Static Analysis Classification, SCAIFE, and Your Org

Feb 22, 2022

Dr. Lori Flynn

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.

DM22-0188

Static Analysis Classification, SCAIFE, and Your Org

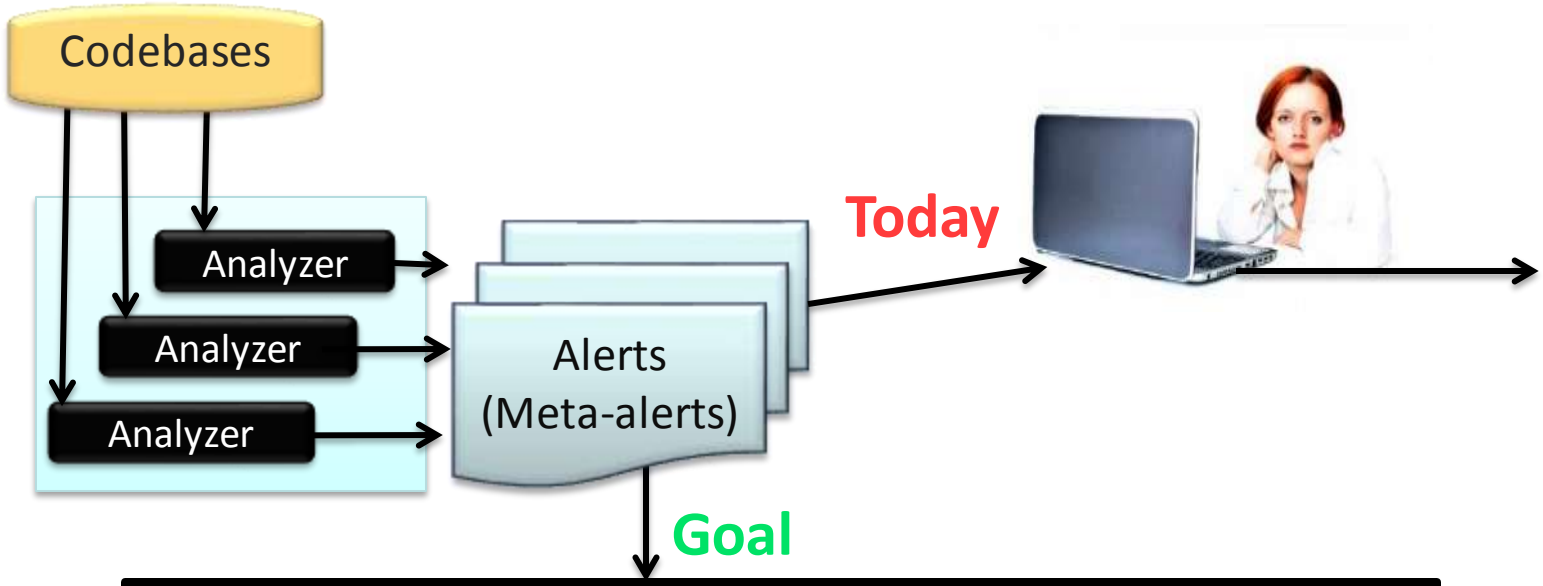
Today, we'll talk about:

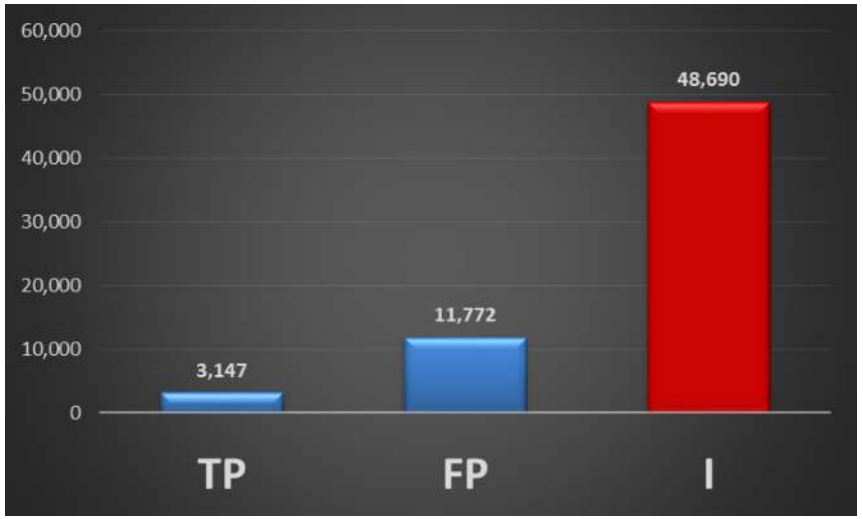
- Static analysis classification
- Particular research features in SCAIFE that you might want to use / take
  - ➢ I will demo some of them!
- The issues you are facing, that you hope static analysis classification might help with

SCAIFE
classification system

Problem: too many static analysis alerts
Solution: automate handling

A **meta-alert** is a static analysis result for a particular line number, filepath, and code flaw condition (e.g., CWE-190).





CI-optional systems that
precisely and with high recall, classify at least as many manually-adjudicated meta-alerts as:

Expected True Positive (e-TP) or
Expected False Positive (e-FP),
and
the rest as Indeterminate (I)

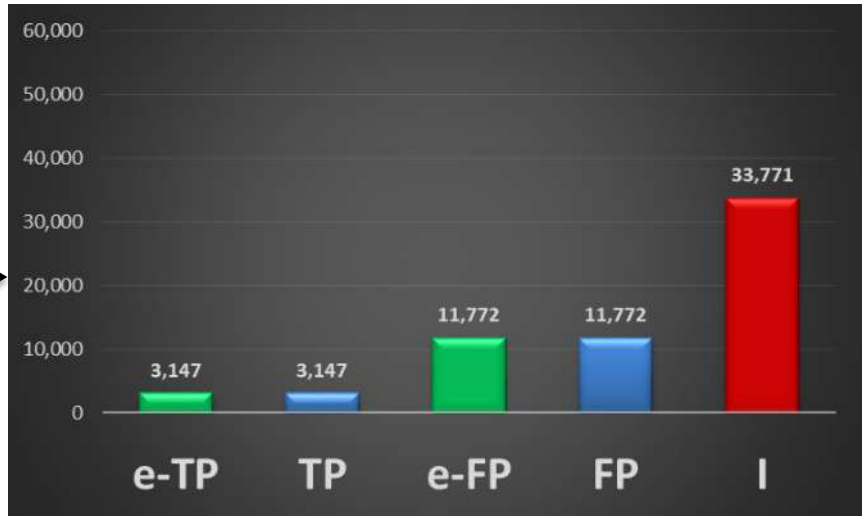


Image of woman and laptop from http://www.publicdomainpictures.net/view-image.php?image=47526&picture=woman-and-laptop "Woman And Laptop"

Research Roadmap: SA Classification

SA classification research: Existing topic for decades. For example:

- 85% accuracy in [Ruthruff]: FindBugs, Logistic Regression, adaptive using code-fix decisions
- 62% precision for top 50 alerts, using locality, flaw type, code version number [Williams]
- 89% precision for *jdom*. Machine learning using Weka tool [Heckman B]
- 9% improvement in cross-project defect prediction, using semantic features [Wang]
- Adaptive algorithms
  - [Heckman] ARM: 81% true positive alerts after investigating only 20% of alerts (vs. avg. of 50 random orderings found 22% after investigating 20%)
  - [Kremenec] Feedback-Rank: 2-8x improvement of performance ratio over random

[Baishakhi] Ray, Baishakhi, et al. "On the naturalness of buggy code." ICSE, 2016.
[Heckman] Heckman, Sarah Smith. "Adaptively ranking alerts generated from automated static analysis.“, Crossroads, 2007.
[Heckman B] S. Heckman, L. Williams, On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques, Empirical Software Engineering and Measurement, 2008, pp. 41–50.
[Hoole] A. Hoole et al. "Improving vulnerability detection measurement" ICSE, 2016.
[Kim] S. Kim, M.D. Ernst, Prioritizing warning categories by analyzing software history, International Workshop on Mining Software Repositories, 2007, p. 27.
[Kremenec] T. Kremenek, K. Ashcraft, J. Yang, D. Engler, Correlation exploitation in error ranking, FSE, 2004, pp.83–93.
[Wang] Wang, Song, Taiyue Liu, and Lin Tan. "Automatically learning semantic features for defect prediction." ICSE, 2016.
[Williams] C. Williams et al., Automatic mining of source code repositories to improve bug finding techniques, Trans. SE 2005.
[Ruthruff] J. Ruthruff et al. "Predicting accurate and actionable static analysis warnings: an experimental approach." ICSE, 2008.
[Zhang] Zhang, Feng, et al. "Cross-project defect prediction using a connectivity-based unsupervised classifier." ICSE, 2016.

Research Roadmap: SA Classification

Current SCAIFE prototype: features and capabilities from multiple SA classification research projects at SEI

Take from SCAIFE: Organizations could tailor a plugin (e.g., CodeDX, SonarQube, etc.) for classifiers, features, other SA frameworks, other system designs and test with datasets.

FY16-19 SA Classification Research Detail

FY16

- Issue addressed: classifier accuracy
- Novel approach: use **multiple static analysis tools as features**
- Result: increased accuracy

FY17

- Issues addressed: **data quality**, **too little labeled data** for accurate classifiers for some conditions (e.g., CWEs, coding rules)
- Novel approach: **audit rules+lexicon; use test suites to automate the production of labeled** (true/false) **meta-alert data* for many conditions**
- Result: high precision for more conditions

FY18-19

- Issue addressed: **little use of automated meta-alert classifier technology** (requires $$, data, experts)
- Novel approach: **develop an extensible architecture with a novel test-suite data method**
- Result: enabled **wider use of classifiers (with less $$, data, experts)** with an extensible architecture, API, software to instantiate architecture, and adaptive heuristic research

* By the end of FY18, ~38K new labeled (T/F) meta-alerts from eight SA tools on the Juliet test suite (vs. ~7K from CERT audit archives over 10 years)

Goal: Enable **practical** automated classification for more secure software and lower cost/effort.

FY20-21 SA Classification Research Detail

- Issue addressed: It takes too much time to adjudicate static analysis results during continuous integration (CI).
- Novel approach: Develop modular CI-enabled design using SA **classifiers** and different **cascading,** enable performance experiments
- Results (highlights)
  - Designed, implemented, and tested CI-SCAIFE system integration
  - Both diff-based and precise cascading developed, tested
  - Enhanced performance metrics collection and auto-setup of experiments
  - Some experiment results and collaborator testing
  - Multiple SCAIFE System, SCAIFE API, and SCAIFE UI Module (SCALe) releases

Goal: Enable **practical** automated classification for more secure software and lower cost/effort.

SCAIFE features you may want to use

- **Static analysis automated classification**: make predictions about confidence a warning is true or false
- **Auto-label test suite codebases using NIST SARD manifests**: generate more labeled static analysis data quickly, for a wide variety of code flaws
- **Gather classification prediction performance data at key points and periodically**. This uses labeled data, making predictions on holdout labeled data. (the existing adjudicated static analysis results, manual and test suite)
- Depending on the performance, **use the classification predictions to order and/or filter not-yet-adjudicated static analysis results**

Static analysis training you might be interested in

You might be interested in **static analysis training** we've developed (or possibly tailoring training to your needs)**:**

- Using rules and lexicon for consistent adjudication of static analysis results
- CERT rules and CWE
- Using SCAIFE
- Using SCALe

SCAIFE Static Analysis Classifiers Detail

Designed for use by machine learning novices, with settings that can be tweaked by experts

Labeled static analysis meta-alerts used to create classifiers:

- Manually adjudicated meta-alerts (true positive, false positive)
- Test suites (e.g., Juliet): SCAIFE automatically adjudicates meta-alerts
- User chooses labeled data sets, classifier, active learning, and other options

Modular ability to add different types of classifiers, active learning, and hyper-parameter optimization methods.

Built-in options:

- Classifiers: XGBoost, Random Forest, LightGBM
- Active Learning (adaptive heuristics): Similarities, K-Nearest Neighbors, and Label Propagation
- Hyper-parameter optimization: Bayesian Optimization

SCAIFE Classification System

Designed to be used in a wide variety of systems, with many other tools

Full SCAIFE system includes all 5 modules

Modular system designed to work with different user interfaces and static analysis tools

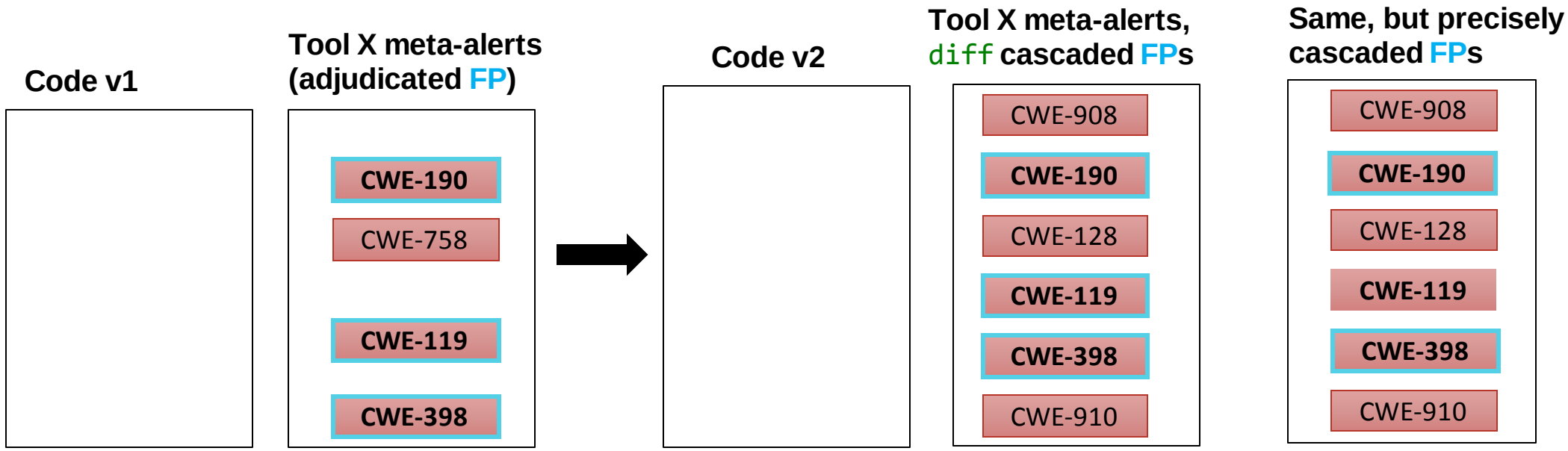
- SARIF static analysis format
- SCARF format (DHS SWAMP)
- Various tools and versions, with standard method for adding new tools

Use SCAIFE for a single code version or a codebase in a CI system

- CI system: updates to code and static analysis for the new code version

SCAIFE: 2 Types of Meta-Alert Adjudication Cascading

- For code versions 1 and 2, can a manual adjudication (e.g., true, false) for a meta-alert from v1 be applied to a meta-alert for code v2?
- Imprecise cascading happens on a per-file analysis and uses regular expression and/or line numbers.
- Precise cascading means analysis across a whole program using control flow, data flow, and type flow.



SCAIFE Architecture

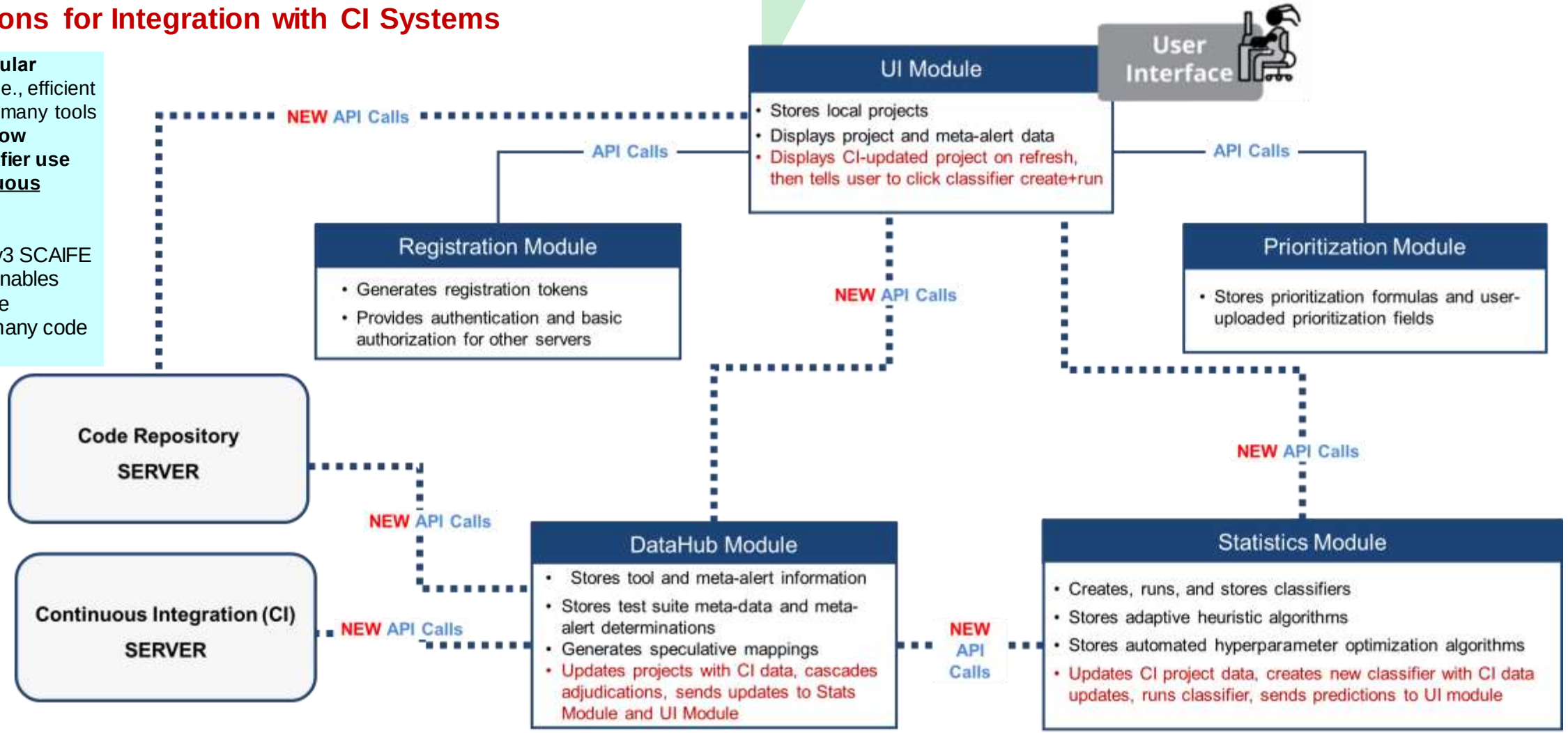
Modifications for Integration with CI Systems

SCAIFE's modular architecture (i.e., efficient integration with many tools and systems) **now enables classifier use <u>during continuous integration.</u>**

The OpenAPI v3 SCAIFE API definition enables automated code generation in many code languages

Any static analysis tool can instantiate APIs to become a UI Module. For example

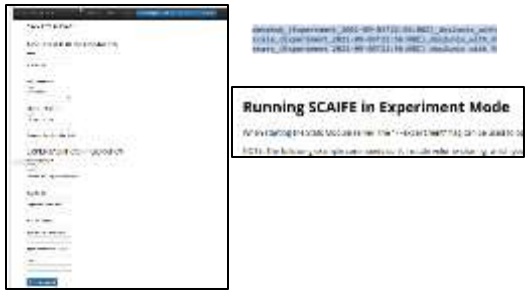
- SEI SCALe
- DHS SWAMP
- CCDC C5ISR SwAT
- Other aggregator tools
- Single static analysis tools

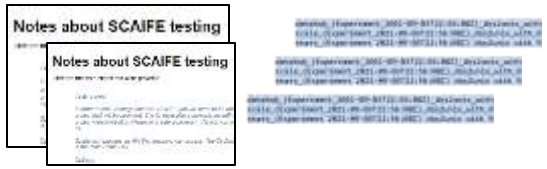


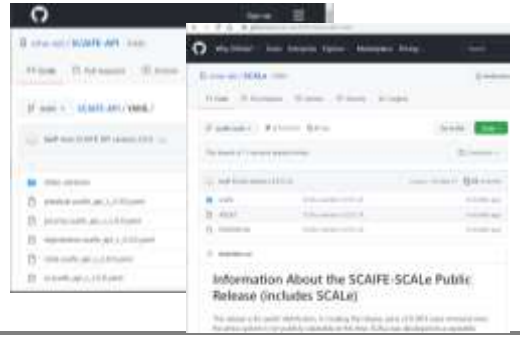
Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

FY21 Select Artifacts

scaife.online.3.0.0.tar.gz







- **Paper** "Test suites as a source of training data for static analysis alert classifiers" by Lori Flynn, William Snavely, and Zachary Kurtz to ICSE-associated Conference on Automation of Software Test (AST) 2021 https://conf.researchr.org/home/icse-2021/ast-2021 and video https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=737855
- **SCAIFE v3 release**
  - Contains much-enhanced performance metrics collection:
    - Experiment mode: auto-setup experiments with configuration files + datasets, collect metrics, auto-end, and export data
    - Metrics include (among others): classifier precision and recall, counts of adjudicated vs. high-confidence predicted, and key step latencies, CPU use (max, avg), bandwidth use (max, avg), memory use (max, avg)
  - Java test suites now fully usable by SCAIFE
- **SCAIFE release test results and analysis:**
  - SEI CI experts did the CI demo, provided feedback (Lyndsi Hughes and Joe Sible)
  - External collaborator testing and feedback
- **GitHub publications of SCAIFE API** https://github.com/cmu-sei/SCAIFE-API
- **GitHub publications of SCAIFE UI module** (SCALe) **code** at https://github.com/cmu-sei/SCALe/tree/scaife-scale

Current Work

- Writing research papers
- Gathering and analyzing test data.
- Interested potential testers welcome!

SCAIFE-ACR Integration

Integrated Static Analysis Classification and Automated Code Repair for CI

Problem: DoD organizations that develop code or analyze code security need to make code more secure, with as little costly manual effort as possible. Automated code repair (ACR) tools can fix some code flaws, and automated SA classifiers can save manual work adjudicating static analysis (SA) results, but they may not work well together as-is.

Solution: A system that can modularly incorporate a wide variety of SA classifiers and ACRs that increases the percent of high-severity SA results[1] addressed automatically after applying ACR, designed for CI. "Automatically" means "automatically repaired" or "automatically classified with confidence 70% or greater" (provided that the classifier has a precision and recall 70% or greater).

Approach: Building on what we've learned and the tools from our previous Line-funded projects (SEI ACR[4] and SCAIFE[2]), we will develop a modularly integrated SCAIFE-ACR system for CI, then use it to measure impact of SEI ACR on the percent of high-severity SA results addressed automatically. We test it with open-source code and collaborators use it in their systems on their code. Also, novel use of ACR fix data to improve classifier predictions, and measure effectiveness with classifier precision and recall comparisons.

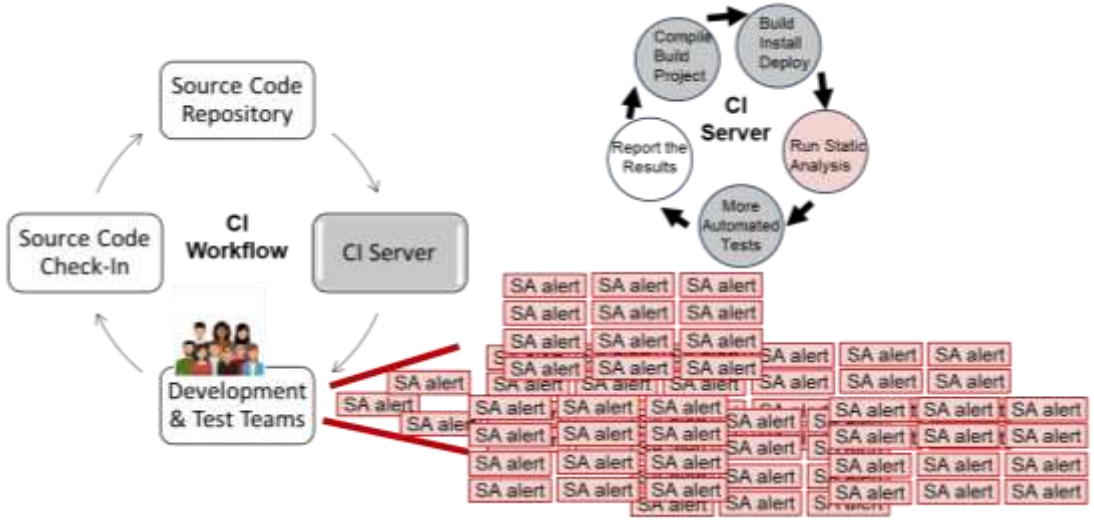
1. **High-severity SA results**: warnings for top 25 dangerous CWE and CERT coding rules with severity level 3
2. **SCAIFE**: modular static analysis classification system with GUI developed by L. Flynn's SEI projects
3. **ACR**: uses a semantic representation of code to fix code flaws
4. **SEI ACR**: ACR tooling for memory safety in C developed by W. Klieber's SEI projects

Auto-Labeled ACR Fixes

1. An ACR fix is made to the code (was version 1, now v2 in new branch)

ACR possible?	Priority	Auto-Repair?	Manual Adjudication	Classifier Confidence True (%)	Condition
yes	8890	YES			CWE-190
yes	8889	NO			INT31-C
yes	8888	YES			CWE-191
yes	8887	YES			CWE-79
yes	8886	YES			CWE-787
yes	8885	YES			CWE-125

2. The ACR code fixes (v2) are committed to the remote repository. Then, the CI server builds and tests the code, and sends results back to the development+test team and to SCAIFE.



3. SCAIFE fuses SA results into meta-alerts. If last code push is an auto-repair, SCAIFE checks **if a meta-alert for the repaired condition re-appears on matched lines* of repaired code**. If yes, it **auto-labels the meta-alert False *if* fix marked 'reliable' by the ACR**. New feature "auto-repair" for classifier. No ACR auto-labels True.)

SEI ACR does not prove the code v1 meta-alert was True, but it fixes many memory safety violations

ACR possible?	Priority	Auto-Repair?	Manual Adjudication	Classifier Confidence True (%)	Condition	Auto-Adjudication?	Line	Filepath
yes	8885	YES		88	CWE-125	N/A	10	dir4/FileX

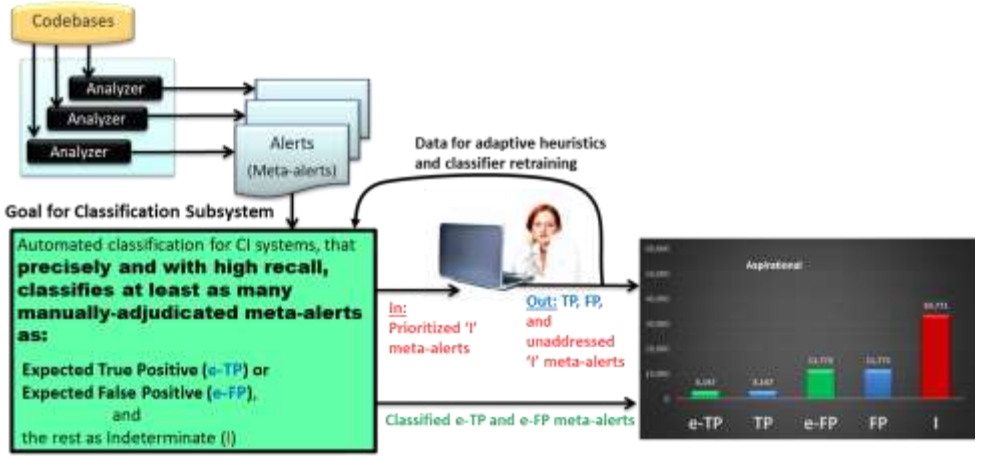
Code v1, ACR-fixed meta-alert

N/A	7000	PREVIOUS		80	CWE-125	FALSE	10	dir4/FileX

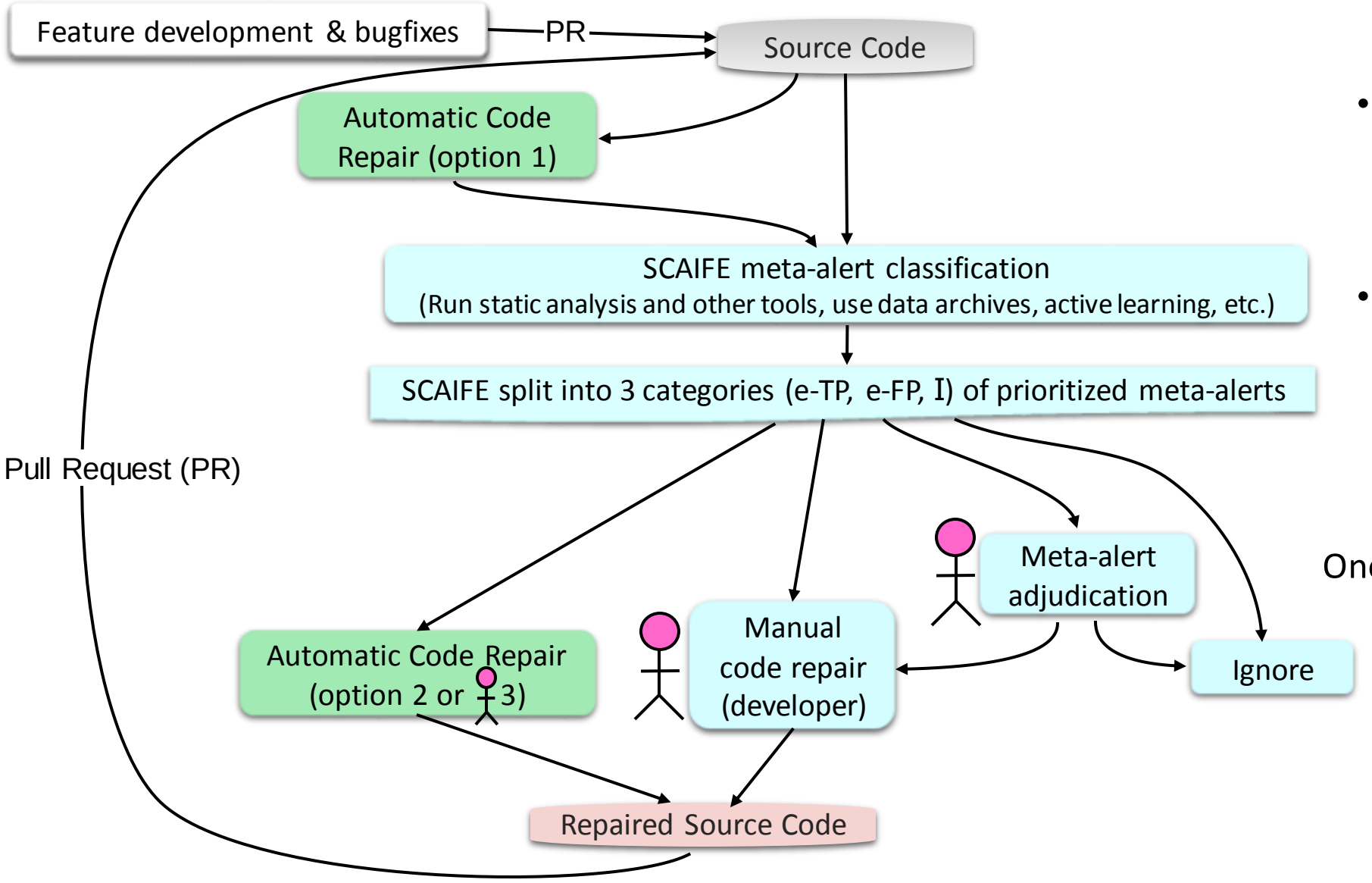
Code v2, same location and condition: meta-alert auto-labeled FALSE

** Lines may be matched using POSIX diff program, possibly enhanced with extra matching information related to the ACR system. E.g., a memory access that previously took 1 line might be expanded by the ACR to take 3 lines in one file plus a new function in another file, and any of those locations would count as a match.*

4. The new labeled data (meta-alert auto-labeled false and associated data) is used to improve the SA classifier predictions for all remaining not-yet-adjudicated meta-alerts, using **adaptive heuristics and/or occasional classifier retraining.**



ACR & SCAIFE Meta-Alert Classification

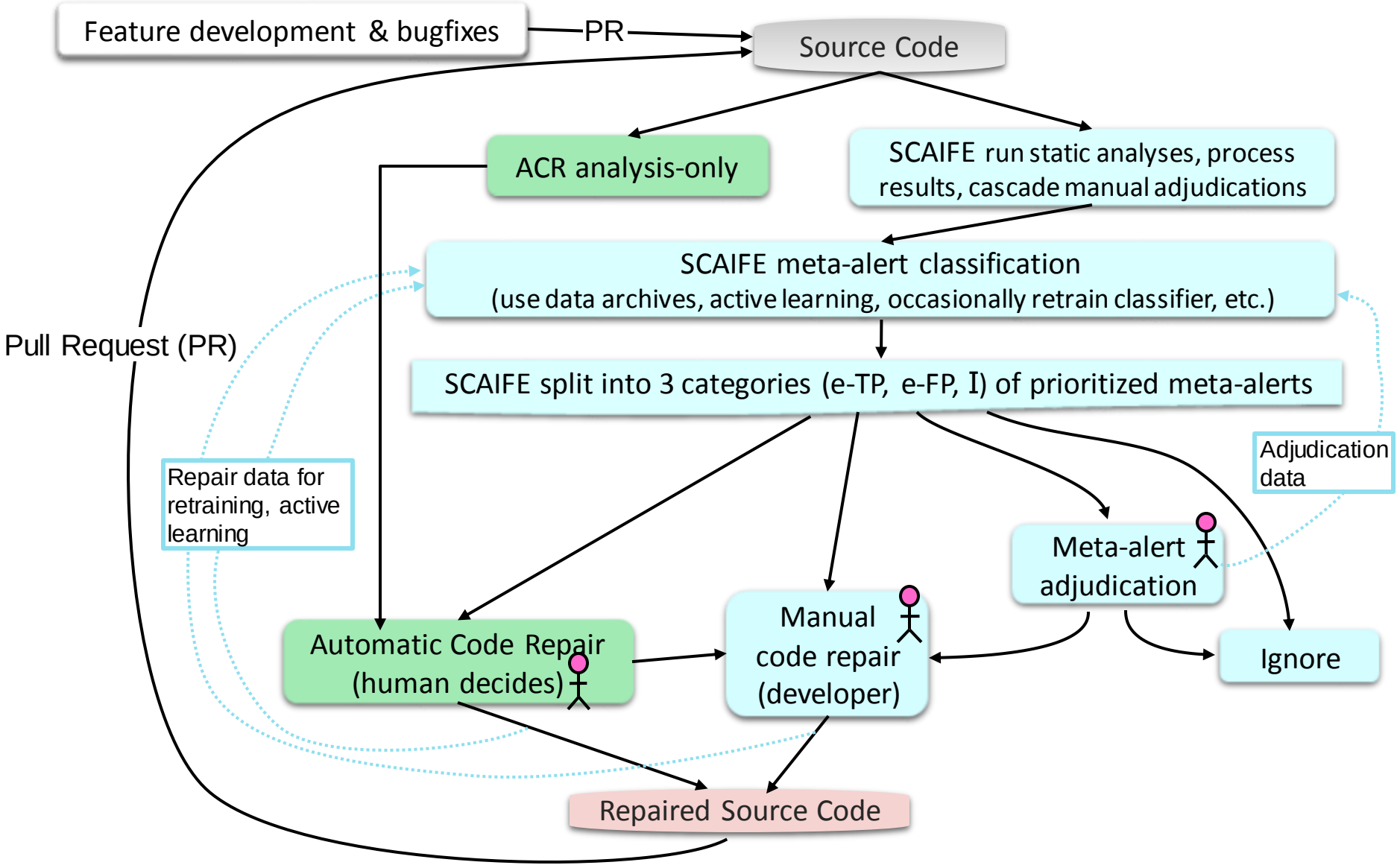


- ACR option 1: Make all possible automatic repairs (worse runtime overhead, better safety)
- ACR option 2: Only repairs higher-priority meta-alerts (less runtime overhead, but might leave unfixed vuls)
- ACR option 3: Analyst views potential repairs for high-priority meta-alerts and approves or rejects each individually.

One path down could happen:

- during one continuous integration build cycle (cycle until build passes, then repeat for new builds)
- during a code security analysis+fix

ACR & SCAIFE Meta-Alert Classification

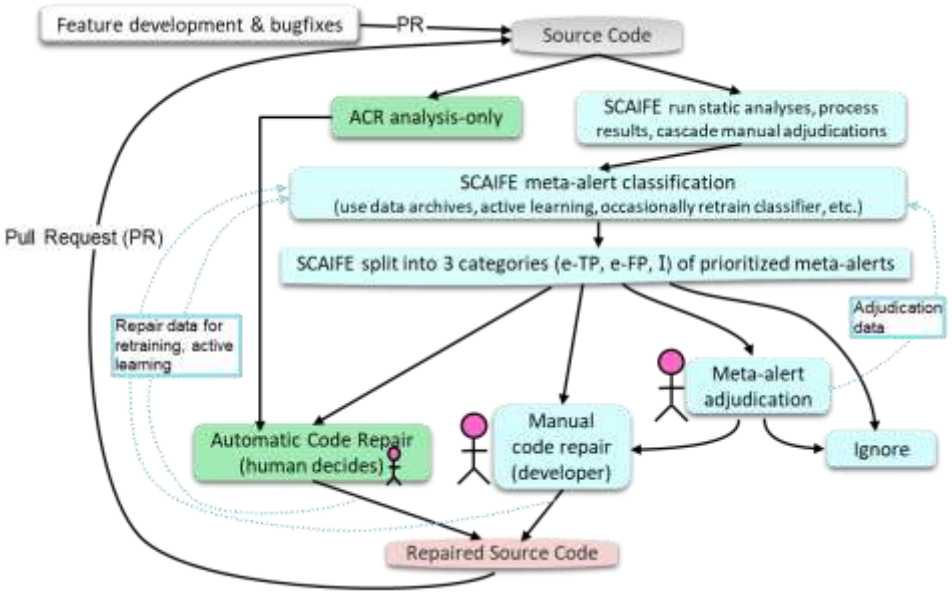


Example: SCAIFE and ACR option 3 (manual decisions)

Parallel manual effort possible:

- ACR decisions
- manual code repair
- meta-alert adjudication

ACR & SCAIFE: Latency Impacts on Design



- ACR option 1: Make all possible automatic repairs (worse runtime overhead, better safety)
- ACR option 2: Only repairs high-priority meta-alerts (less runtime overhead, but might leave unfixed vuls)
- ACR option 3: Analyst views potential ACR repairs for high-priority meta-alerts and approves or rejects each individually.

One vertical flow could happen:

a. for a CI build, with SCAIFE / ACR results that the CI system can use
b. during a code security analysis + fix
c. for one code commit, with manual repairs/analyses working down priority lists

- **For fast ACR and SCAIFE analyses**, the analyses (and potential repairs) could be potentially be done for each CI build.
- **For slower ACR and SCAIFE analyses**, (e.g., for large codebases) the envisioned design would have the analysis be done on a code version (e.g., on a particular commit to a repository). After, ACR (1,2, or 3) and SCAIFE "I" adjudications and "e-TP" manual repair continue on that code version until new analyses are available for a later commit.
- This project will **measure latencies** to determine if ACR and SCAIFE analyses should be done in parallel (as shown) or if some ACR analyses should be done after SCAIFE analyses, to reduce ACR analysis latency.

GUI Mock-up

List of meta-alerts*, ACR-fixable at top. Click to see associated code and reliability. SCAIFE GUI meta-alert list has many more fields: line, filepath, notes, etc.

* **Meta-alert**: fused SA alerts mapped to same condition from taxonomy of code flaws (e.g., CWE-190)

ACR possible?	Priority	Auto-Repair?	Manual Adjudication	Classifier Confidence True (%)	Condition
yes	8890	ACCEPT			CWE-190
yes	8889	REJECT			INT31-C
yes	8888	ACCEPT			CWE-191
yes	8887	ACCEPT			CWE-79
yes	8886	ACCEPT			CWE-119
yes	8885	PENDING DECISION		88	CWE-787
yes	8884	PENDING DECISION		86	CWE-125
yes	8883	PENDING DECISION		85	CWE-89
yes	8882	PENDING DECISION		84	CWE-200
yes	8881	PENDING DECISION		82	CWE-416
.....					
no	1000	N/A		90	CWE-352
no	999	N/A		85	CWE-787
no	998	N/A		84	CWE-22
no	997	N/A		82	CWE-476
no	996	N/A		81	CWE-287
no	995	N/A		80	CWE-434
no	994	N/A		78	CWE-732
no	993	N/A		77	CWE-94
no	992	N/A		73	CWE-522
no	991	N/A		65	EXP34-C

Bold text specifies current code view

Diff View for ACR Fixes

Source Code for Manual Adjudication

Reliable fix: True

For this ACR fix, select files on right to see edits below (red text) that will be made if fix is accepted.

File X File Z
File Y File Q

Original Source Code File X

```
#define BUF_SIZE 256
char nondet_char();

int main() {
    char* p = malloc(BUF_SIZE);
    char c;
    while ((c = nondet_char()) != 0) {
        *p = c;
        p = p + 1;
    }
    return 0;
}
```

Repaired Source Code File X

```
#include "fat_header.h"
#include "fat_stdlib.h"
#define BUF_SIZE 256
char nondet_char();

int main() {
    FatPtr_char p =
          fatmalloc_char(BUF_SIZE);
    char c;
    while ((c = nondet_char()) != 0) {
        *bound_check(p) = c;
        p = fatp_add(p, 1);
    }
    return 0;
}
```

GUI Mock-up

Changed classifier confidences and order: ACR labeled meta-alerts are used by adaptive heuristic and when retraining the classifier.

ACR possible?	Priority	Auto-Repair?	Manual Adjudication	Classifier Confidence True (%)	Condition
yes	8890	ACCEPT			CWE-190
yes	8889	REJECT			INT31-C
yes	8888	ACCEPT			CWE-191
yes	8887	ACCEPT			CWE-79
yes	8886	ACCEPT			CWE-119
yes	8885	ACCEPT			CWE-787
yes	8884	REJECT			CWE-125
yes	8883	ACCEPT			CWE-89
yes	8882	ACCEPT			CWE-200
yes	8881	ACCEPT			CWE-416
.....					
no	1000	N/A	True positive		CWE-352
no	999	N/A	True positive		CWE-787
no	998	N/A	False positive		CWE-611
no	997	N/A	True positive		CWE-476
no	996	N/A		82	EXP34-C
no	995	N/A		80	CWE-434
no	994	N/A		78	CWE-732
no	993	N/A		77	CWE-94
no	992	N/A		73	CWE-522
no	991	N/A		65	CWE-22

Bold text specifies current code view

Diff View for ACR Fixes

Source Code for Manual Adjudication

Source Code dos2unix-7.2.2/common.c

```
427   fname_len = strlen(dir) + strlen("/d2utmpXXXXXX") + sizeof (char);
428   if (!(fname_str = malloc(fname_len)))
429      goto make_failed;
430   sprintf(fname_str, "%s%s", dir, "/d2utmpXXXXXX");
431   *fname_ret = fname_str;
432
433   free(cpy);
434       while ((c = nondet_char()) != 0) {
435   #ifdef NO_MKSTEMP
436            name = mktemp(fname_str);
437             *fname_ret = name;
438             if ((fd = fopen(fname_str, W_CNTRL)) == NULL)
439                goto make_failed;
```

SCAIFE and SEI ACR: Current and Future Work (1/2)

SEI-ACR:

- Currently works on some small/medium (10 kLOC) real-world codebases.
- Still needs additional development to consistently handle real-world codebases.
  - We estimate about 8 person-weeks of effort.

SEI SCAIFE:

- Modular system, containerized with Docker.
- Combines the results from multiple SA tools.
- Manual adjudication of meta-alerts via a GUI.
- Can create labeled data for classifiers from test suites.
- GUI-based specification, training, and running of static analysis classifiers.
- Formally defined APIs for each module, using OpenAPI v3.
  - Parts can be swapped out (e.g., classifier, active learning, user interface).
- It has some continuous integration (CI) functionality.
  - It needs further development to support CI builds: Stats Module re-classification should start automatically, plus add a 'CI Orchestrator' to ensure system consistency.

SCAIFE and SEI ACR: Current and Future Work (2/2)

Integrating ACR with SCAIFE:

- GUI for previewing and accepting/rejecting auto-repairs.
- Define the API for communications between ACR and SCAIFE.
- Develop API-related code.
- Containerize SEI ACR.
- Add code hooks to record metrics to analyze for the project.

Transitioning into production use:

- Enable use in a DevSecOps pipeline.
- Expected to require:
  - security hardening,
  - performance improvements,
  - possibly a cloud-ready design.

Select Performance Metrics

Among many other metrics we plan to gather:

- Validation of our tool on codebases representative of what DoD encounters in software assurance. Metrics:
- Compare pre- and post-repair
  1. classifier precision & recall;
  2. adjudication cascading (compare counts and manually verify random selections)
- Counts of auto-repaired code (lines, SEI ACR suggestions accepted)
- Performance per goal: percent automatically-handled meta-alerts, precision, recall

SCAIFE and ACR:
Tools & Research

Invitation to Collaborate

Interest in SCAIFE-ACR Integration?

To do the integration work described, we would need more funding.

Would you or your org be interested? If so, please contact us!

Invitation to Test

We invite you to test and/or extend SCAIFE:

- Full SCAIFE system release limited to DoD and DoD contractors (Distribution D)
- Testing does *not* have to include data sharing
- If interested please contact: lflynn@cert.org

Deployment and testing support by SCAIFE:

- release system Docker-containerized, with configuration files (ports, URLs, names) to ease integration in wide variety of systems
- comes with documentation, much-extended in last year per collaborator feedback
- hands-on demos and tutorials, for quick start
- Development support includes set of CI tests

Thanks + Contact Info

Thank you for listening!

Questions?

Feedback and **potential funding/collaborations** are welcome.

Dr. Lori Flynn
lflynn@sei.cmu.edu

Carnegie Mellon University
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612