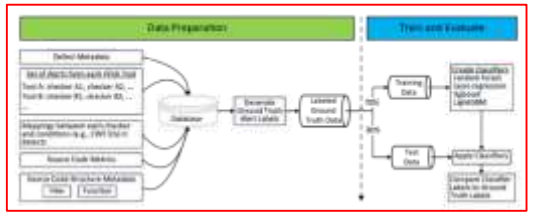
Conference on Automation of Software Test (AST)



Test Suites as a Source of Training Data for Static Analysis Classifiers

Lori Flynn

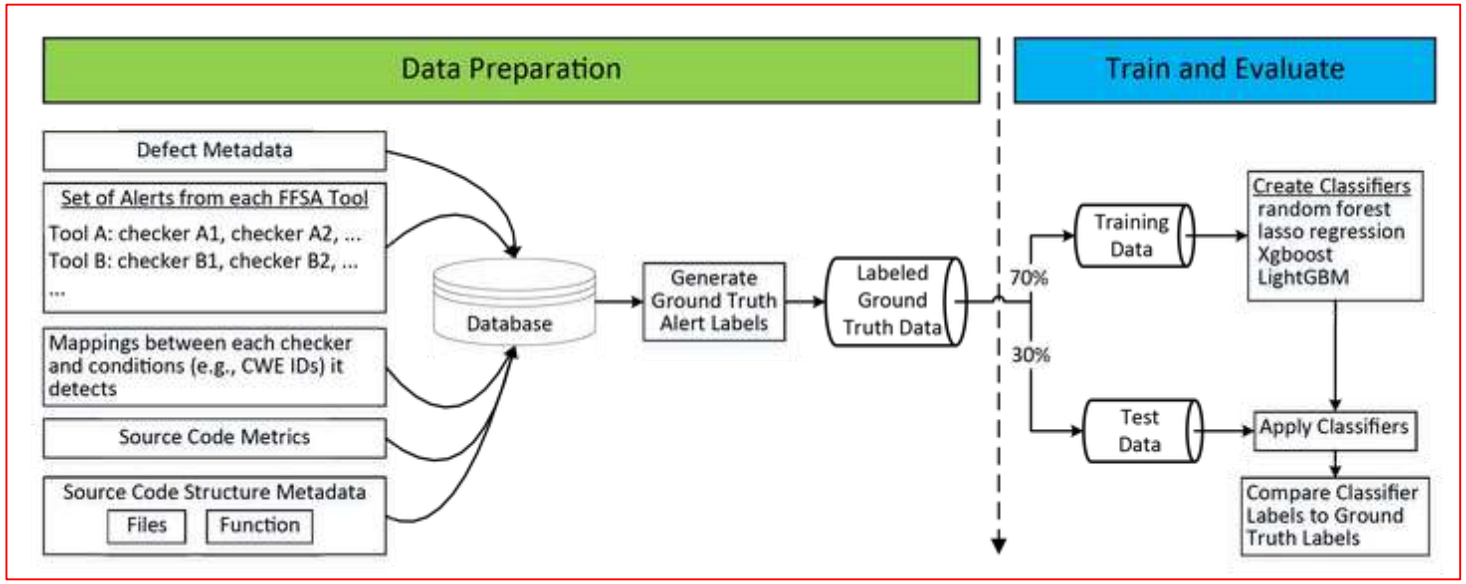
May 21, 2021

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Test Suites as a Source of Training Data for Static Analysis Classifiers

- We developed a novel method that uses test suites to automatically generate labeled data for static analysis classifiers;
- We implemented the method in a software system;
- In a case study, we generated a large quantity of labeled data for many CWE, using the Juliet C/C++ v 1.2 test suite. With that, we created 4 types of classifiers and tested them on holdout data.
- We tested speculative mapping and devised an effort-efficient part-automated method to map static analysis tools to test suite taxonomies.



Classifiers

Problem addressed: too many static analysis alerts
Solution: automate handling

Alerts that share the same line, filepath, and code flaw condition (e.g., CWE-190) map to the same **meta-alert**.

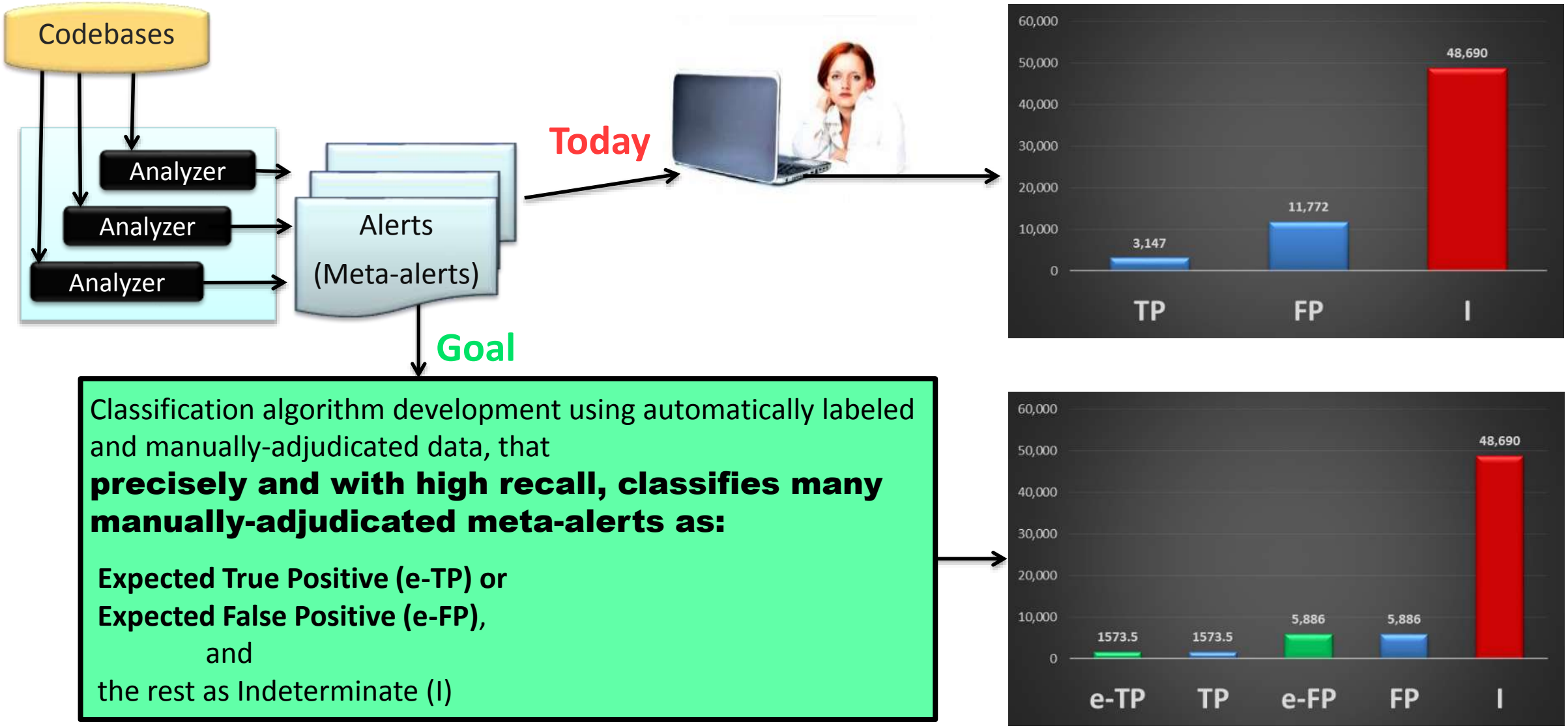


Image of woman and laptop from http://www.publicdomainpictures.net/view-image.php?image=47526&picture=woman-and-laptop "Woman And Laptop"

Related Work: Highlights

a. **Delaitre et al.** tested static analysis tool code flaw coverage. Coverage per-tool averages 20%, using multiple tools helps but compounds the too-many-alerts problem.

b. **Pugh and Ayewah** large empirical study found a mean time of 117 seconds per static analysis adjudication, analyzing data from 282 Google engineers and over 10,000 adjudications.

c. Data-rich Google (**Ruthruff et. al.)** developed classifier models with 85% accuracy predicting false positives.

d. **Cross-project classifier prediction** is an area of research developed to address insufficient labeled data within a code project. Our work involves a type of cross-project prediction, though our focus is on quickly & cheaply generating new labeled data.

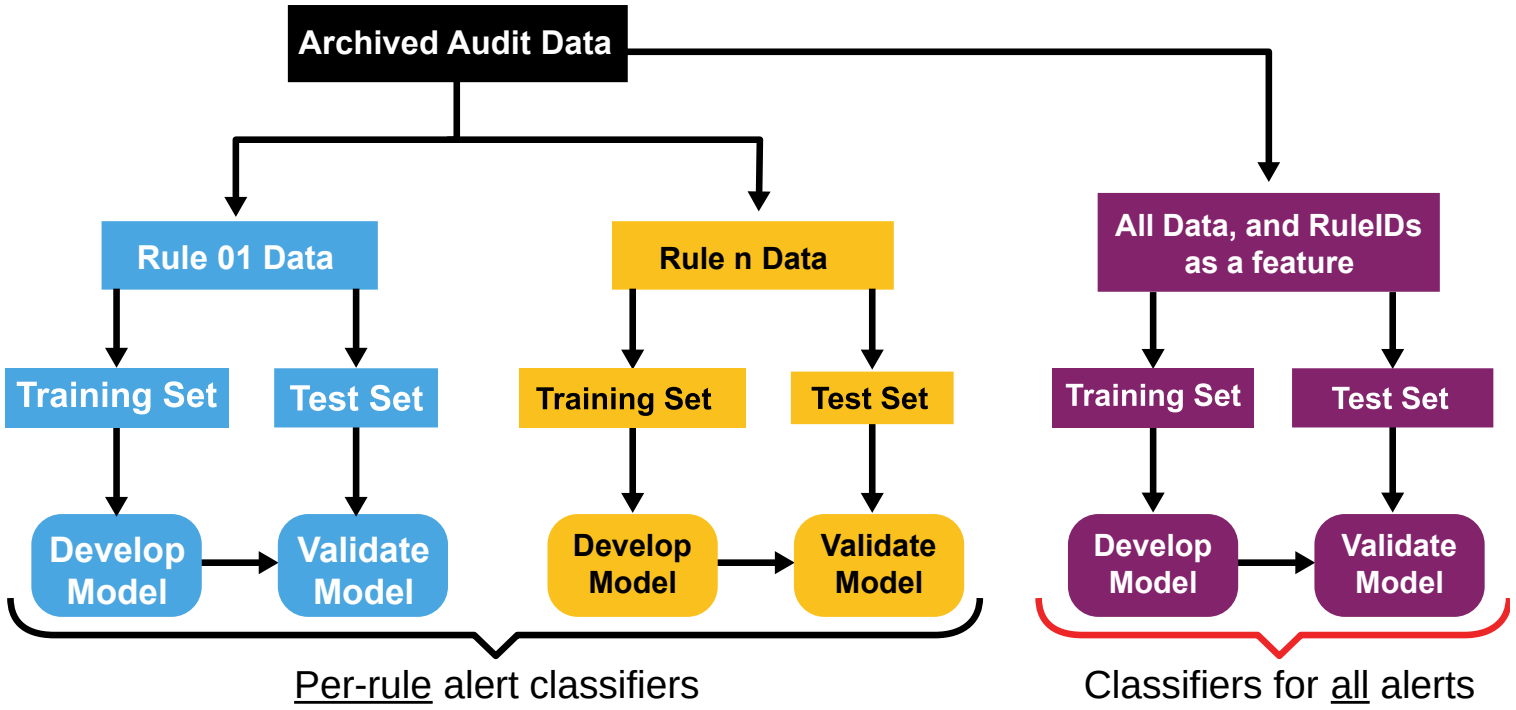
e. **Heckman and Williams**' extensive survey of methods that classify and prioritize actionable alerts, including various features, classification algorithms, and active learning. No work in their survey (nor elsewhere prior to ours) uses test suites how we do.

f. **NIST** provides over 600,000 cost-free, open-source test suite programs in its Software Reference Dataset (SARD)

Full references are in the paper.

Machine Learning with Static Analysis Audit Archives

Combined use of:
1) multiple analyzers, 2) variety of features,
3) competing classification techniques



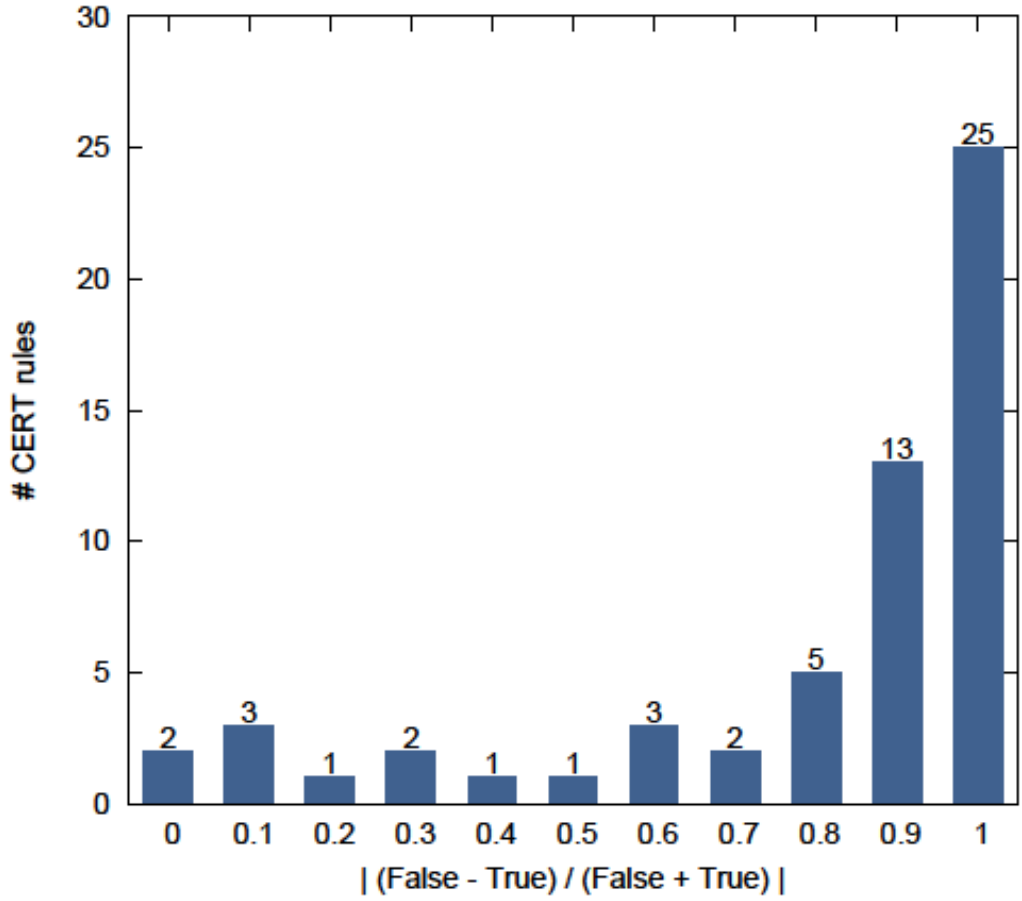
Competing Classifiers to Test
Lasso Logistic Regression (glmnet)
LightGBM
Random Forest (H2O.ai)
Extreme Gradient Boosting (XGBoost)

Some of the features used (many more)
Analysis tools used
Significant LOC
Complexity
Coupling
Cohesion
SEI coding rule

CERT- Audited Archives: Typical Issues for Classifiers

Typical code flaw coverage issue: Ten years of adjudicated static analysis archives still lacks a lot, impacting classifier performance.

- 58 CERT coding rules with 20 or more audited (labeled) alerts
- 25 rules all (or nearly all) determined one way (True or False)
- Other 324 CERT rules have little or no labeled data
- Labeled data for 158 of 382 CERT rules
- 2,487 True and 4,980 False



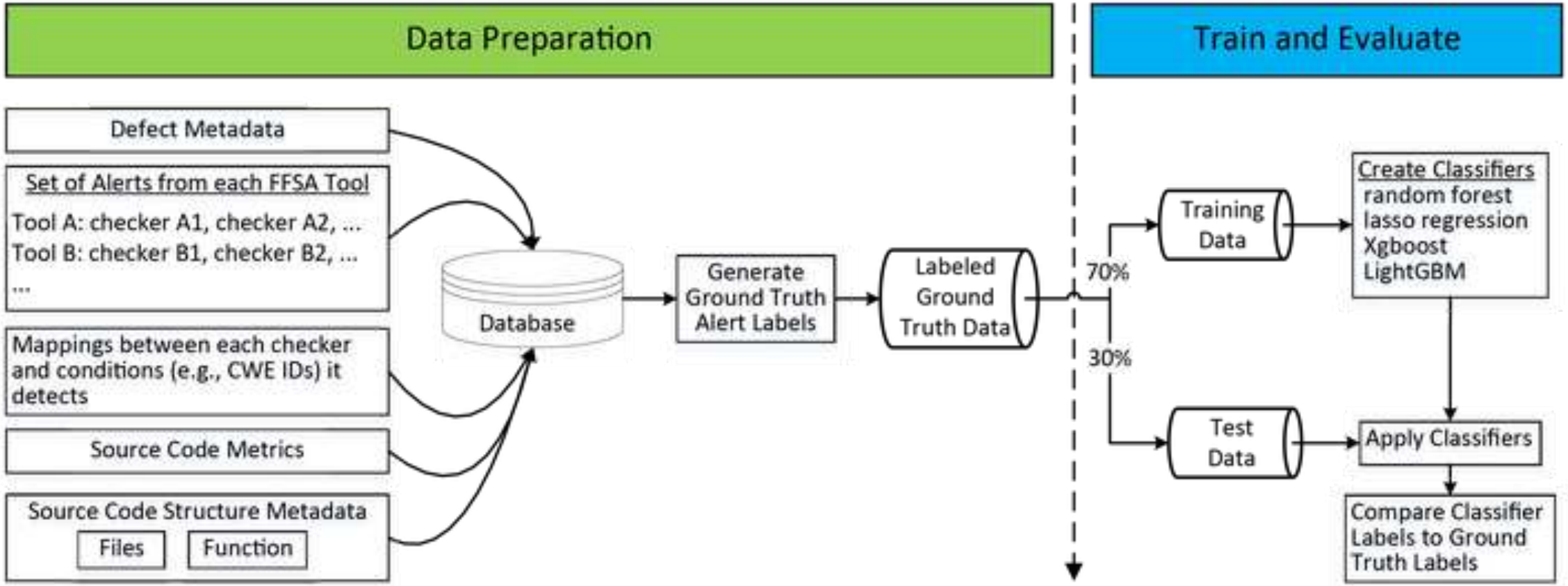
Use Test Suites to Generate Labeled Data

Problem: too few manually adjudicated meta-alerts to make precise classifiers, for many flaw types
Solution: automate adjudicating meta-alerts, using test suites

We want to rapidly, relatively cheaply generate labeled data for static analysis classifier creation and active learning.

- Over 600,000 cost-free open-source NIST SARD tests, & more elsewhere
- Developed to test tool coverage, we use test suites to make labeled data instead
- Use test suites' metadata, to quickly and automatically generate many "adjudicated" meta-alerts.
    - Use with CWE mappings to CERT rules, to generate labeled data for CERT rules
- Metrics analyses of test suite code, to get feature data
- Run static analysis tools on the code

We Made a System for Automated Static Analysis Labeling & Classifier Training



We Ran a Case Study: Juliet Test Suite C/C++ v 1.2

- Juliet (NSA CAS) 61,387 C/C++ tests
- TRUE POSITIVES: identified using Manifest file. NIST SARD XML format with flaw-present information (CWE, line, filepath)
- FALSE POSITIVES: Metadata in function name (string “GOOD”) identifies no flaw present (for one CWE)

Analysis of Juliet Test Suite: Initial CWE Results

- We automated defect identification of Juliet flaws with location **2 ways**
  - A Juliet program tells about only one type of CWE
  - Exact line defect metadata, for TPs
  - Function line spans, for FPs

Number of "**Bad**" Functions	103,376
Number of "**Good**" Functions	231,476

- Used 8 static analysis tools on Juliet programs
- Automated alert-to-defect matching
- Automated alert-mapping-to-meta-alert (meta-alerts: same line & CWE)

Lots of new data for creating classifiers

Adjudication	Meta-alerts (one per all alerts for the same line, file, & CWE)
TRUE	**36,968**
FALSE	**84,269**

- These are initial metrics (more as use more tools, STONESOUP)

Analysis of Juliet Test Suite: Initial CWE Results

Lots of new data for creating classifiers
(121,237 labeled alerts)

Adjudication	Meta-alerts (one per all alerts for the same line, file, & CWE)
TRUE	**36,968**
FALSE	**84,269**

- Big savings: A manual audit of *any* 121,237 alerts from non-test-suite programs would take an **estimated 3,940 hours** (117 seconds per adjudication [1]).
  - It's unlikely that these meta-alerts would cover many conditions/flaws covered by the Juliet test suite.
  - We needed true and false labels for classifiers.
  - **Realistically**, a hugely larger manual auditing time is required to develop equivalent data; way more than 3,940 hours would be required.
- These are initial metrics (more data as we use more tools and test suites)

[1] Nathaniel Ayewah and William Pugh. "The Google FindBugs fixit." *Proceedings of the 19th International Symposium on Software Testing and Analysis*. ACM, 2010.

Classifier Results without Speculative Mappings

Classifier	Accuracy	Precision	Recall	AUPRC
rf	0.947	0.898	0.902	0.979
lasso	0.880	0.891	0.628	0.833
xgboost	0.949	0.922	0.882	0.974
lightgbm	0.958	0.917	0.928	0.985

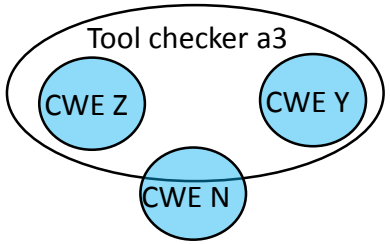
- Per-CWE and per-CERT-Rule results are included in the paper, for LightGBM.
- Other classifiers performed similarly, for per-CWE and per-CERT-rule classification

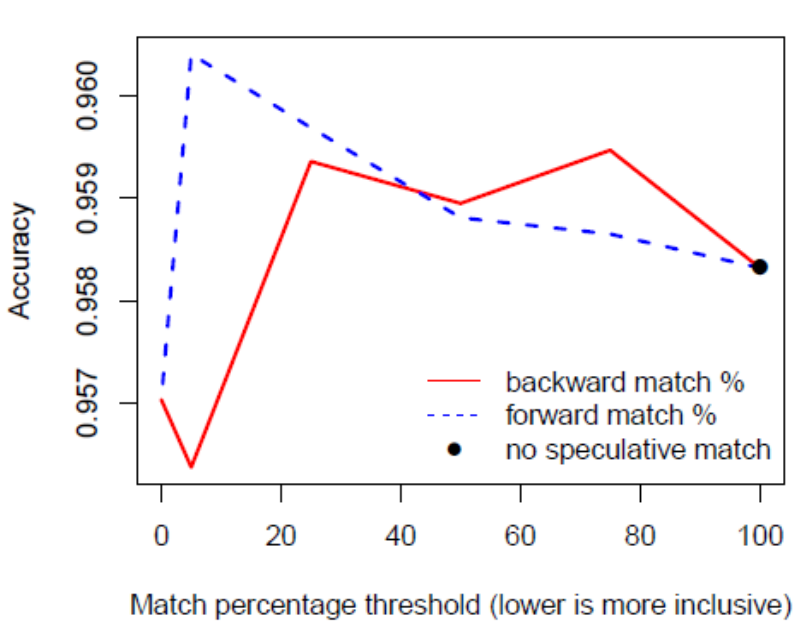
Speculative Mapping for Tools without CWE Mapping

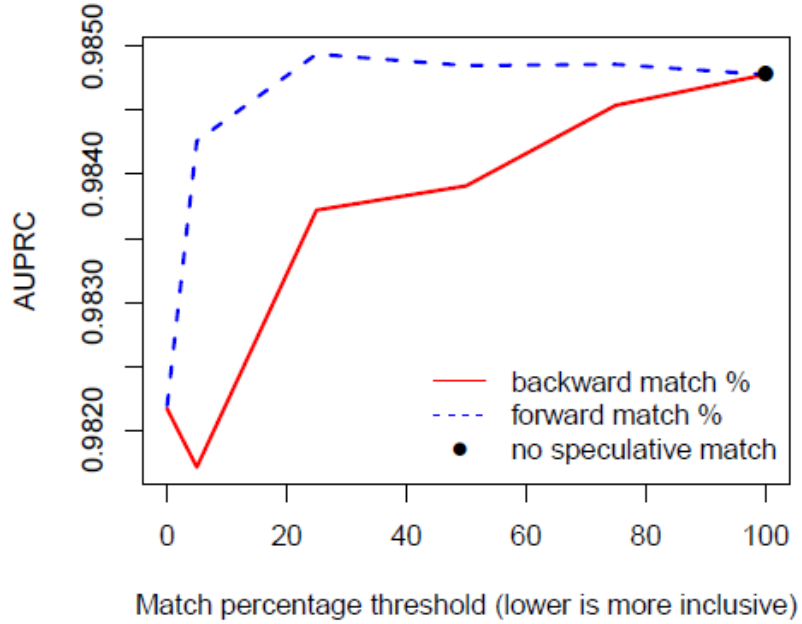
- Compared multiple mapping thresholds
- Mapped forward (checker to CWE) or backwards (CWE to checker)
- Actual relationship could be SUBSET-OF, SUPERSET-OF, PARTIAL OVERLAP, EQUALS, or COMPLETELY-DIFFERENT

Example

a3 is superset-of 2 CWEs, partial overlap with other CWE







Unexpected results

- Had expected to be slightly better with high-threshold matches
- Expected it to be worse, for low-threshold matches

Speculative Mapping for Tools without CWE Mapping

Possible source of improvement of classifiers for severely under-represented conditions

- At most inclusive, there are 102 CWE in training data, versus only 82 in the non-speculative training data

Developed effort-saving method for mixing automation with manual mapping

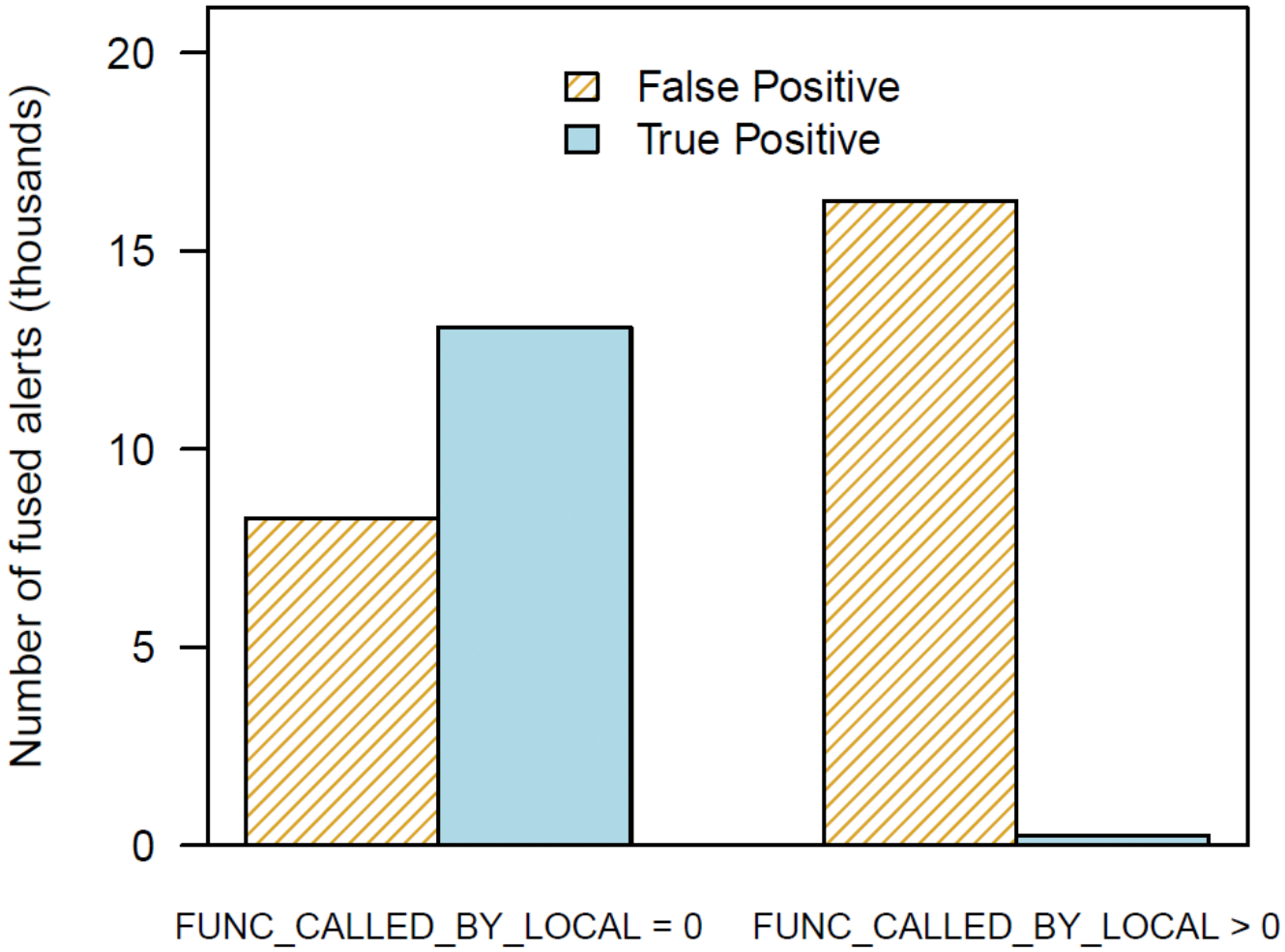
a. First, speculative mapping to generate possible matches
b. Second, manually evaluate the possible match

* Approximately 700 CWE
* Static analysis tools may have hundreds of checker IDs

Too much effort, to validate ~70,000 potential mappings manually

With 3 speculative mappings per checker, still **save factor of 233 less effort to do the mappings**

Feature Importance: Function Called by Local Function



Alerts for a function called by a local function were far more likely to be false positive.

FUNC_CALLED_BY_LOCAL had the highest information gain.

RC_Data Open Dataset for Static Analysis Classification Research

- We published the open-source data used to develop classifiers and make mappings in this work, in the RC_Data dataset: https://wiki.sei.cmu.edu/confluence/display/seccode/Open+Dataset+RC_Data+for+Classifier+Research
  - RC_Data includes Juliet Java test suite data not part of this paper's case study. It was created using the auto-labeling system presented in the paper.
  - The proprietary static analysis tools' alerts and checker mappings are not in RC_Data, due to license restrictions

- More info in11/2/20 **SEI blog** "A Public Repository of Data for Static-Analysis Classification Research" (Flynn) https://insights.sei.cmu.edu/sei_blog/2020/11/a-public-repository-of-data-for-meta-alert-classification-research.htm

Summary and Next Steps

- We developed a novel method that uses test suites to automatically generate labeled data for static analysis classifiers;
- We implemented the method in a software system;
- In a case study, we generated a large quantity of labeled data for many CWE, using the Juliet C/C++ v 1.2 test suite. With that, we created 4 types of classifiers and tested them on holdout data.
- We also tested speculative mapping and devised an effort-efficient part-automated method to map static analysis tools to test suite taxonomies.

Next steps (already begun!):

- work with test suites involving naturally-developed and larger codebases
- Attempt to improve classification by using active learning strategically combining manually-adjudicated alerts on natural code with test suite auto-labeled alerts.