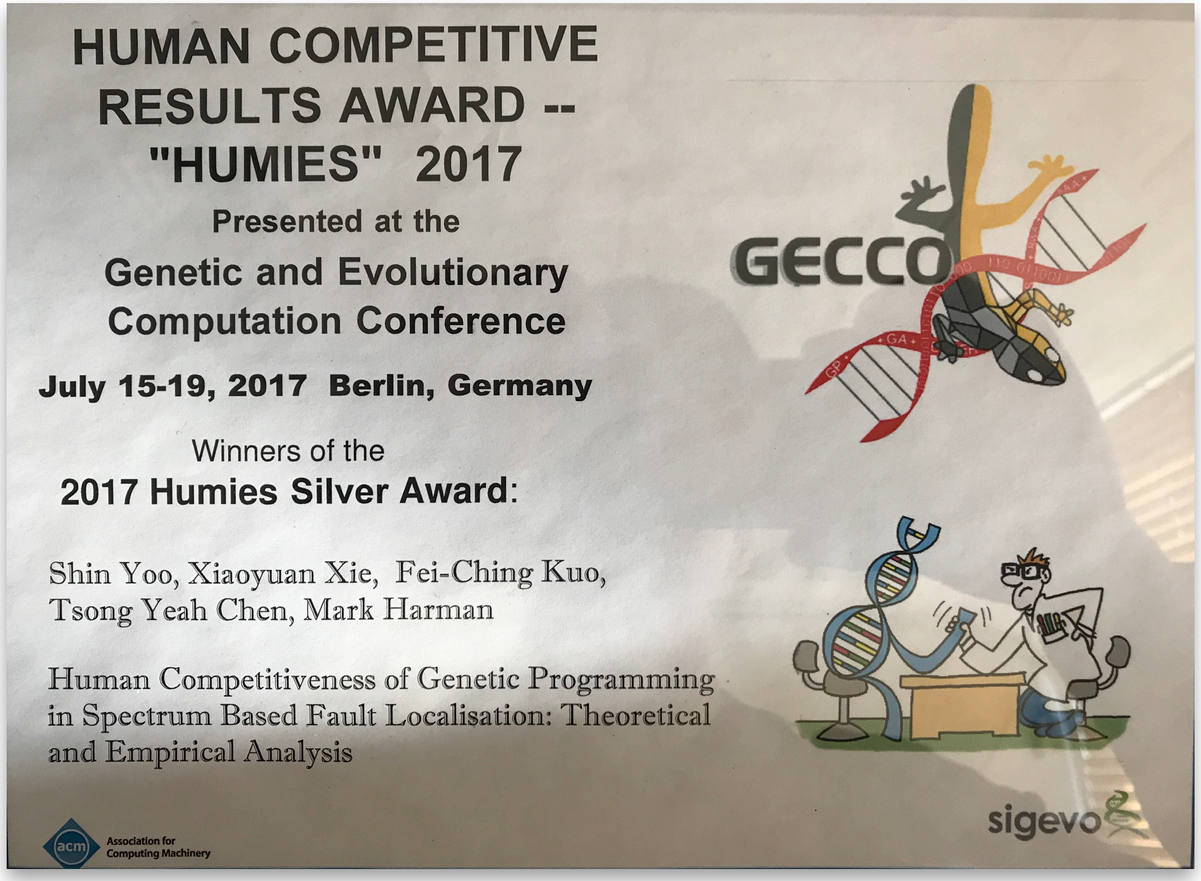
No Greatest SBFL Formula!

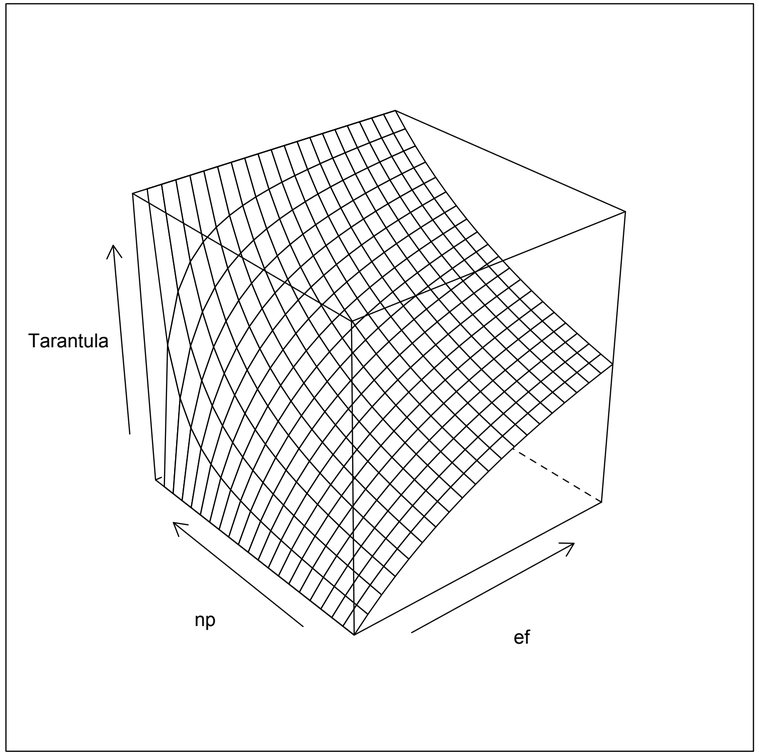
Yoo + Xie et al., TOSEM 2017

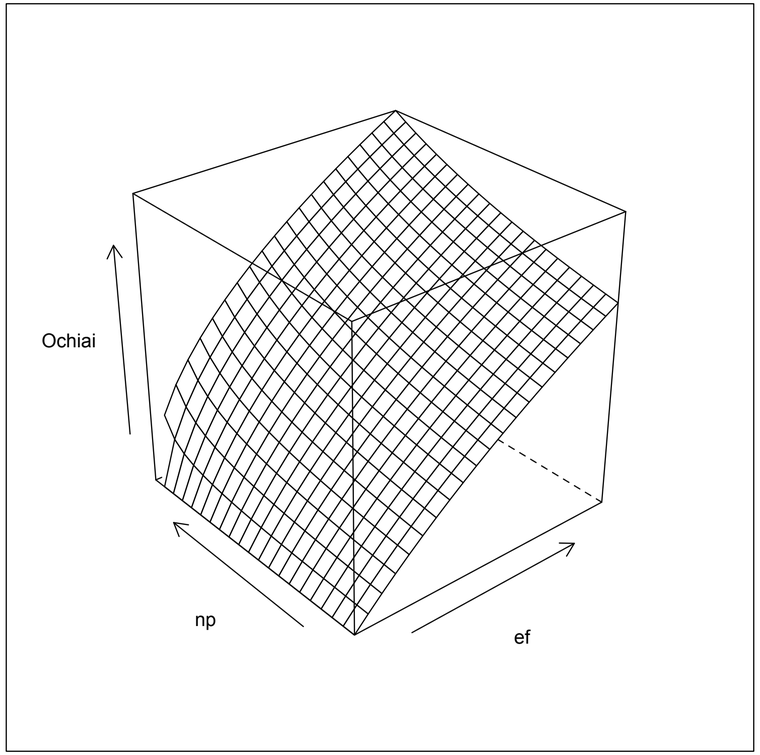
Soon after that lunch, I ended up with a proof that such formula cannot exist.

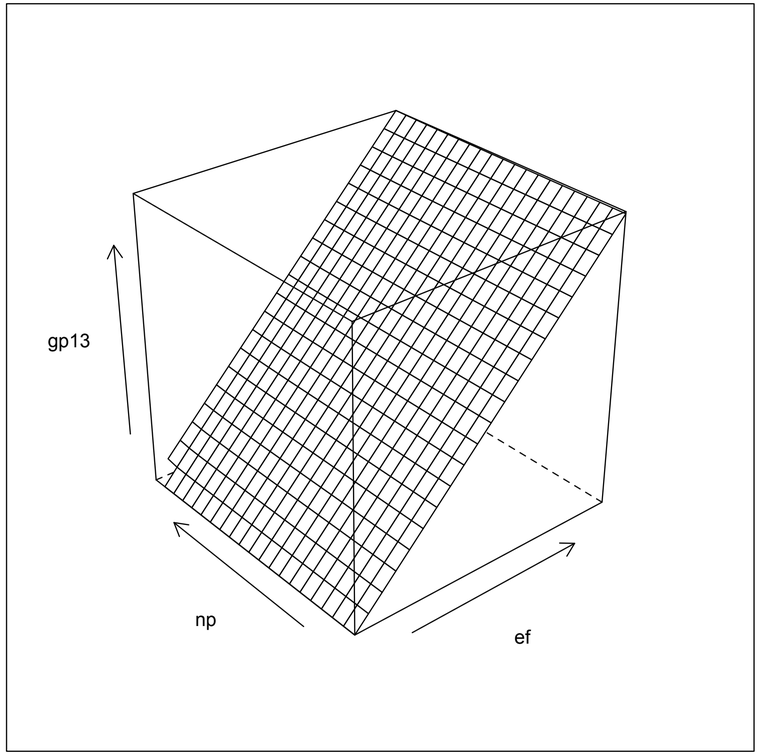
Sent it to Xiaoyuan and TY for rigorous check, kept my fingers crossed.

And then it was confirmed!

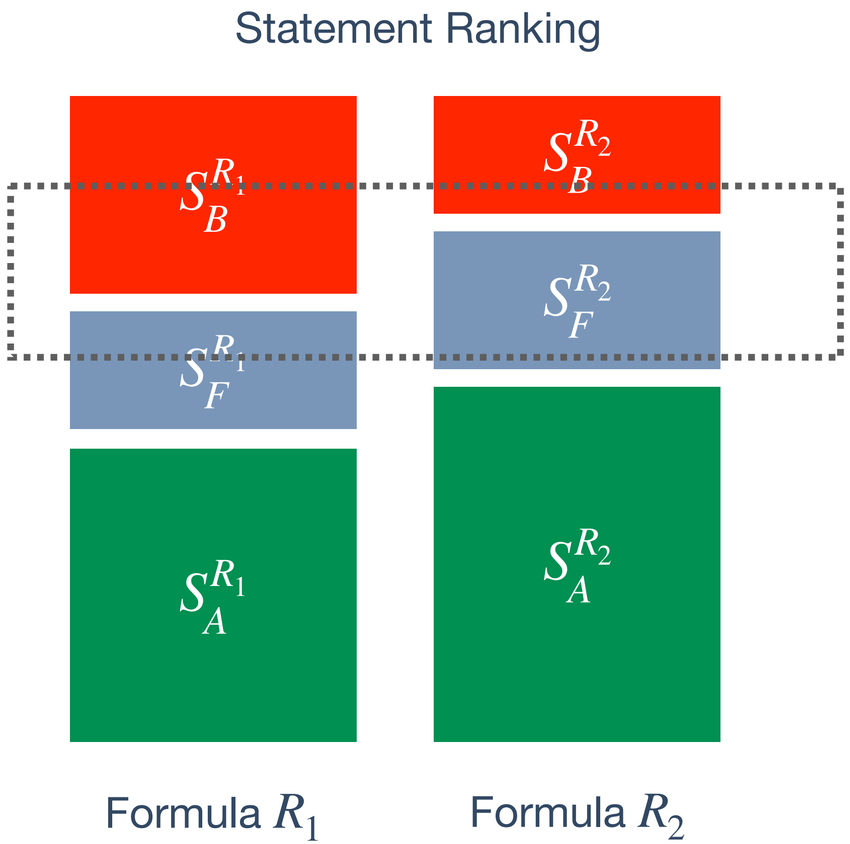



Tarantula
np
ef


Ochiai
np
ef


gp13
np
ef

Important Caveat: Consistent Tie-breaker

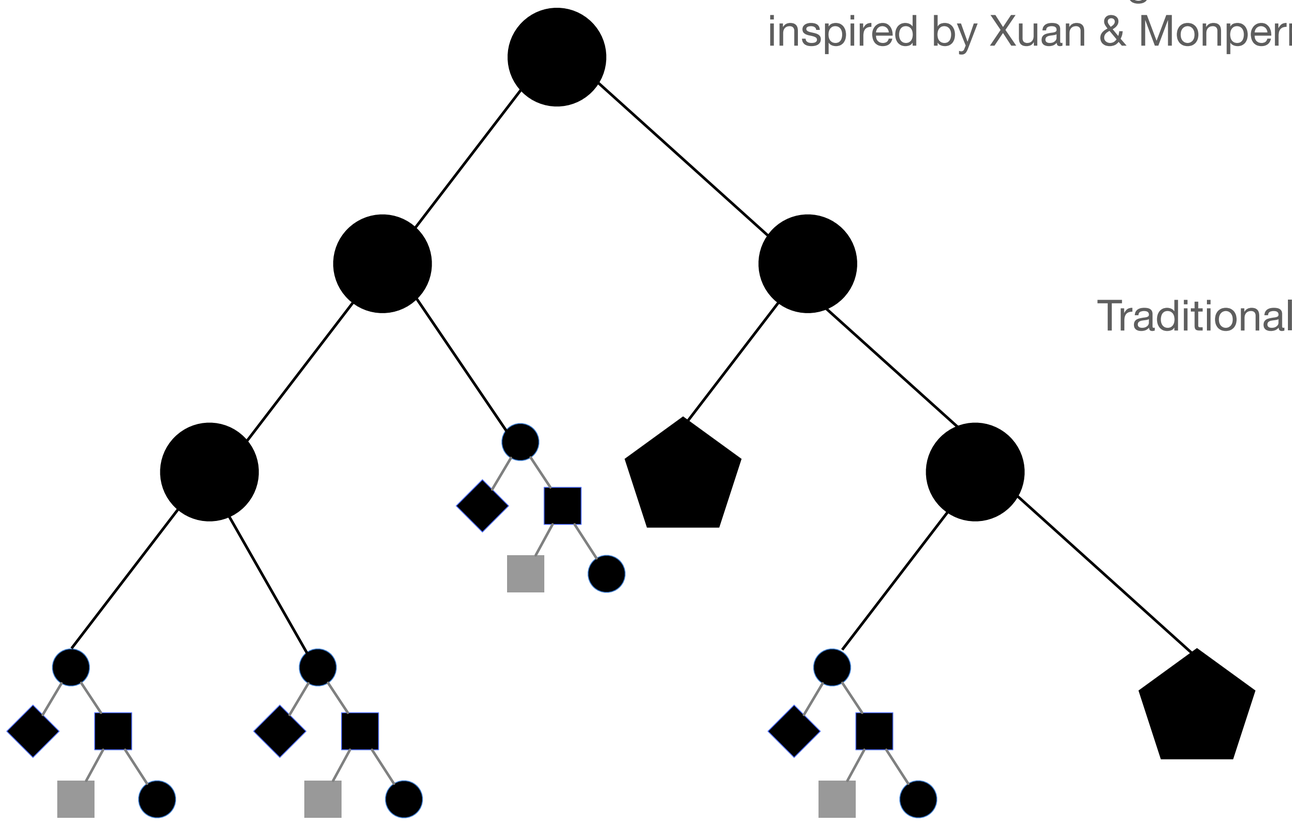


A choice of tie-breaker, combined with specific spectrum, can result in empirical results that contradict theoretical hierarchy.

Isn’t FL just DP but happening later?

Sohn & Yoo, ISSTA 2017

Still using GP but this time to aggregate multiple scores,
inspired by Xuan & Monperrus (ICSME 2014) as well as Le et al. (ISSTA 2016)



Traditional Defect Prediction (DP) features added:
age, churn, and complexity

Multiple SBFL formulas are terminal nodes

Dr. Jeongju Sohn

Lessons from my SBFL story

Sometimes imitating the cool stuff(people) works out :)

ort zone?
comfort
er, Clash

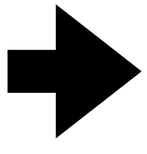
Do mental exercise; test your comfort zone.

Mutating buggy programs can help FL

Moon et al., ICST 2014

Mutate 

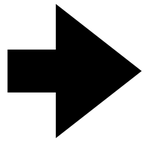
Correct Code

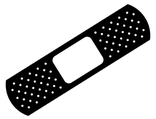


(mostly)

Mutate 

Buggy Code



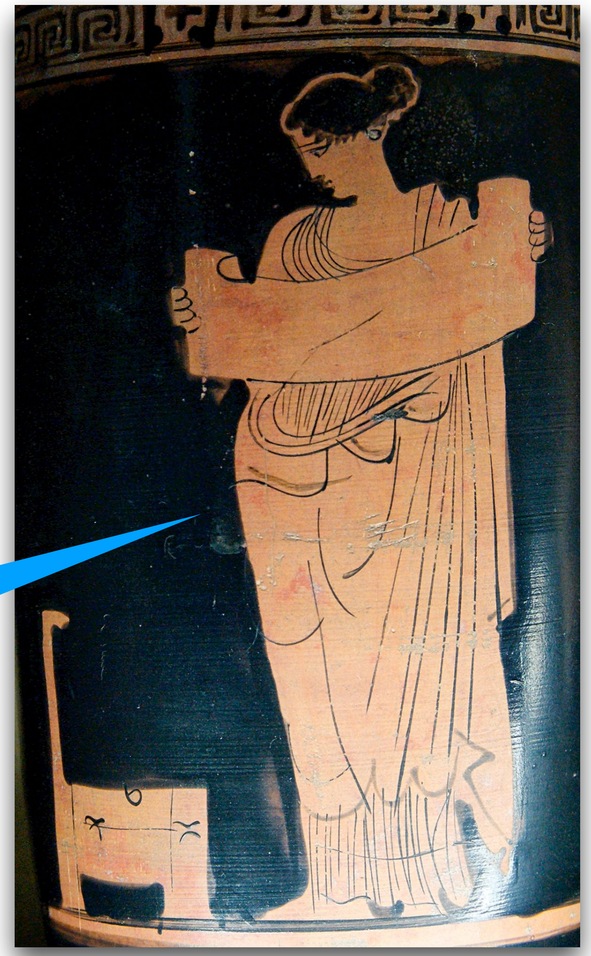


(sometimes!)

Prof. Moonzoo Kim

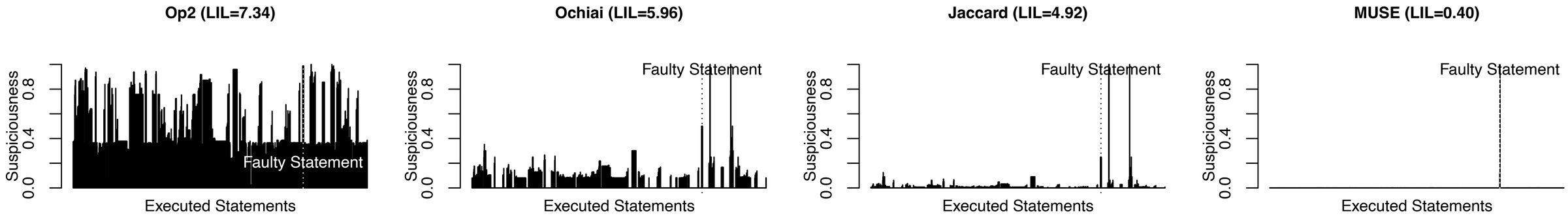
Prof. Yunho Kim

"But what did YOU do?"



Ranking is only for humans

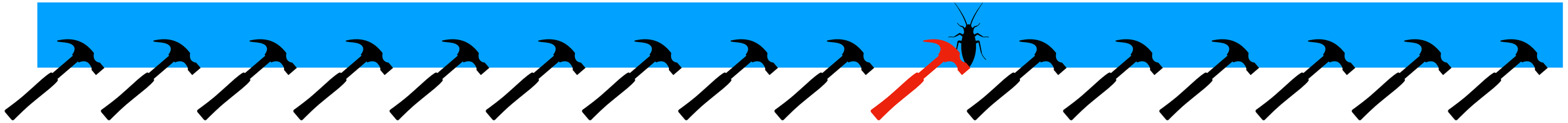
Moon et al., ICST 2014



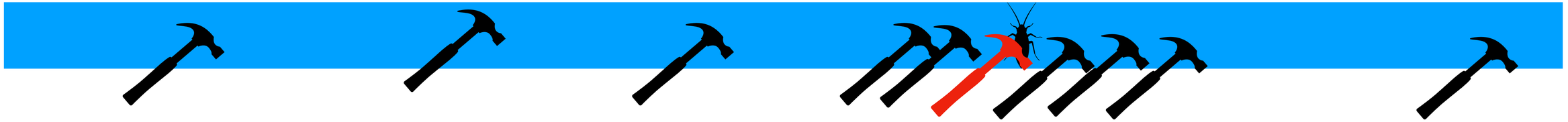
- Humans consume FL results as ranking: order matters.
- Machines (APR) consume FL results as weights: magnitude matters.
- I suggested Kullback-Leibler divergence as an FL evaluation metric… didn't catch on :)
- Qi et al., ISSTA 2013 is an independent confirmation.

A moment of reckoning years later…

MUSE



Unified Debugging, Lou et al., ISSTA 2020



I had a “doh!” moment when I read this paper - although they frame it as APR helping FL, to me it was incremental mutation that made MUSE more lightweight. A really clever idea.

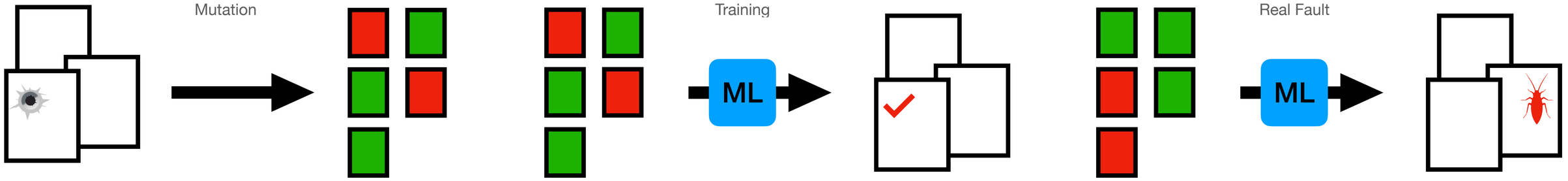
Can we pay first? 💸

Kim et al., ISSRE 2021

Prof. Robert Feldt

Dr. Jinhan Kim

Ahead-of-Time MBFL: mutants and bugs elicit similar reactions from test cases if they share the same location.

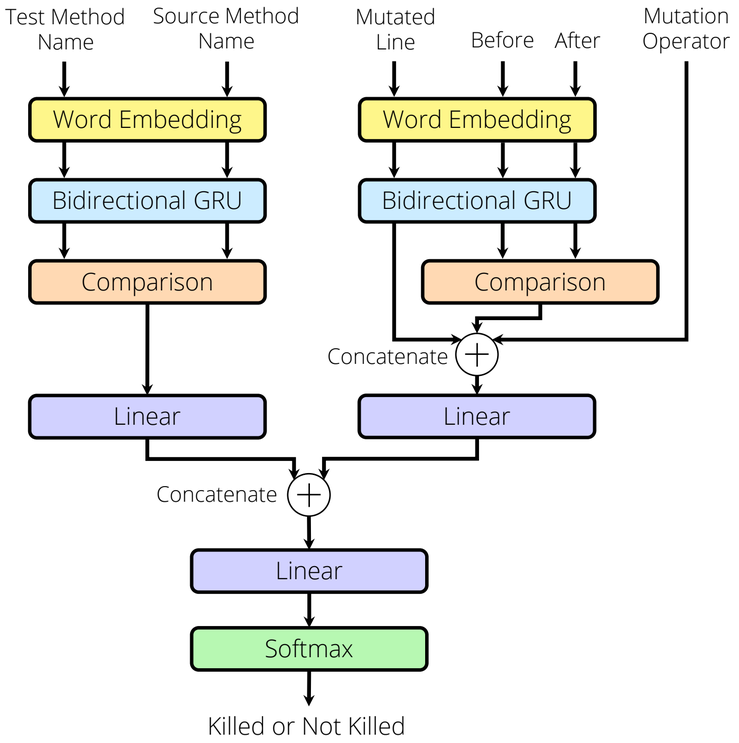


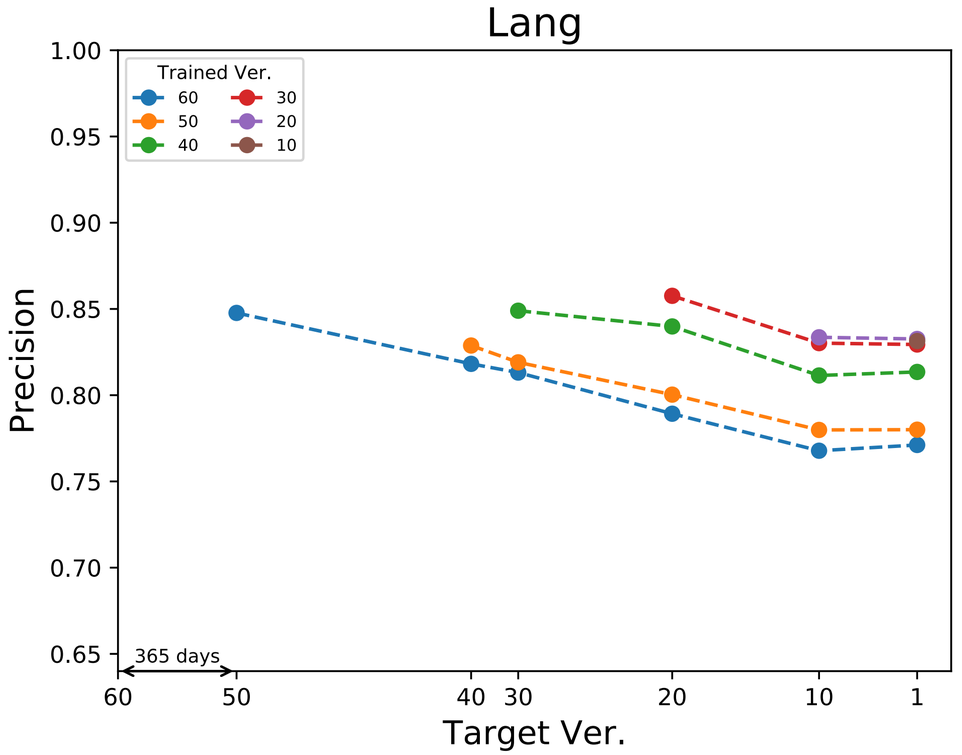
1. Do mutation analysis
2. Learn the reverse relation.
3. Pretend if real bug is a mutant

What if a new test fails?

Kim et al., TOSEM 2022

Dr. Jinhan Kim



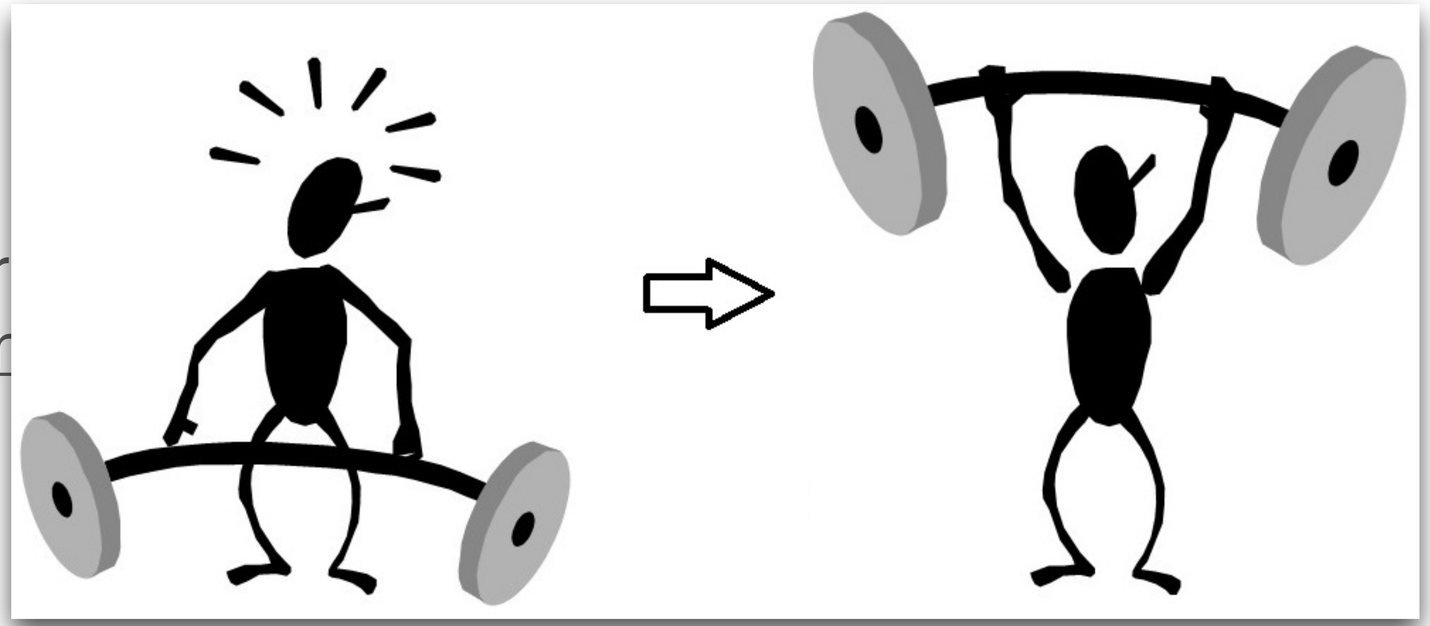


Learn the relationship between killed mutants and killing tests via the natural language channel

We can predict whether which new test cases will kill which new mutants pretty well.

Lessons from my MBFL story

Ideas may come too late, but they do come, and it is okay ;)



Your weakness may become your strength: often it is already a good problem to solve.

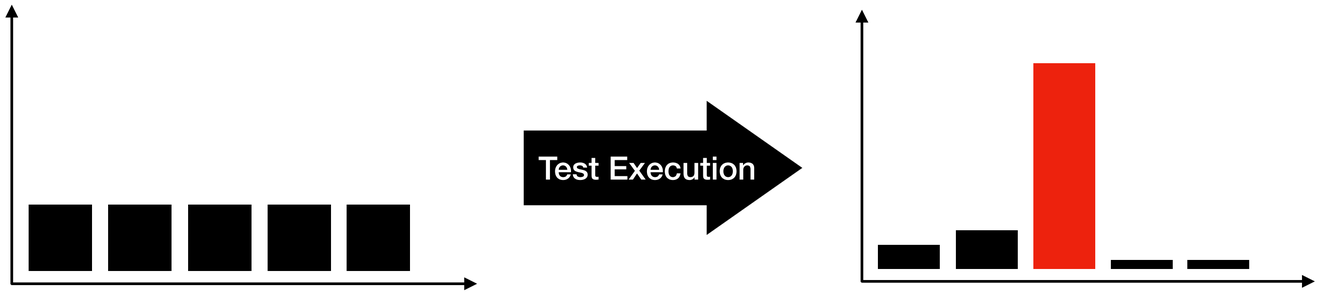
Information Theory can help FL

Yoo, Clark, and Harman, TOSEM 2013

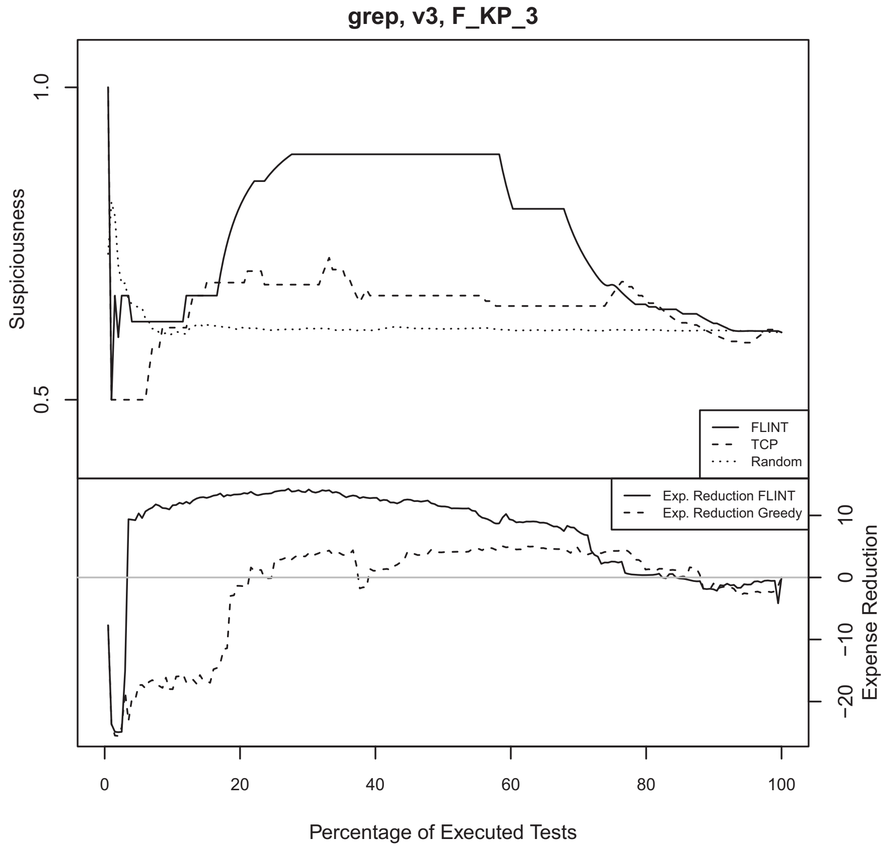
Dr. David Clark

Prof. Mark Harman

Certain test cases provide more “information” to FL than others.
Can we prioritize them?



FL is entropy reduction! That way, we can quantify the contribution from individual test cases.

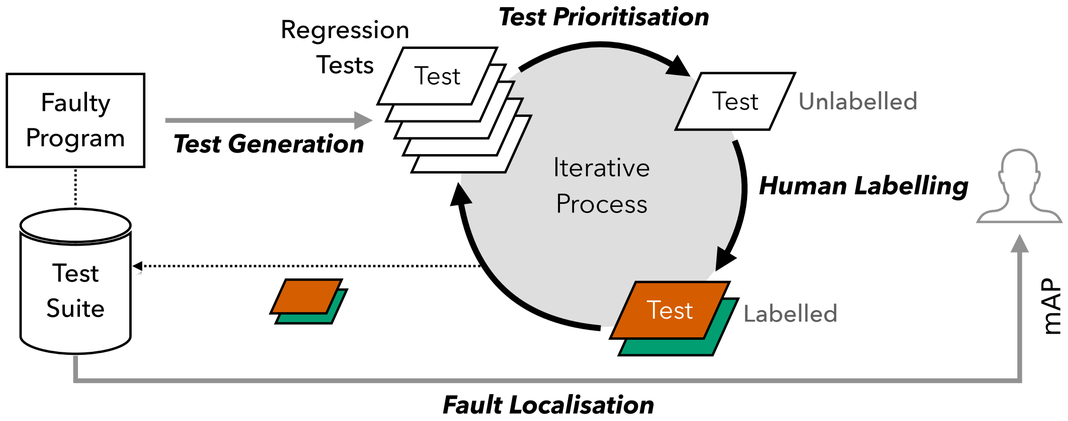


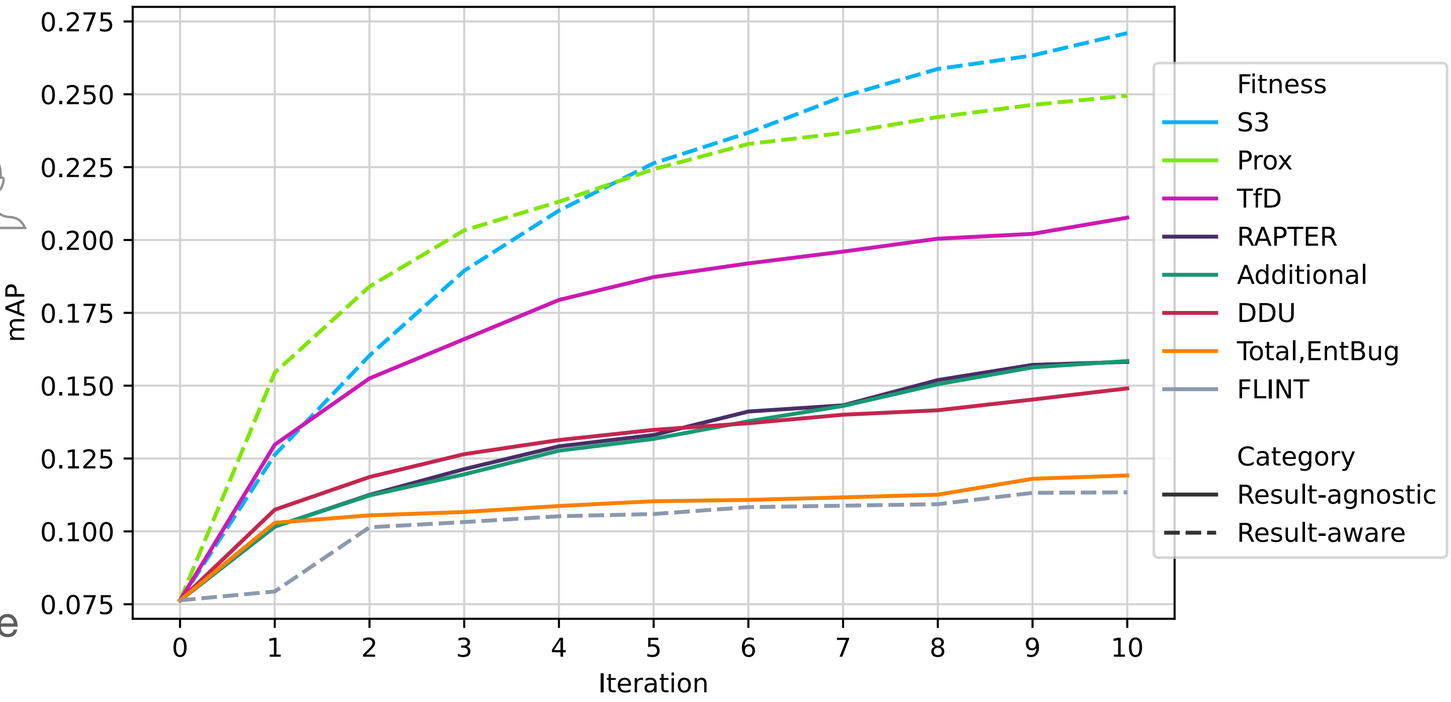
(a)

Quantitative Diagnosability of Tests

An & Yoo, ISSTA 2022

Gabin An
(PhD Candidate)





If we have to augment the test suite for FL as we go along, we want to involve the human oracle for the smallest number of time. For this, we want to choose the test case with the most information about fault diagnosis as the next test.

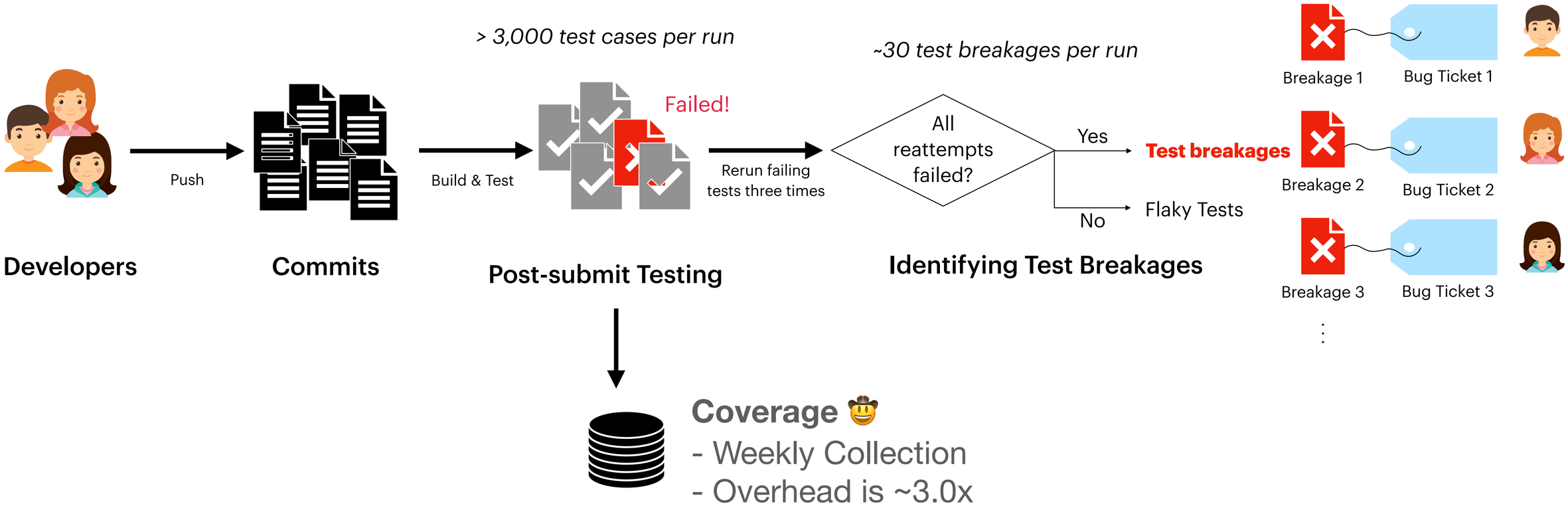
Taking FL to Industry

Our partner in crime…

Jingun Hong
SAP Labs Korea

Dongwon Hwang
SAP Labs Korea

SAP HANA: In-memory RDBMS, >11MLoc, >50K files in C/C++



Triaging with FL

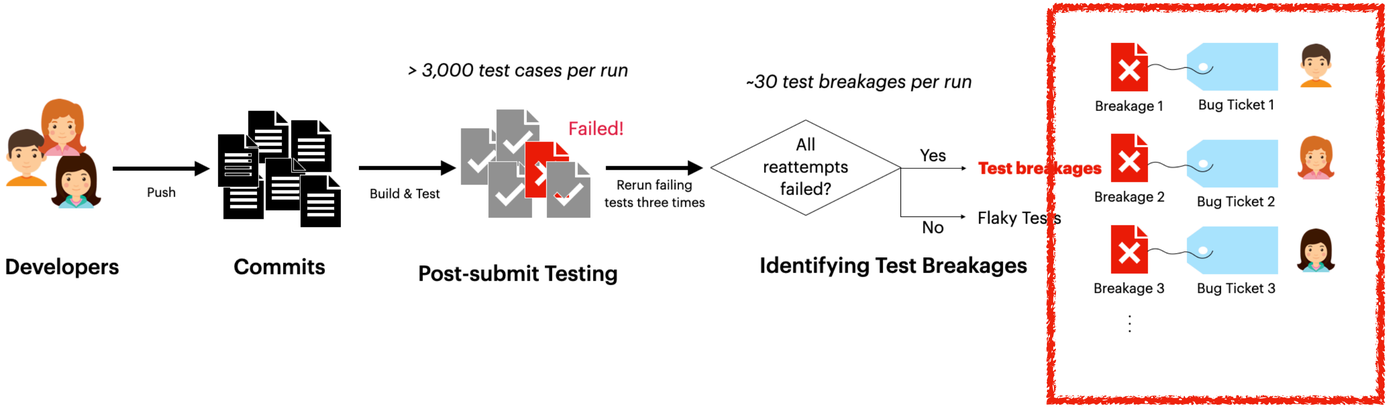
Sohn et al., ICST 2021 Industry

Dr. Jeongju Sohn

Gabin An
(PhD Candidate)

Jingun Hong
SAP Labs Korea

Dongwon Hwang
SAP Labs Korea



FL applied for **bug triage**: it can complement the static test-component relationship.

As an automated technique, this can aid the decision maker so that issues are assigned faster.

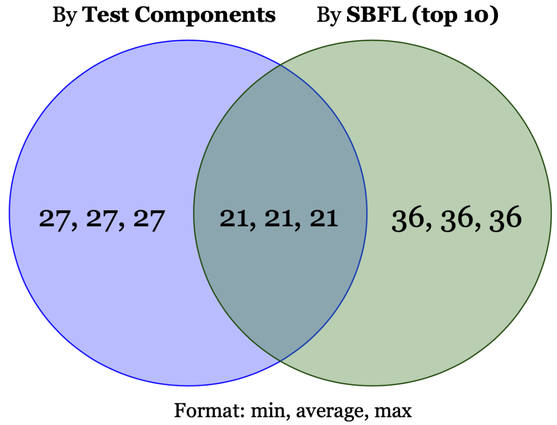
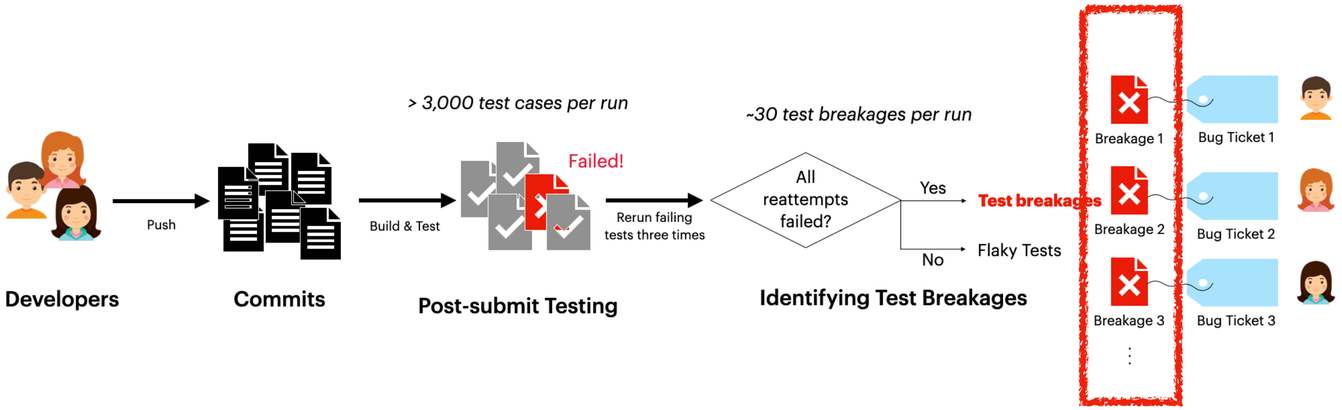


Fig. 10: Comparison between the component mapping and SBFL with voting V_D within the top 10: V_D located 9 more faults compared to the baseline. Among 57 faults localised by V_D, 36 of them are newly localised.

Similar Root Causes

Sohn et al., ICSE 2022 SEIP



Dr. Jeongju Sohn

Juyeon Yoon
(PhD Candidate)

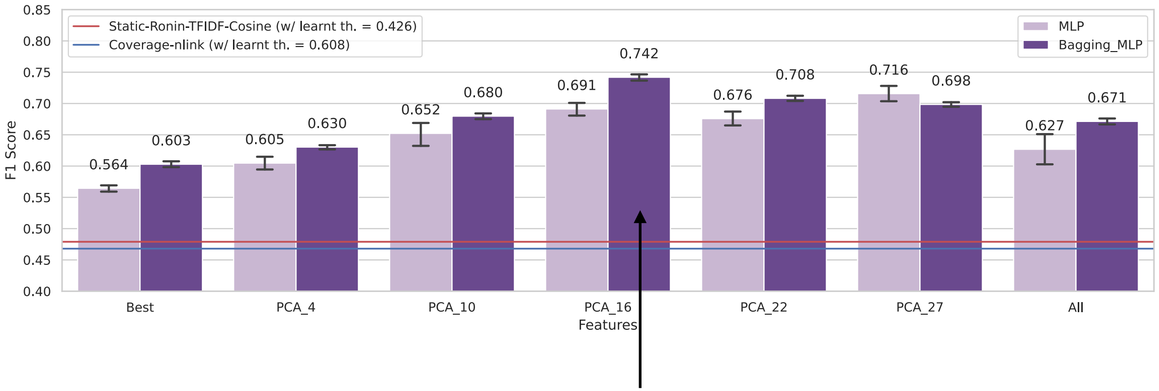
Gabin An
(PhD Candidate)

Dongwon Hwang
SAP Labs Korea

Jingun Hong
SAP Labs Korea

FL **in existence of multiple faults**: we need to isolate individual faults.

Given a pair of failures, classify whether they share the same root cause?



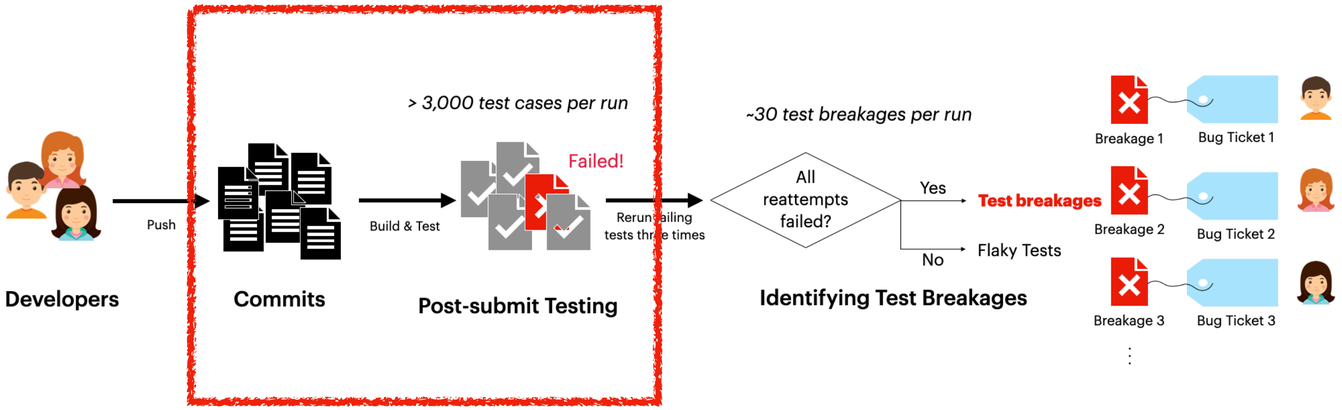
Precision 0.88, recall 0.64

Finding the Origins of Bugs

An et al., ICSE 2023

Gabin An
(PhD Candidate)

Jingun Hong
SAP Labs Korea



- We can project the FL results back to the commits to find BICs
  - Filtering out stylistic changes
  - Applying time-dependent decay
- Keep the information source minimal!

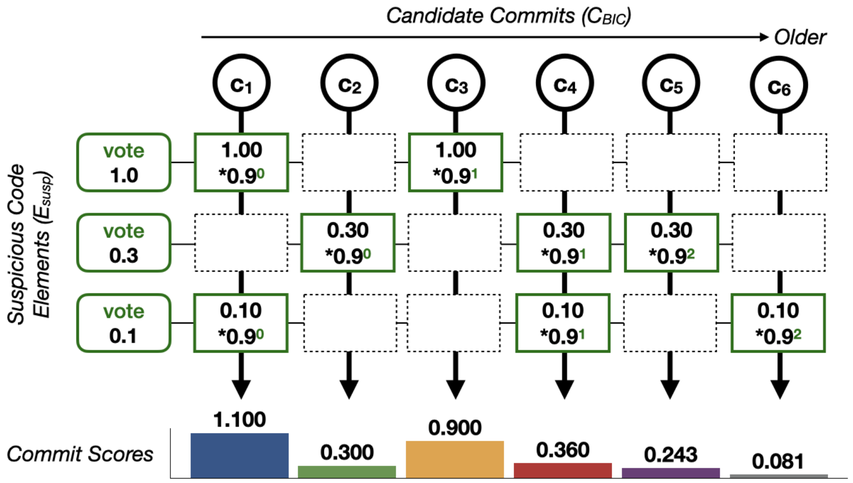


Fig. 3. Example of computing the commit scores when $\lambda = 0.1$

	MRR	Accuracy				
		@1	@2	@3	@5	@10
FONTE	**0.528**	**47** (36%)	**66** (51%)	**85** (65%)	**98** (75%)	**110** (85%)
Other Techniques (on C)						
Bug2Commit	0.155	11	18	22	25	39
FBL-BERT	0.037	1	3	5	7	10

Lessons from Industry Collaborations

Your technique may help unexpected tasks, but as long as they are helping, nothing else matters! A small delta may still be appreciated.

Simplicity

Keeping the information source of your technique to the minimum is very important.

Prof. Robert Feldt

What lies ahead?

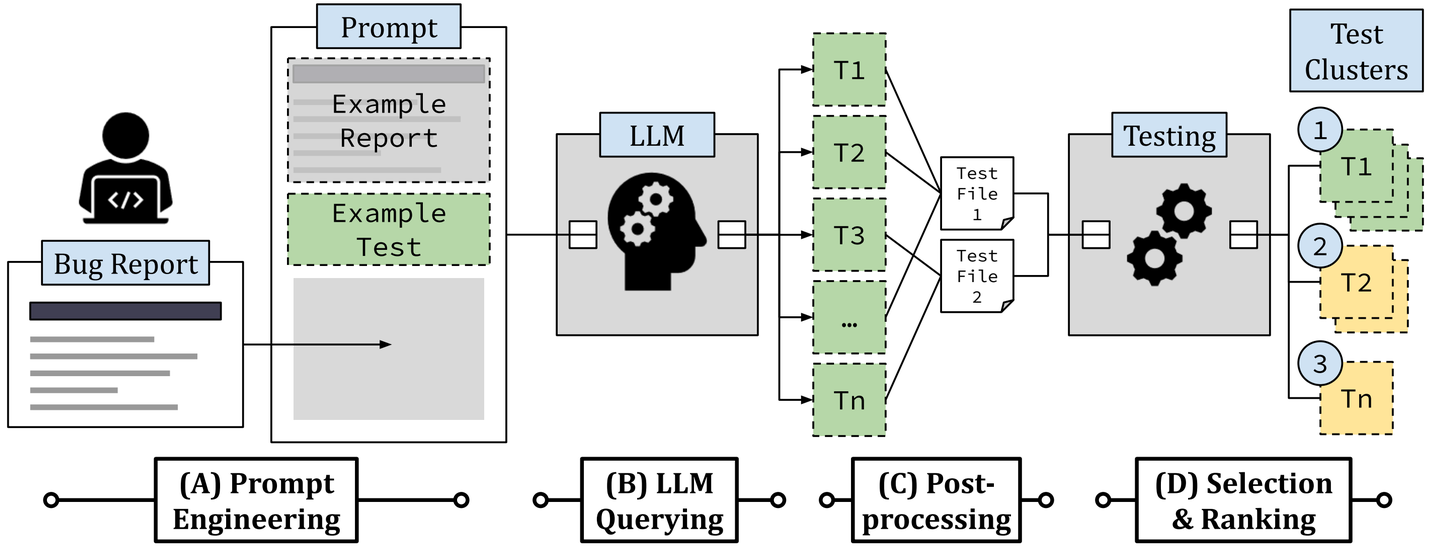
Harnessing the power of AI/ML

- Keeping the information source minimal.
- Explainability.
- Maintenance of trained models.

- Well, we interact with a single pre-trained model, at least…?
- CoT, ReAct…?
- Someone else maintains them…?

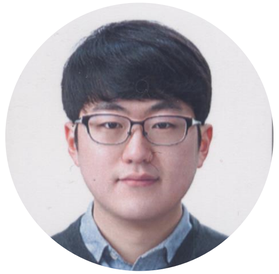
Reproducing the Bug to begin with…

Kang, Yoon & Yoo, ICSE 2023



Title	assertContainsIgnoringCase fails to compare i and I in tr_TR locale
	See org.assertj.core.internal.Strings#assertContainsIgnoringCase [url] I would suggest adding [url] verification to just ban toLowerCase(), toUpperCase() and other unsafe methods: #2664

```
public void testIssue952(){
  Locale locale = new Locale("tr", "TR");
  Locale.setDefault(locale);
  assertThat("I").as("Checking in tr_TR locale")
  .containsIgnoringCase("i");
}
```



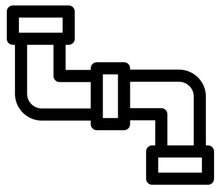
Sungmin Kang
(PhD Candidate)

Juyeon Yoon
(PhD Candidate)

PTLMs are interesting because…

Great Autocompletion

+



Dynamic Postprocessing

Differential Testing
Metamorphic Testing
Test Input Generation
…

+

CoT
Wei et al., 2022

Emergent Behaviour

+

NL/AL Duality

PAL
Gao et al., 2023
Dual Channel
Casalnuovo et al., 2020

+

ReAct
Yao et al., 2022
AutoGPT

Use of Tools

+…

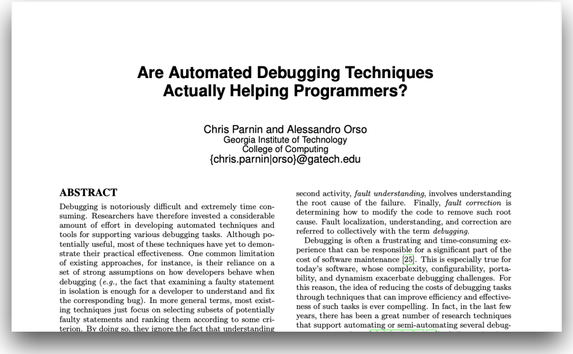
History of FL and Changes of Perspectives…

Initially conceived as an aid for humans

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

$$\frac{1}{2}\left(\frac{e_f}{e_f+n_f}+\frac{e_f}{e_f+e_p}\right)\qquad\frac{\frac{e_f}{e_f+n_f}}{\frac{e_p}{e_p+n_p}+\frac{e_f}{e_f+n_f}}$$

Then developed as an automated tool for humans



Are Automated Debugging Techniques Actually Helping Programmers?

Chris Parnin and Alessandro Orso
Georgia Institute of Technology
College of Computing
{chris.parnin|orso}@gatech.edu

ABSTRACT

Debugging is notoriously difficult and extremely time consuming. Researchers have therefore invested a considerable amount of effort in developing automated techniques and tools for supporting various debugging tasks. Although potentially useful, most of these techniques have yet to demonstrate their practical effectiveness. One common limitation of existing approaches, for instance, is their reliance on a set of strong assumptions on how developers behave when debugging (*e.g.*, the fact that examining a faulty statement in isolation is enough for a developer to understand and fix the corresponding bug). In more general terms, most existing techniques just focus on selecting subsets of potentially faulty statements and ranking them according to some criterion. By doing so, they ignore the fact that understanding

second activity, *fault understanding*, involves understanding the root cause of the failure. Finally, *fault correction* is determining how to modify the code to remove such root cause. Fault localization, understanding, and correction are referred to collectively with the term *debugging*.

Debugging is often a frustrating and time-consuming experience that can be responsible for a significant part of the cost of software maintenance [25]. This is especially true for today's software, whose complexity, configurability, portability, and dynamism exacerbate debugging challenges. For this reason, the idea of reducing the costs of debugging tasks through techniques that can improve efficiency and effectiveness of such tasks is ever compelling. In fact, in the last few years, there has been a great number of research techniques that support automating or semi-automating several debug-

Then criticized for not helping humans

PROGRAM-REPAIR.ORG

Then redeemed to help machines to debug

But now machines do debugging in a totally different way…?

?

Looking Both Ways…

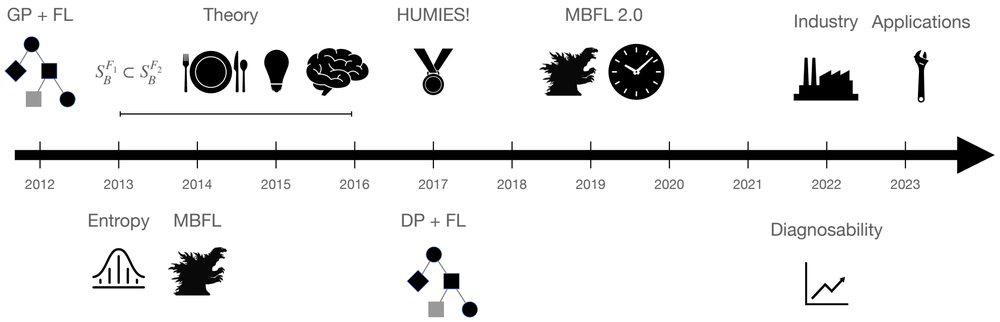
Find ways to apply what we already know to practice

- collaboration with Samsung SSDs
- Deploying FL into testing tools from SuresoftTech

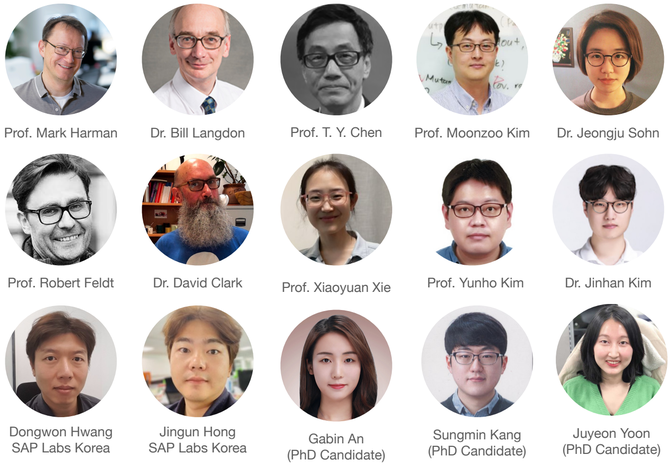
Reimagine the role of FL in the LLM era

- Many exciting work-in-progress: keep watching COINSE :)

Retrospective on My FL Work



Everything started with people.



PTLMs are interesting because…

Differential Testing
Metamorphic Testing
Test Input Generation
…

PAL
Gao et al., 2023
Dual Channel
Casalnuovo et al., 2020

Looking Both Ways…

Find ways to apply what we already know to practice

- collaboration with Samsung SSDs
- Deploying FL into testing tools from SuresoftTech

Reimagine the role of FL in the LLM era

- Many exciting work-in-progress: keep watching COINSE :)

38

Spectrum 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						

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