TU Delft

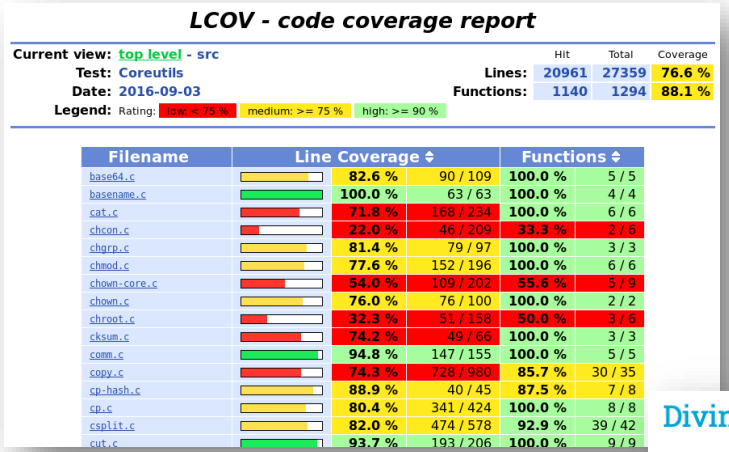
UNIVERSIDAD DE CÓRDOBA

ICST 2025

Towards Refined Code Coverage: A New Predictive Problem in Software Testing

Carolin Brandt, Aurora Ramírez

c.e.brandt@tudelft.nl aramirez@uco.es

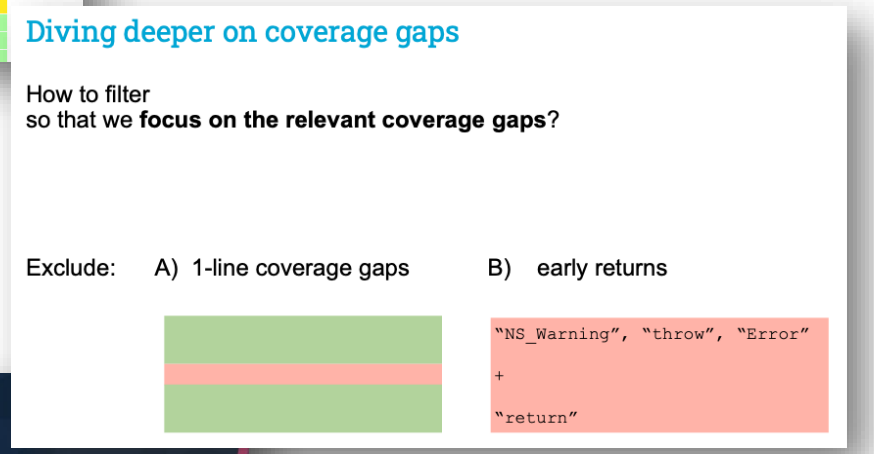


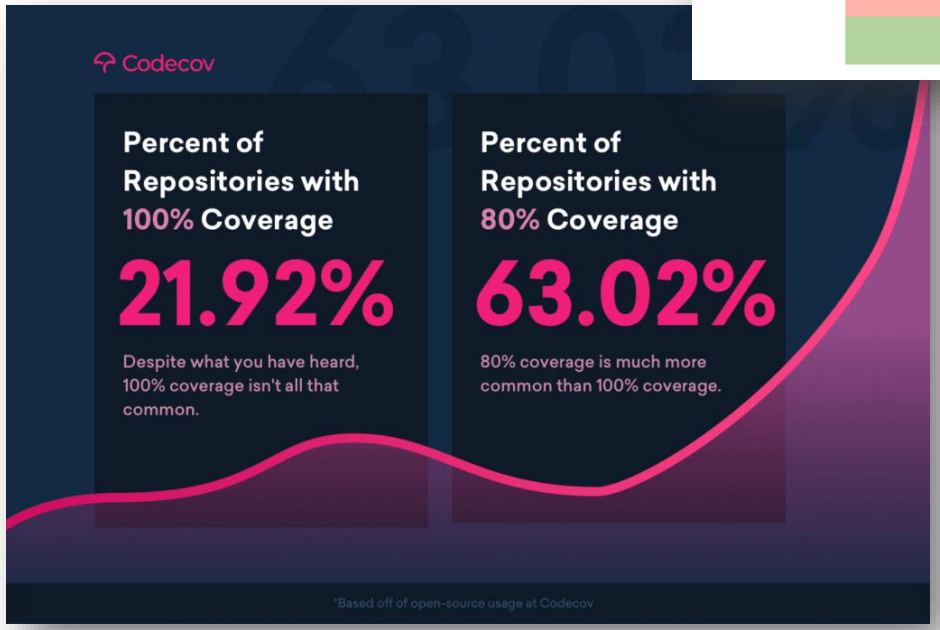
LCOV - code coverage report

Current view: top level - src
Test: Coreutils
Date: 2016-09-03
Legend: Rating: low: < 75 % medium: >= 75 % high: >= 90 %

	Hit	Total	Coverage
Lines:	20961	27359	76.6 %
Functions:	1140	1294	88.1 %

Filename	Line Coverage		Functions	
base64.c	82.6 %	90 / 109	100.0 %	5 / 5
basename.c	100.0 %	63 / 63	100.0 %	4 / 4
cat.c	71.8 %	168 / 234	100.0 %	6 / 6
chcon.c	22.0 %	46 / 209	33.3 %	2 / 6
chgrp.c	81.4 %	79 / 97	100.0 %	3 / 3
chmod.c	77.6 %	152 / 196	100.0 %	6 / 6
chown-core.c	54.0 %	109 / 202	55.6 %	5 / 9
chown.c	76.0 %	76 / 100	100.0 %	2 / 2
chroot.c	32.3 %	51 / 158	50.0 %	3 / 6
cksum.c	74.2 %	49 / 66	100.0 %	3 / 3
comm.c	94.8 %	147 / 155	100.0 %	5 / 5
copy.c	74.3 %	728 / 980	85.7 %	30 / 35
cp-hash.c	88.9 %	40 / 45	87.5 %	7 / 8
cp.c	80.4 %	341 / 424	100.0 %	8 / 8
csplit.c	82.0 %	474 / 578	92.9 %	39 / 42
cut.c	93.7 %	193 / 206	100.0 %	9 / 9





Code Coverage

- 80% realistic to aim for
- Mozilla Devs:
  "not worth the effort to add test"
- Codecov:
  - Test Quality over Test Quantity
  - Engineering Time is Finite
  - Not All Code is Equal

Pictures:
https://interrupt.memfault.com/blog/testing-vs-overhead
https://about.codecov.io/blog/the-case-against-100-code-coverage/

But which 80% should we cover?

Let’s learn from open source projects!

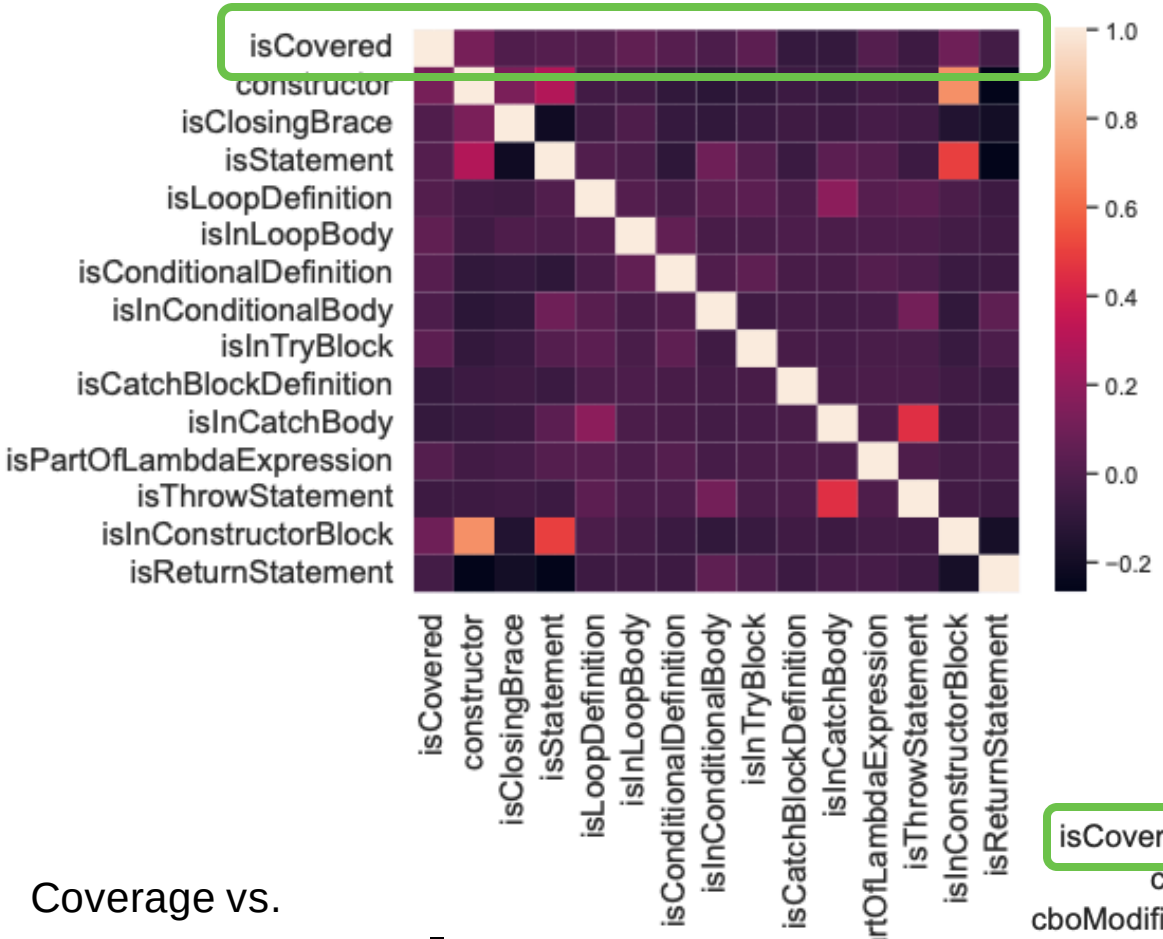
- Characterizing covered vs. not covered lines in Java projects

- Data Collection:
  - per-Line coverage (JaCoCo)
  - OO-metrics (”ck” by Mauricio Aniche)
  - AST syntax elements (/w tree-sitter)

- For this short paper: 1 Java project (allegro/hermes)

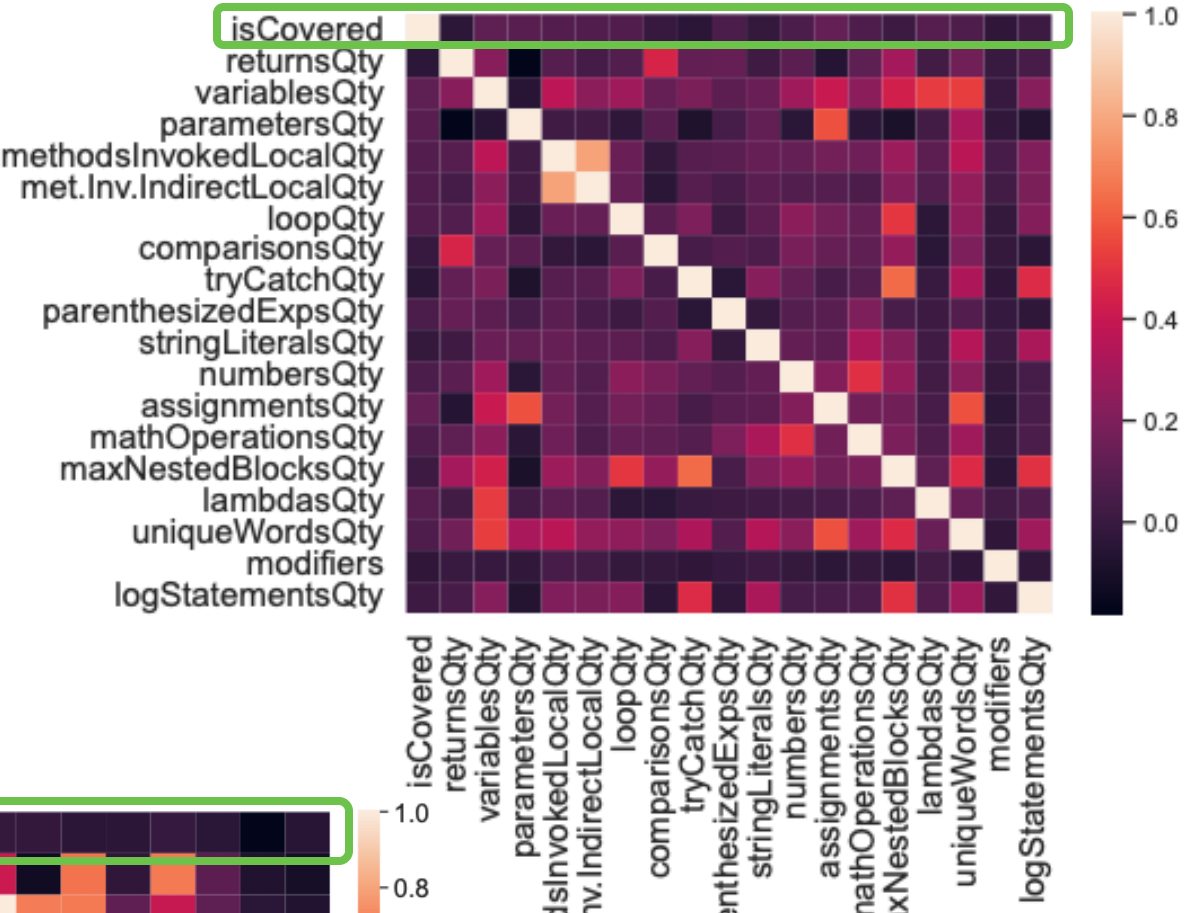
Can we apply machine learning?

- Do our metrics correlate with the target value (isCovered)?

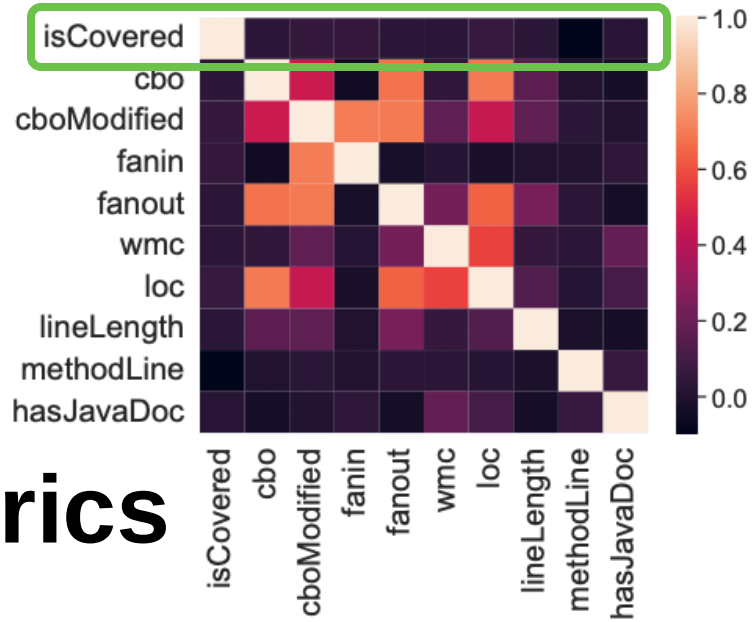


Coverage vs.

Syntax element



Counting metrics



OO-metrics

TU Delft

UNIVERSIDAD DE CÓRDOBA

Can we apply machine learning?

- No single metric correlates strongly (good!)

- Traditional ML can predict it well by combining metrics

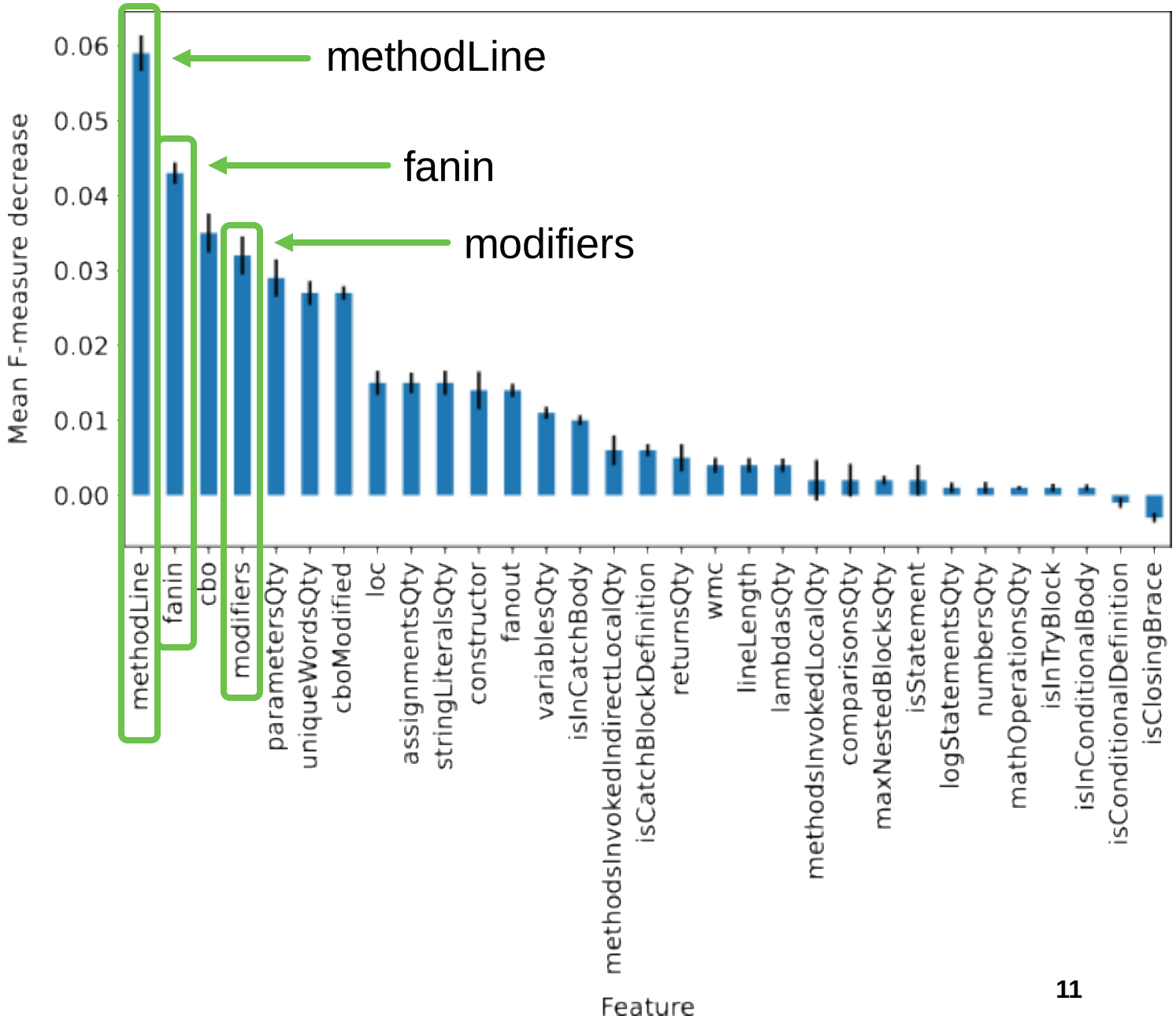
Dataset is balanced
10k not vs 18k covered

TABLE II: Classification results

Alg.	Accuracy	Bal. Accuracy	Precision	Recall	F-measure
DT	0.8930	0.8933	0.8701	0.8964	0.8830
kNN	0.7923	0.7897	0.7729	0.7633	0.7681
NB	0.5966	0.5779	0.5779	0.3888	0.4649
RF	0.9023	0.9018	0.8876	0.8968	0.8922

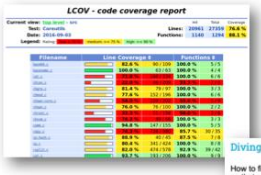
Important Features?

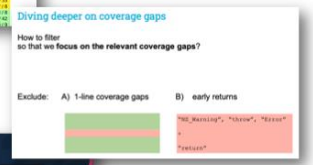
- Here:
  importance for random forest
  (strongest results)
- Interesting when sketching meaningful explanations!



What is next?

- More projects (still Java / Open Source)
  - meaningful differences? cross-project learning feasible?
- Explainable AI: Can we create meaningful explanations to developers on why this code should be tested?
- Ultimately: Build developer-driven coverage metric





Code Coverage

- 80% realistic to aim for
- Mozilla Devs: "not worth the effort to add test"
- Codecov:
  - Test Quality over Test Quantity
  - Engineering Time is Finite
  - Not All Code is Equal

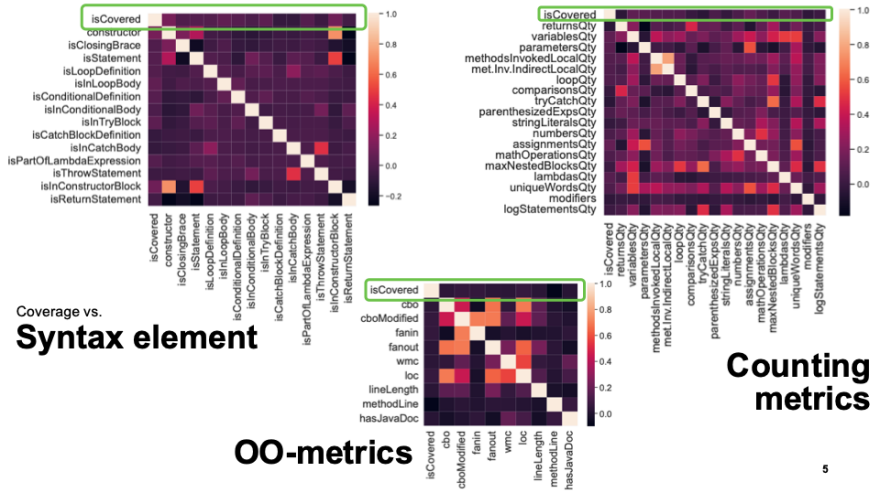
Pictures:
https://interrupt.memfault.com/blog/testing-vs-overhead
https://about.codecov.io/blog/the-case-against-100-code-coverage/

2

Let's learn from open source projects!

- Characterizing covered vs. not covered lines in Java projects
- Data Collection:
  - per-Line coverage (JaCoCo)
  - OO-metrics ("ck" by Mauricio Aniche)
  - AST syntax elements (/w tree-sitter)
- For this short paper: 1 Java project (allegro/hermes)

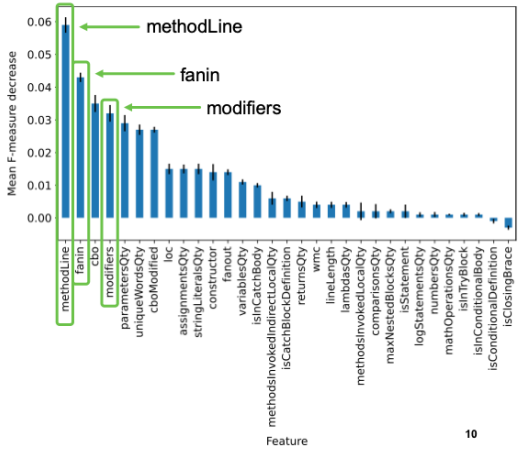
4



5

Important Features?

- Here: importance for random forest (strongest results)
- Interesting when sketching meaningful explanations!



10

c.e.brandt@tudelft.nl

Carolin Brandt, Aurora Ramírez

aramirez@uco.es

Towards Refined Code Coverage: A New Predictive Problem in Software Testing