Static Analysis Classification Research FY16-20

for Software Assurance Community of Practice

September 22, 2020
Presenter: Dr. Lori Flynn (PI)

FY20 Team: Ebonie McNeil, Matt Sisk, David Svoboda, Hasan Yasar, Joseph Yankel, David Shepard, and Shane Ficorilli

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

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.

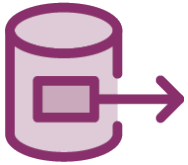
DM20-0813

Static Analysis Classification: Research FY16-20

Today, techniques & tools developed in 5 years of research projects I've led:

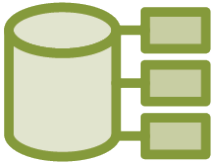
- Each project built on tools and techniques from the previous project.

<u>Highlighted FY16-20 Research Focus Areas</u>

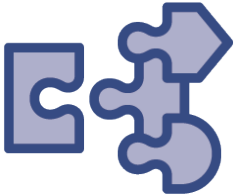


Improve classifier precision & recall

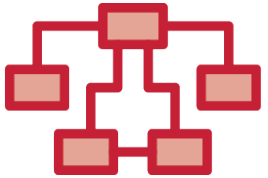
Data quality



Wide variety of labeled data



Enable classifier use via modular architecture



Enable classifier use in CI systems

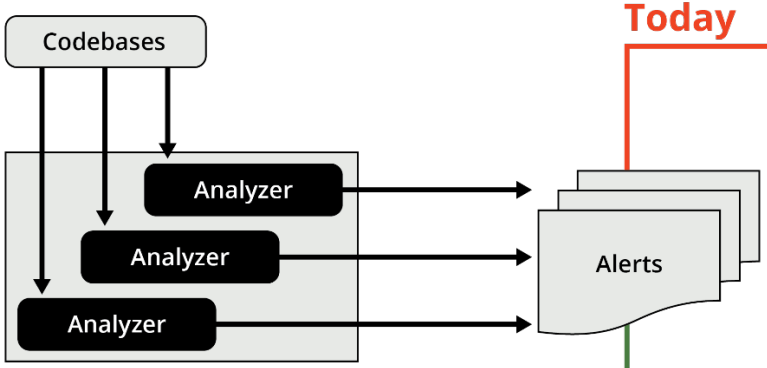
Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

Overview

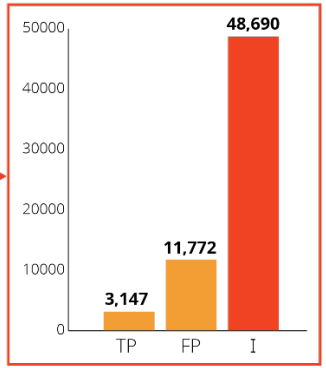
Definitions: An *alert* is an SA warning (with a checker ID, line #, filepath, message); an *alertCondition* is an alert mapped to a code flaw taxonomy item (e.g., CWE-190); and a *meta-alert* is mapped to by the set of all alertConditions that differ only by checker ID. We do classification and adjudication at the meta-alert level.

Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

Problem: too many alerts
Solution: automate handling

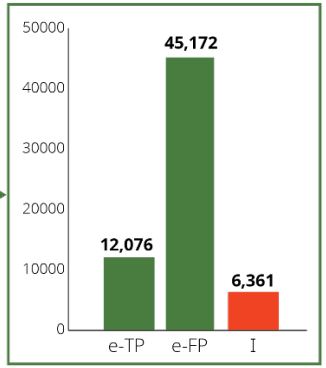


Today



Project Goal

Enable *practical* automated classification, so all meta-alerts can be addressed, accurately classifiying meta-alerts as:
Expected True Positive (e-TP)
or
Expected False Positive (e-FP)
and the rest as
Indeterminate (I)



Static analysis (SA) is an analysis of code without executing it:

- Automated SA is widely used.
- It is a normal part of testing by DoD and commercial organizations.

Static Analysis Classification Research FY16-20

Five Years in Two Slides

FY16-19 Static Analysis Meta-Alert Classification Research

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

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 $$, data, experts)
- Novel approach: **develop an extensible architecture with a novel test-suite data method**
- Result: **wider use of classifiers (less $$, 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)

FY20 Static Analysis Meta-Alert Classification Research

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

FY20 (of a two-year project, FY20-21)

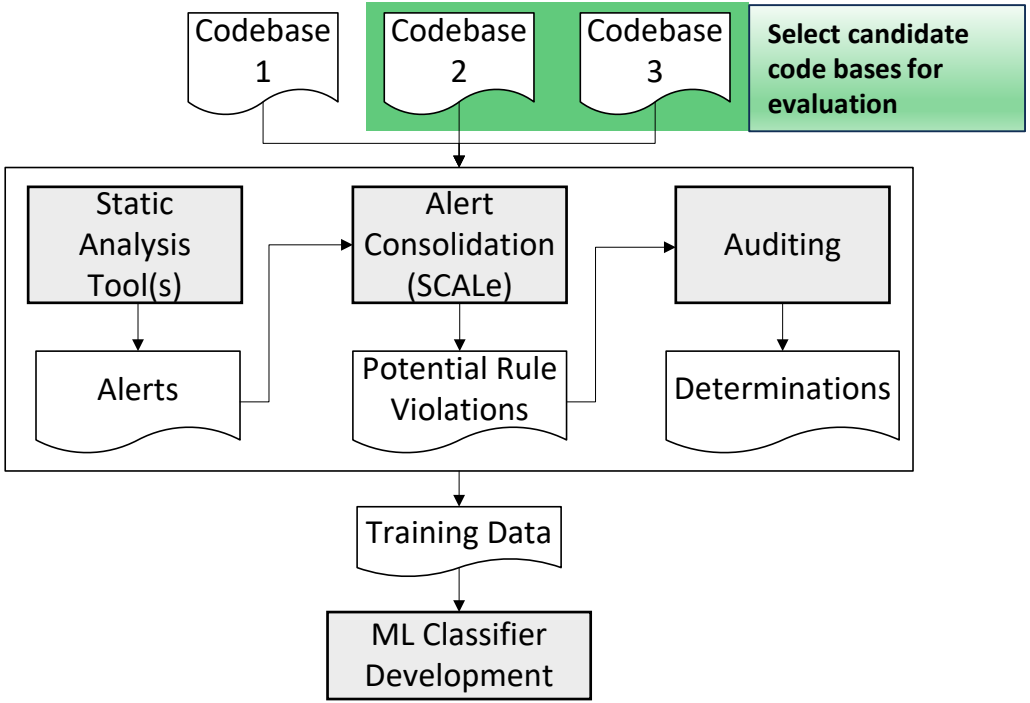
- Issue addressed: It takes too much time to adjudicate (i.e., audit) static analysis meta-alerts during continuous integration (CI).
- Novel approach: During CI builds, use **classifiers** with **precise cascading** and **CI/CD features**.
- Results
  - Design for CI-SCAIFE system integration
  - SCAIFE System v 1 release (classifier defined, run, and results can be viewed from [G]UI module)
  - Defined cascading API
  - Less-precise cascading using the API
  - Test results for less-precise cascading
  - Significant progress on CI-SCAIFE system integration development
  - Deployment and testing by DoD collaborators (multiple rounds)
  - A published RC_Data open dataset for improved classifier research
  - APIs, technical manuals, and SCALe public publication

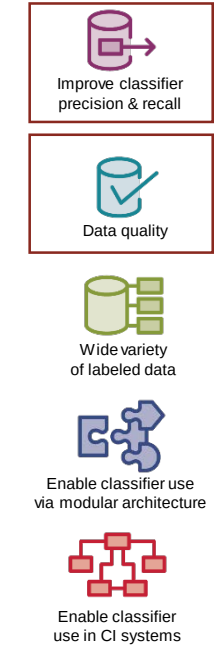
FY21 plan: a precise cascading algorithm, improved classifiers, full integration

Static Analysis Classification Research FY16-20

Data Quality: Audit Lexicon and Rules

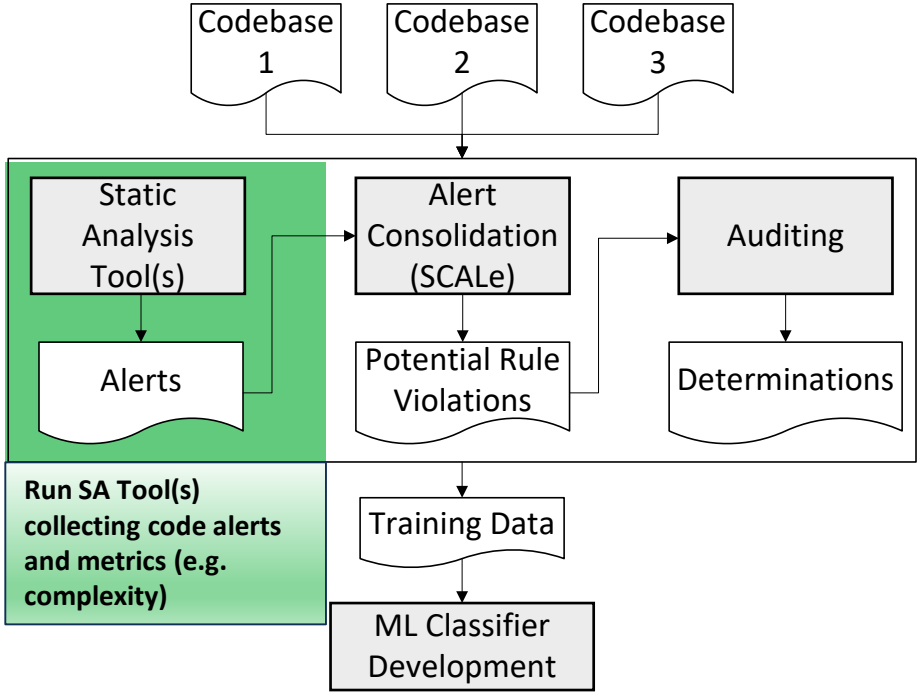
Meta-Alert Classifiers and Data Quality

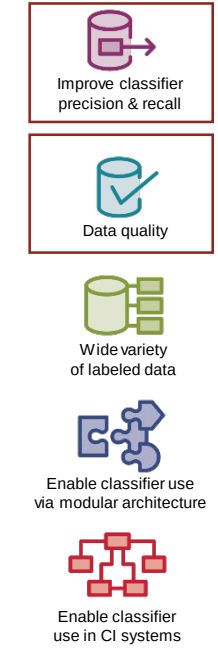




Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

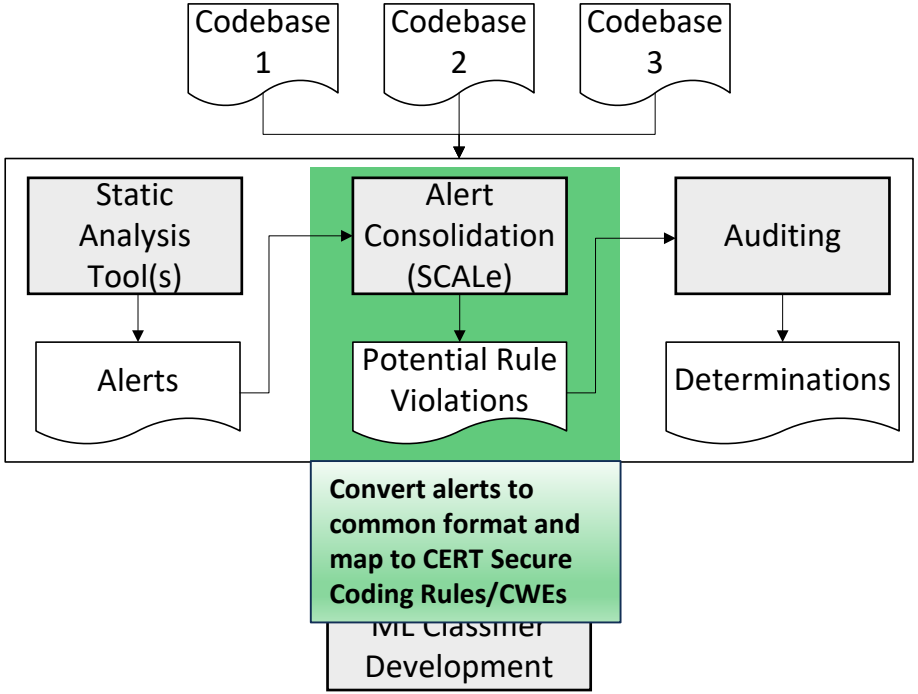
Meta-Alert Classifiers and Data Quality

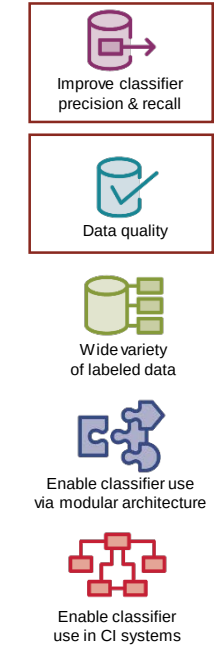




Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

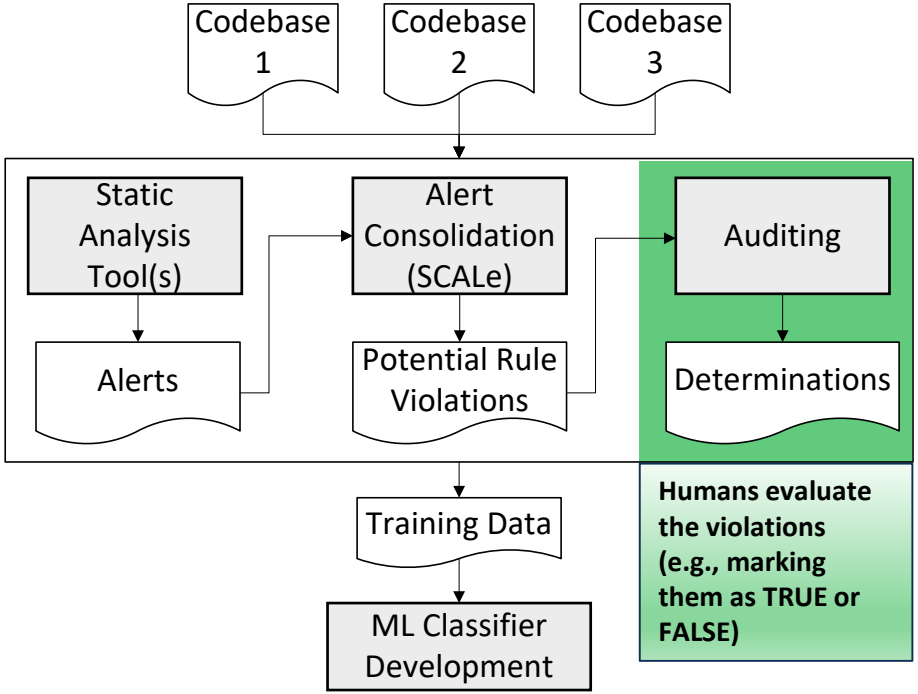
Meta-Alert Classifiers and Data Quality

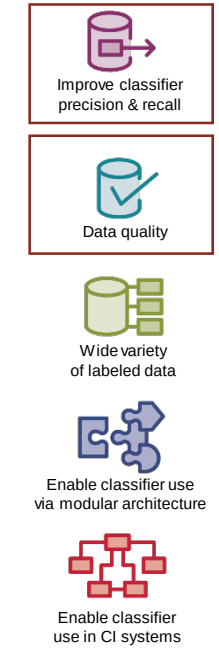




Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

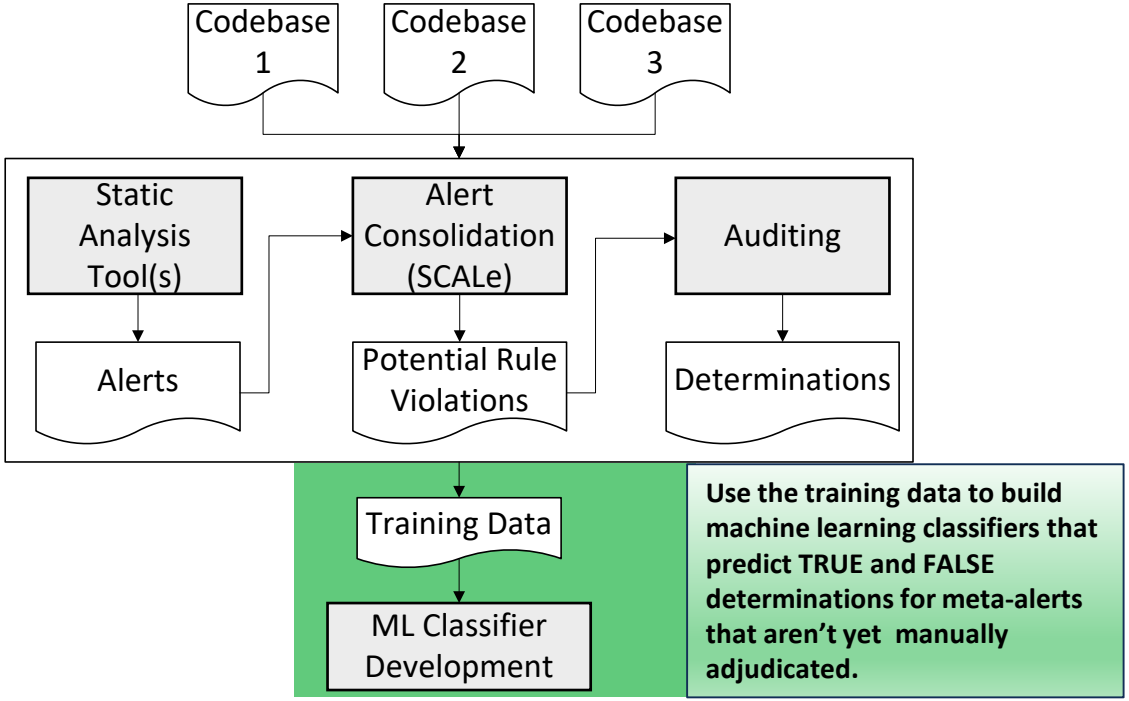
Meta-Alert Classifiers and Data Quality

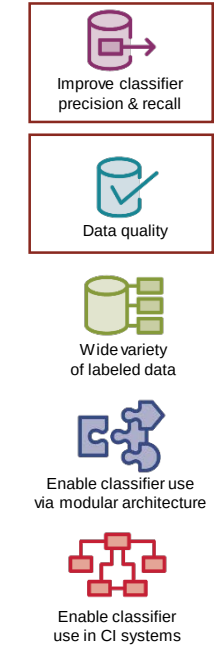




Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

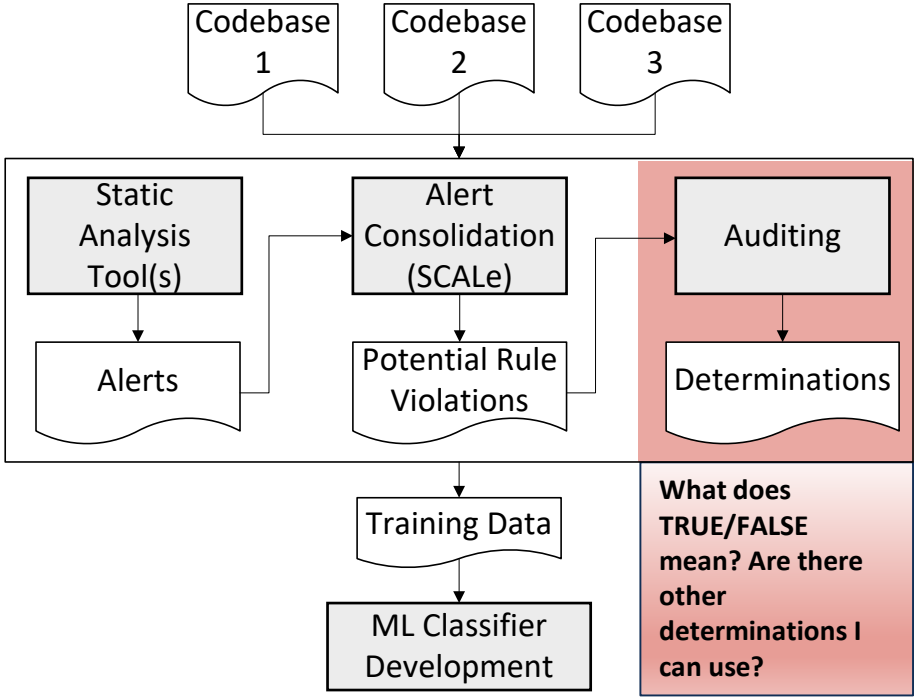
Meta-Alert Classifiers and Data Quality

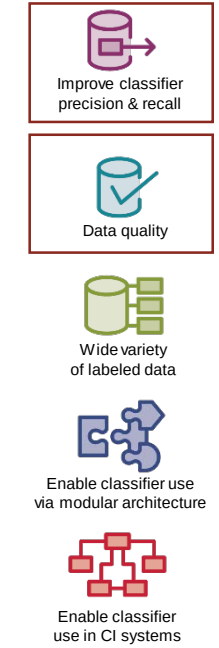




Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

Meta-Alert Classifiers and Data Quality



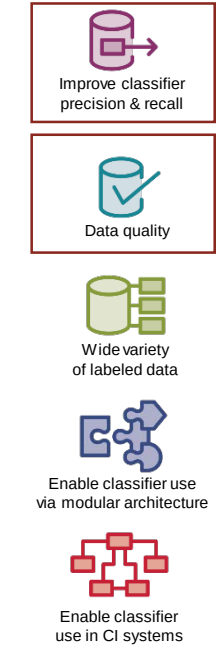


Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

Data Quality: What Is Truth?

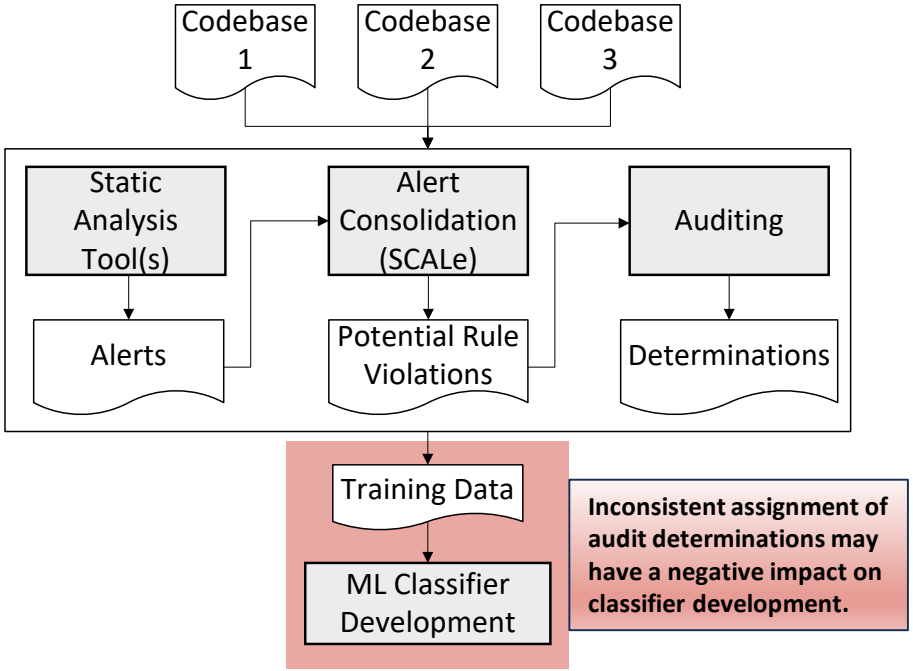
One collaborator reported using the determination True to indicate that the issue reported by the meta-alert was a real problem in the code.

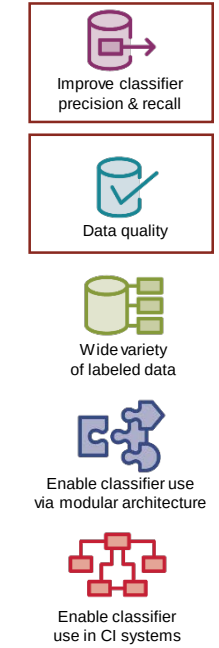
Another collaborator used the determination True to indicate that something was wrong with the diagnosed code, even if the specific issue reported by the meta-alert was a false positive.



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

Meta-Alert Classifiers and Data Quality





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

Data Quality: Lexicon and Rules

- We developed a **lexicon** and auditing **rule set** for our collaborators.
- It includes a standard set of well-defined **determinations** for static analysis meta-alerts.
- It also includes a set of **auditing rules** to help auditors make consistent decisions in commonly encountered situations.

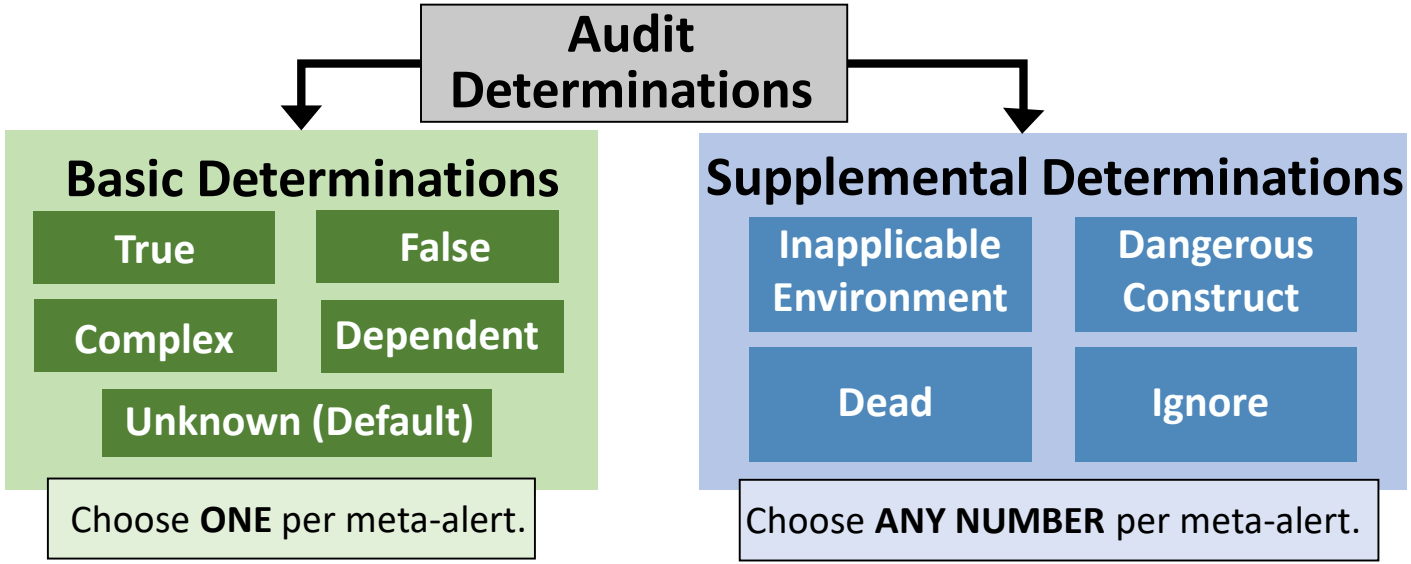
Different auditors should make the **same determination** for a given meta-alert.

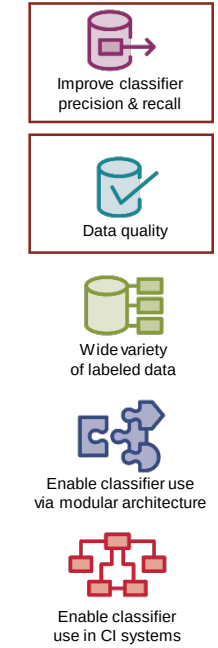
Improve the **quality and consistency** of audit data for the purpose of building **machine learning classifiers**.

Help organizations make **better-informed** decisions about **bug fixes**, **development**, and **future audits**.

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

Lexicon: Audit Determinations





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

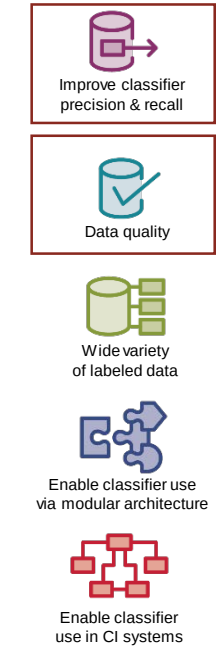
Data Quality: Audit Rules

Goals

- Clarify **ambiguous or complex** auditing scenarios.
- Establish **assumptions** auditors can make.
- Overall, help make audit determinations **more consistent**.

We developed **12 rules**:

- We drew on our experiences auditing code bases in the SEI CERT Division.
- We trained three groups of engineers on the rules, and incorporated their feedback.



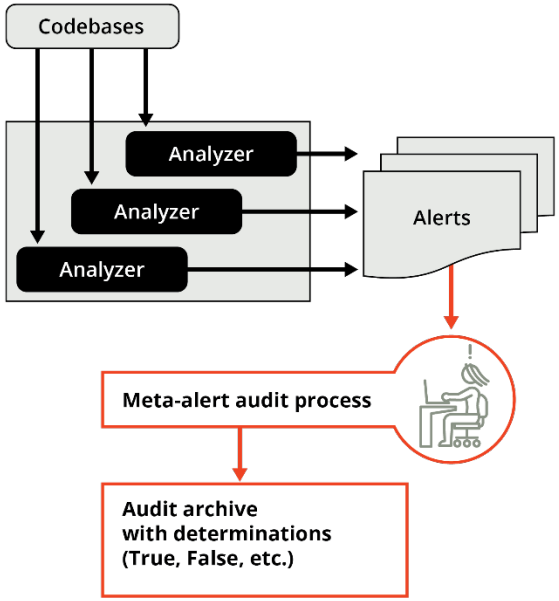
Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

Static Analysis Classification Research FY16-20

Classifier Development, Data, & Enabling Architectures

Research Tooling Too

SEI SCALe Framework: Background



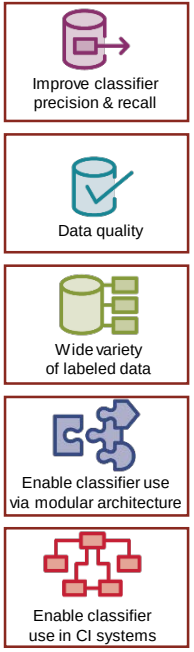
Static Analysis Meta-Alert Auditing Framework

Developed by the SEI for ~10 years.

- GUI front end to examine meta-alerts and associated code
- Meta-alert adjudications (true, false) stored in database

Use for Research Projects

- We enhance the framework with features for research.
- Collaborators use it on their codebases.
- Researchers analyze audit data.

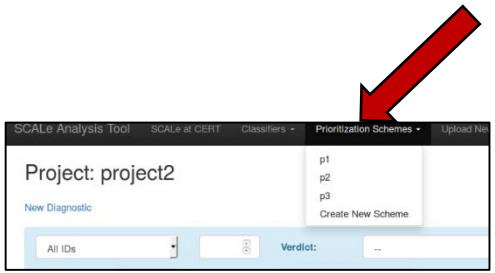


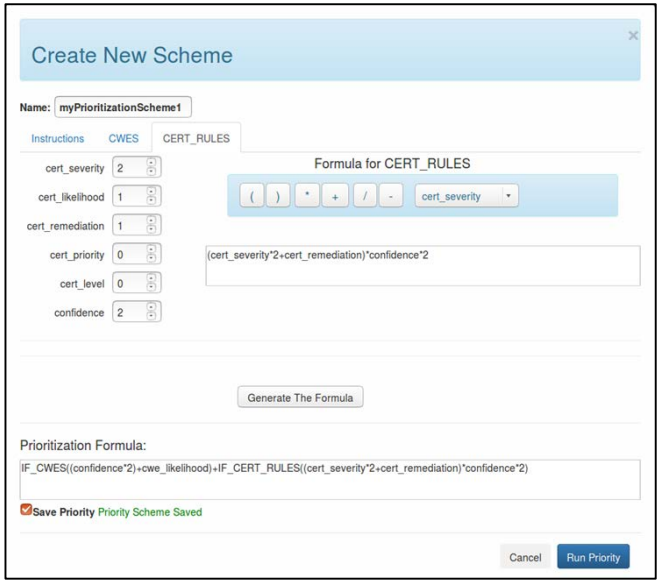
After running SA tools, meta-alert adjudication can happen at any point in the software development lifecycle.

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

Prioritization Schemes

Prioritization schemes with mathematical formulas user can create and/or use





Practical use of classification

Improve classifier precision & recall

Data quality

Wide variety of labeled data

Enable classifier use via modular architecture

Enable classifier use in CI systems

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

User Field Uploads

User field uploads

- These uploads are for advanced users who can work with SQLite databases and generate values.
- Uploaded fields can be used in priority schemes.
- The CSV uploaded file has the following:
  - One line per project meta-alert ID
  - A left-most field with a meta-alert ID
  - A top row that holds field labels

```
meta_alert_id,safeguard_countermeasure,
vulnerability,residual_risk,impact,
threat,risk,complexity,severity,coupling
112,5,1,4,9,1,1,5,5,1
2,9,3,3,3,1,1,1,9,3
3,3,1,1,1,8,1,5,5,1
4,6,1,1,5,2,1,8,8,1
5,2,1,1,2,3,1,7,7,5
6,5,1,4,4,1,2,4,5,1
7,8,5,3,4,8,2,4,9,9
8,2,1,3,2,8,3,8,8,1
9,6,4,3,6,9,1,4,4,4
10,3,2,2,5,7,1,4,5,9
11,6,1,1,9,6,1,7,7,1
12,2,8,4,1,6,1,4,4,8
```

Practical use of classification

Improve classifier precision & recall

Data quality

Wide variety of labeled data

Enable classifier use via modular architecture

Enable classifier use in CI systems

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

Archive Sanitizer for Collaborator Data Sharing

We added a data sanitizer to SCALe that has the following functions:

- Anonymizes sensitive fields
- Has an SHA-256 hash with salt
- Enables analysis of features correlated with meta-alert confidence

The audit archive for the project is in a database:

- DB fields may contain sensitive information.
- The sanitizing script anonymizes or discards fields:
  - Diagnostic message
  - Path, including directories and filename
  - Function name
  - Class name
  - Namespace/package
  - Project filename

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

Practical use of classification

Improve classifier precision & recall

Data quality

Wide variety of labeled data

Enable classifier use via modular architecture

Enable classifier use in CI systems

Data Used for Classifiers

Data used to create and validate classifiers:

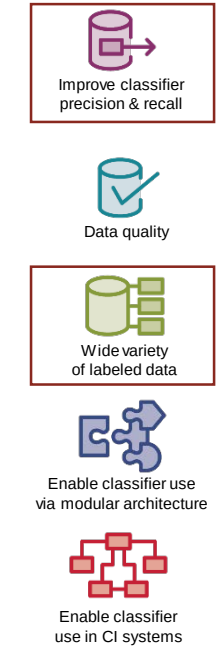
- CERT-audited meta-alerts:
  - ~7,500 audited meta-alerts
- Three collaborators that audit their own codebases with our auditing research prototype tool called “enhanced SCALe”

We pooled data (CERT and collaborators) and segmented it:

- Segment 1 (70% of data): train model
- Segment 2 (30% of data): testing

We added classifier variations on a dataset:

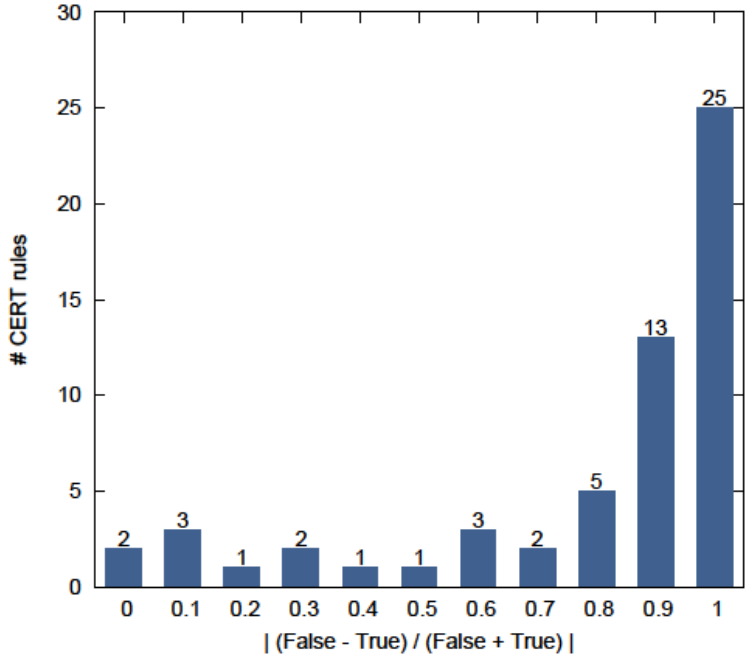
- Per-rule
- Per-language
- With/without tools
- Others



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

CERT-Audited Archives Characterization

- We had labeled data for 158 of 382 CERT rules.
- There were 58 CERT coding rules with 20 or more audited (labeled) meta-alerts.
- The other 324 CERT rules have little or no labeled data.
- For 25 rules, all (or close) were determined one way (True or False).
- 2,487 True and 4,980 False



Improve classifier precision & recall

Data quality

Wide variety of labeled data

Enable classifier use via modular architecture

Enable classifier use in CI systems

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

Analysis of Juliet Test Suite: Initial 2018 Results

Automated Adjudication	Labeled Meta-Alert (counts a fused alertCondition once)
TRUE	**13,330**
FALSE	**24,523**

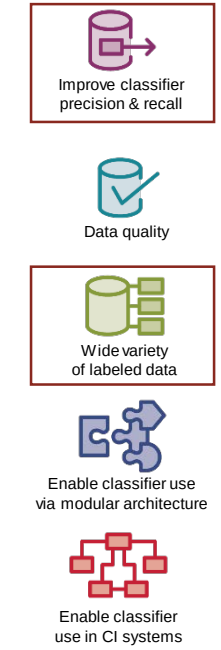
Lots of new data for creating classifiers

(37,853 labeled meta-alerts)

Big savings: a manual audit of 37,853 meta-alerts from non-test-suite programs would take an unrealistic minimum of 1,230 hours (117 seconds per meta-alert audit*).

- The first 37,853 meta-alert audits wouldn't cover many conditions (and sub-conditions) covered by the Juliet test suite.
- We needed true and false labels for classifiers.
- **Realistically**, an enormous amount of manual auditing time is required to develop that much data.

These are initial metrics; we will collect more data as we use more tools and test suites.



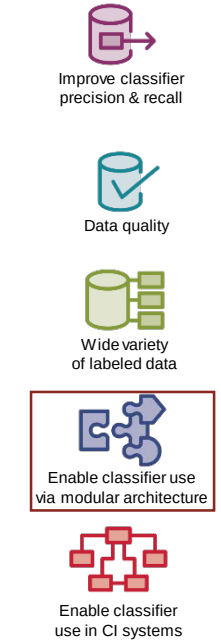
*N. Ayewah and W. Pugh. "The Google FindBugs Fixit", *International Symposium on Software Testing and Analysis,* ACM, 2010.

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

SCAIFE Definitions

SCAIFE is **a modular architecture that enables static analysis meta-alert classification** plus advanced prioritization.

- The **SCAIFE API** defines interfaces between the modular parts.
- **SCAIFE systems** are software systems that instantiate the API.
- Our SCAIFE system releases include a SCALe module plus much more.



SCAIFE = Source Code Analysis Integrated Framework Environment

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

SCAIFE Architecture Approach

For efficient development of a robust API to enable widespread classifier use, we need a system architecture that:

- Integrates with existing static analysis tools and aggregators (including SCALe)
- Supports classification and adaptive heuristic functionality
- Demonstrates fast response times for average and worst-case scenarios
- Provides extensibility for future research in static analysis, classification, architecture, and SecDevOps


Improve classifier precision & recall


Data quality


Wide variety of labeled data


Enable classifier use via modular architecture


Enable classifier use in CI systems

Swagger/OpenAPI Open-Source Development Toolset

- Quickly develops APIs following the OpenAPI standard
- Auto-generates code for servers and clients in many languages
- Tests server and client controllers with Swagger UI
- Is widely used (10,000 downloads/day)

- Big O analysis was useful.
- Design decisions required balancing goals and analyzing tradeoffs.

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

SCAIFE Architecture

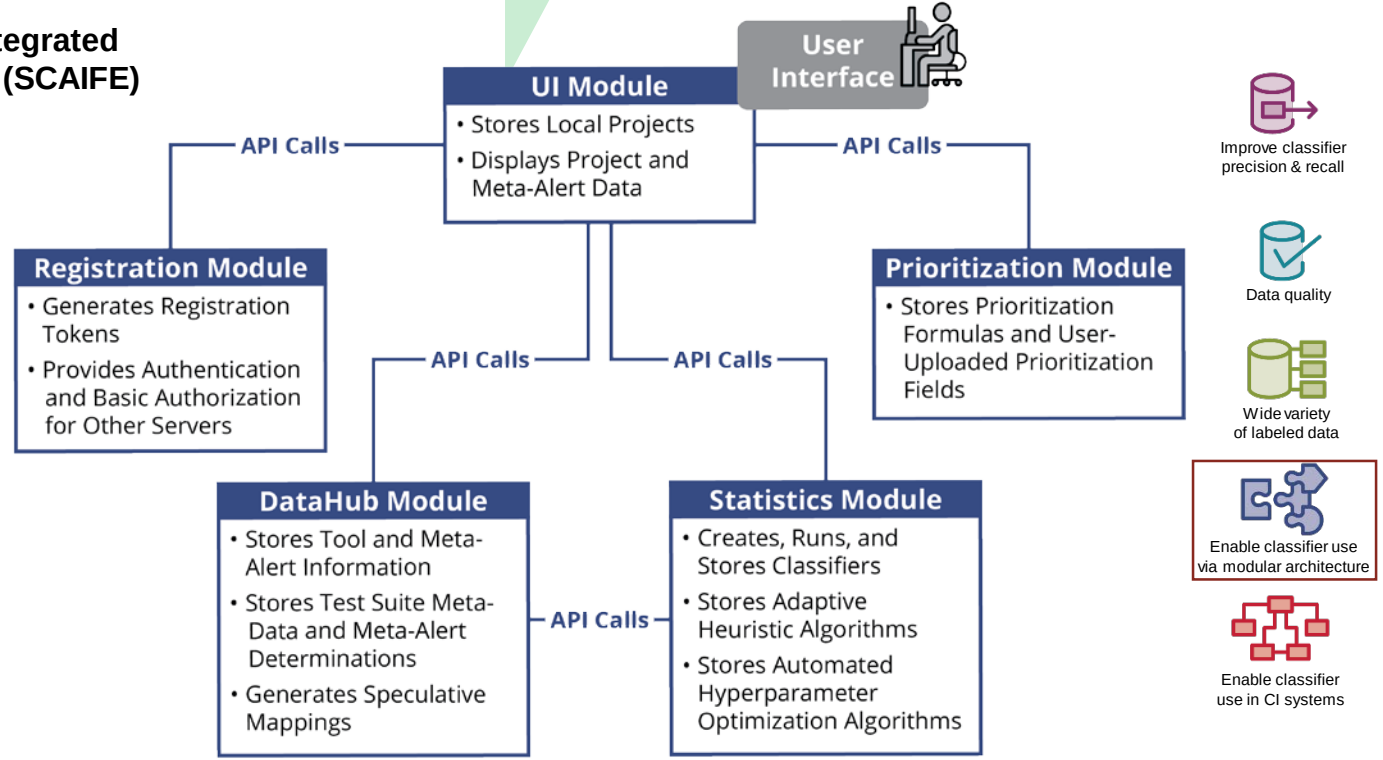
Any static analysis tool can instantiate APIs to become a UI Module. For example

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

Source Code Analysis Integrated Framework Environment (SCAIFE)

SCAIFE is a modular architecture that **enables users to efficiently start to use classifiers with a wide variety of systems and tools:**

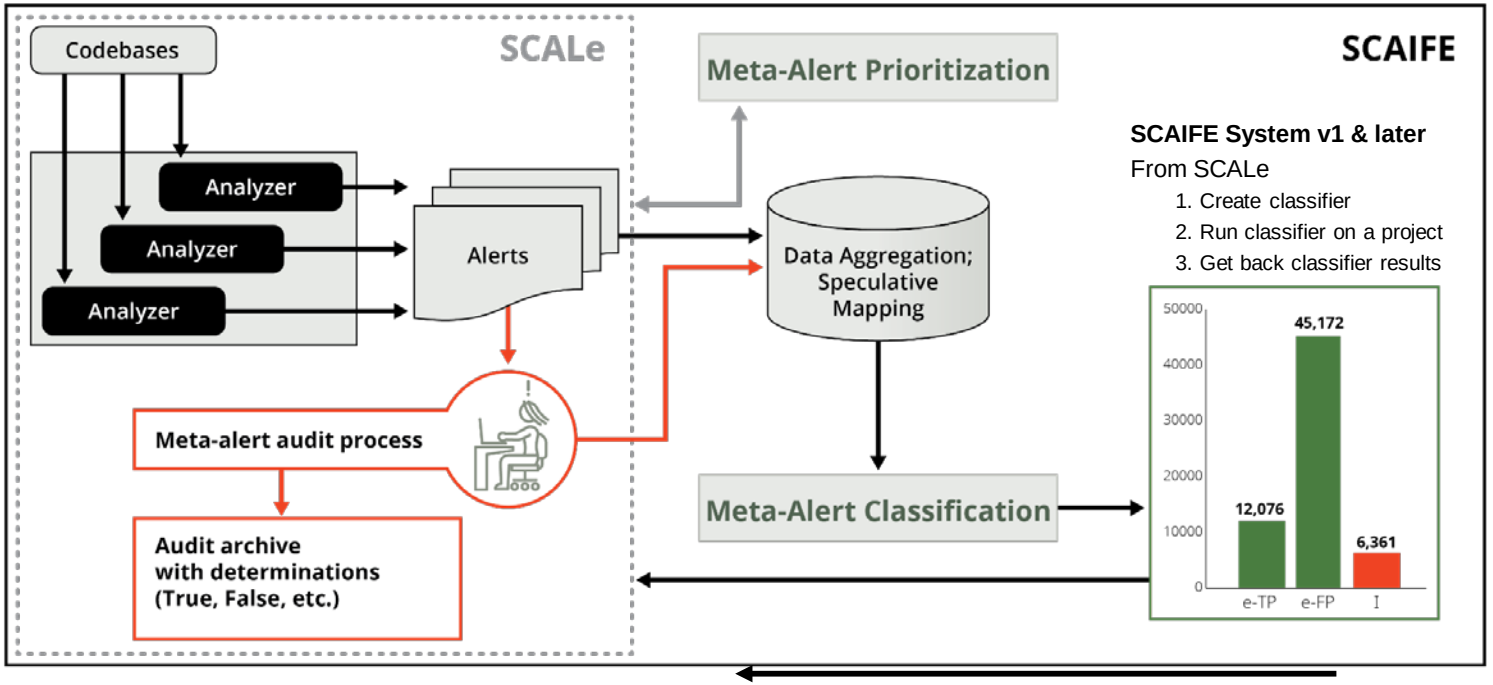
- The formal SCAIFE API definition enables automated code generation to quickly instantiate API calls and generate server stubs in many code languages. This reduces the effort required to integrate existing systems and tools.
- The UI Module instantiation of SCALe is publicly available (GitHub scaife-scale branch).
- Collaborators can get a full SCAIFE instantiation and use it as-is or substitute any module(s) and use the others.



L. Flynn, E. McNeil, and J. Yankel. "How to Instantiate SCAIFE API Calls: Using SEI SCAIFE Code, the SCAIFE API, Swagger-Editor, and Developing Your Tool with Auto-Generated Code." SEI Technical Manual. July 2020.

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

SCAIFE Meta-Alert Dataflow with SCALe Module



Improve classifier precision & recall

Data quality

Wide variety of labeled data

Enable classifier use via modular architecture

Enable classifier use in CI systems

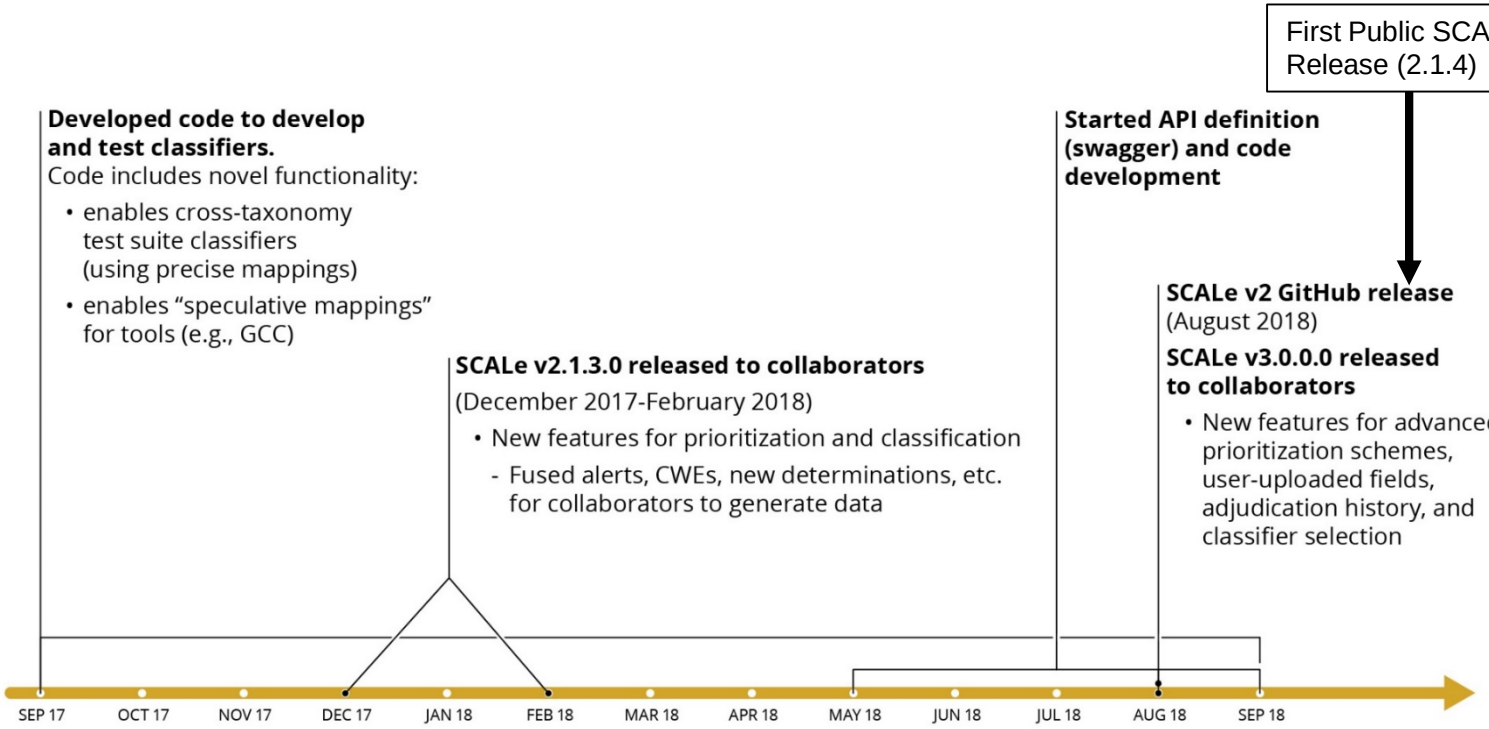
Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

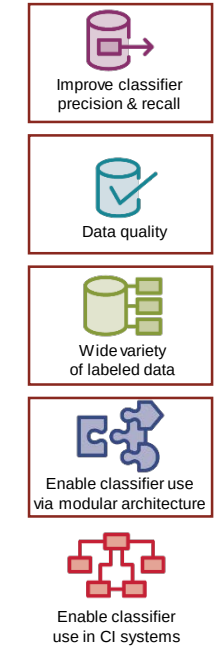
Static Analysis Classification Research FY16-20

FY18-19 Artifacts

FY18 Software Artifacts

- More recent versions available
- Notable: Multiple releases for collaborator feedback throughout



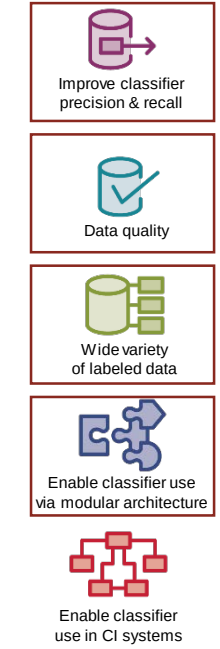


Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

FY18: Non-Code Publications

These resources remain useful

Publication Goal	Publications and Papers
Help developers and analysts provide feedback on our API, and use new SCALe features.	• SEI special report: *Integration of Automated Static Analysis Alert Classification and Prioritization with Auditing Tools* (August 2018) • SEI blog post: *SCALe: A Tool for Managing Output from Static Code Analyzers* (September 2018)
Explain classifier development research methods and results.	• Paper: *Prioritizing Alerts from Multiple Static Analysis Tools, Using Classification Models*, SQUADE (ICSE workshop) • SEI blog post: *Test Suites as a Source of Training Data for Static Analysis Alert Classifiers* (April 2018) • SEI podcast (video): *Static Analysis Alert Classification with Test Suites* (September 2018)
Enable developers and analysts to better understand tool coverage for code flaws using our inter-taxonomy precise mapping method.	• CERT manifest for Juliet (created to test CWEs) for testing CERT rule coverage with tens of thousands of tests (previously under 100) • Per-rule precise CWE mapping in two new CERT C Standard sections [1] [2]



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

Juliet Test Suite Classifiers: Initial Results (Hold-Out Data)

Classifier	Accuracy	Precision	Recall
rf	0.938	0.893	0.875
lightgbm	0.942	0.902	0.882
xgboost	0.932	0.941	0.798
lasso	0.925	0.886	0.831

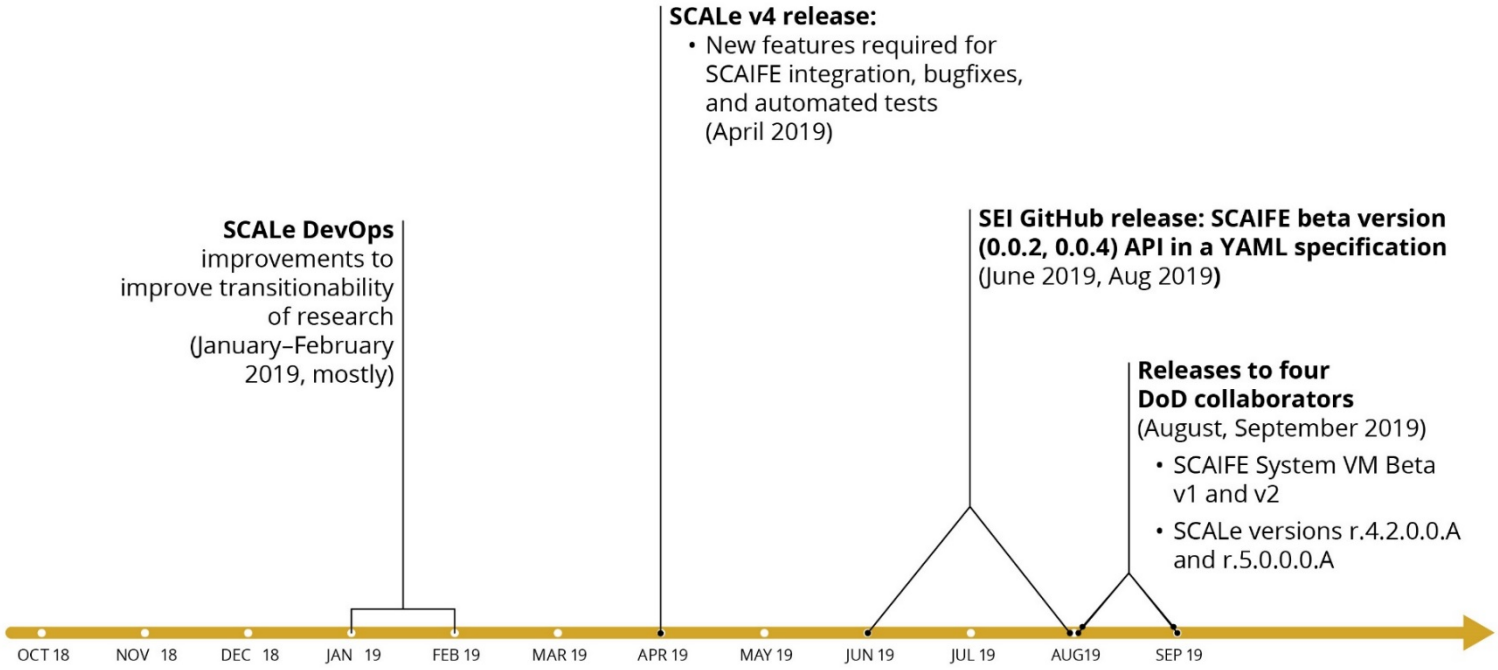
	Total Population	Actual Condition: Condition true	Actual Condition: Condition false	
Predicted Condition	Predicted condition true	True positive	False positive	Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition true}}$
	Predicted condition false	False negative	True negative	
		$\frac{\Sigma \text{ True positive}}{\Sigma \text{ (Condition true)}}$	False positive rate = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ (Condition false)}}$	

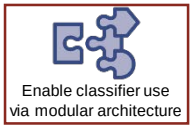
Accuracy = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$

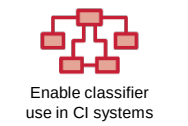
Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

FY19 Releases: Software and YAML API Definitions

- More recent versions available
- Notable: Multiple releases for collaborator feedback throughout







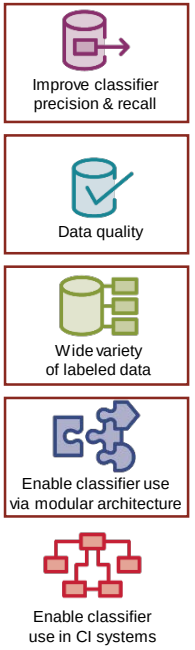
Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

FY19: Select Non-Code Publications –1

These resources remain useful

Publications to Explain Research and Development Methods and Results

- SEI blog post: *An Application Programming Interface for Classifying and Prioritizing Static Analysis Alerts* by Lori Flynn and Ebonie McNeil (July 2019)
- SEI whitepaper: *SCAIFE API Definition Beta Version 0.0.2 for Developers* by Lori Flynn and Ebonie McNeil (June 2019)
- SEI technical report: *Integration of Automated Static Analysis Alert Classification and Prioritization with Auditing Tools: Special Focus on SCALe* by Lori Flynn, Ebonie McNeil, David Svoboda, Derek Leung, Zach Kurtz, and Jiyeon Lee (May 2019)
- SEI blog post: *SCALe v3: Automated Classification and Advanced Prioritization of Static Analysis Alerts* by Lori Flynn and Ebonie McNeil (December 2018)
- Presentation: *Automating Static Analysis Alert Handling with Machine Learning: 2016-2018* (one-hour presentation at Raytheon's CyberSecurity Technical Interchange Meeting) by Lori Flynn (October 2018)



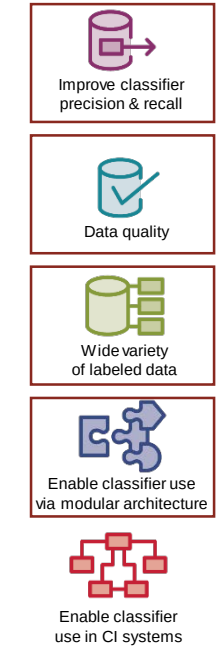
Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

FY19: Select Non-Code Publications –2

These resources remain useful

Publications to Demonstrate New Features of SCALe and SCAIFE

- Manual: *How to Review & Test the Beta SCAIFE VM* by L. Flynn and E. McNeil (v1 August 2019, v2 September 2019)
- SEI Cyber Minute by Ebonie McNeil (August 2019)
- SEI webinar: *How can I use new features in CERT's SCALe tool to improve how my team audits static analysis alerts?* (video and slides) by Lori Flynn (November 2018)
- SwACon paper: *Introduction to Source Code Analysis Laboratory (SCALe)* (one-hour presentation, including demo at Software Assurance Conference [SwACon]) by Lori Flynn (November 2018)

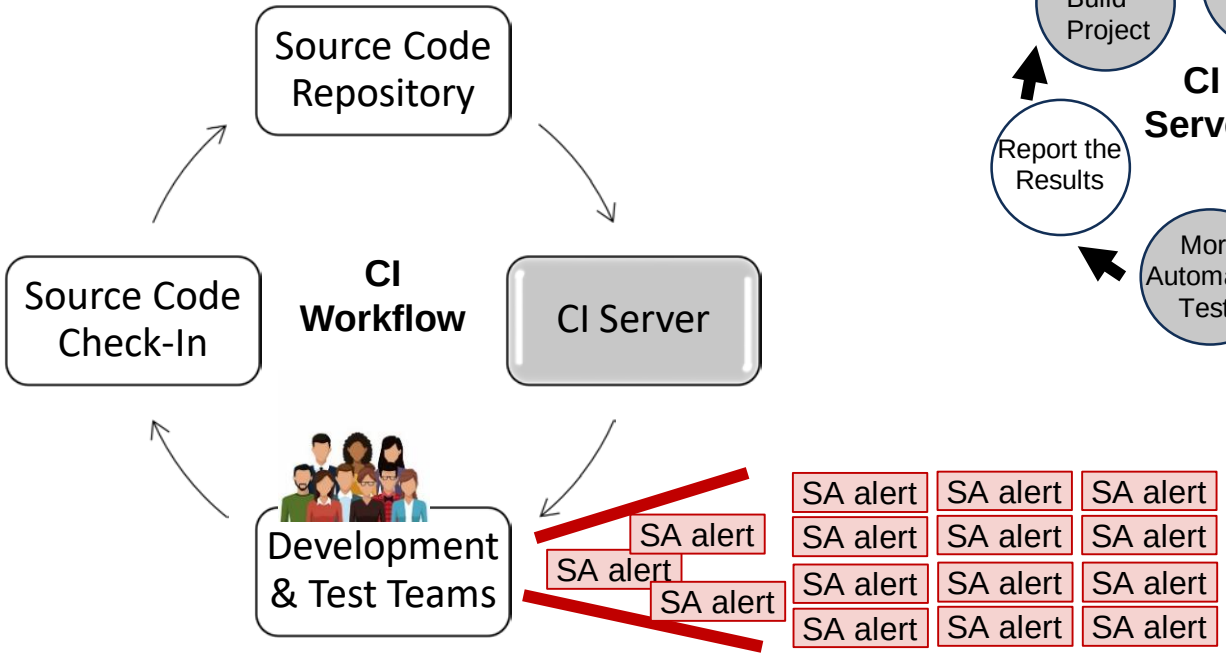


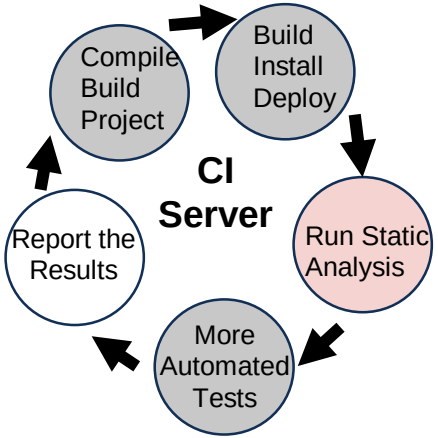
Goal: Enable **practical** automated classification, so all meta-alerts can be addressed

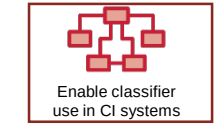
Static Analysis Classification Research FY16-20

FY20 Research Topic Detail + Artifacts

Rapid Adjudication of Static Analysis Alerts During CI

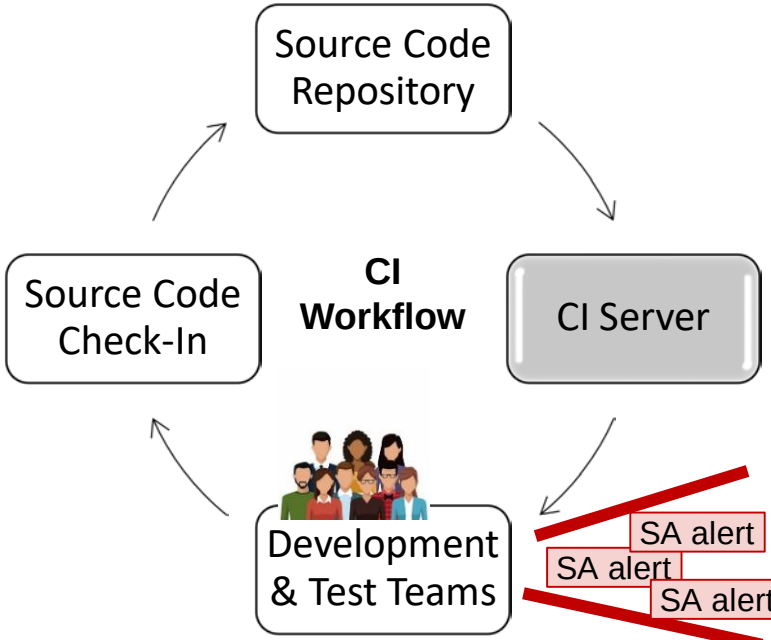


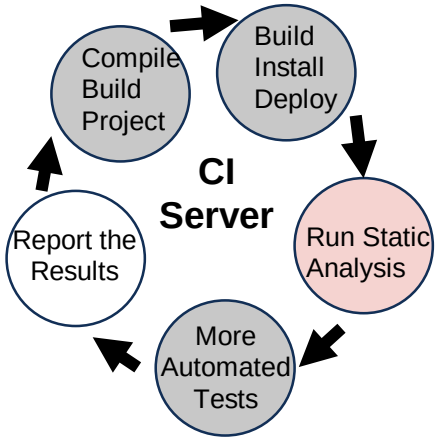


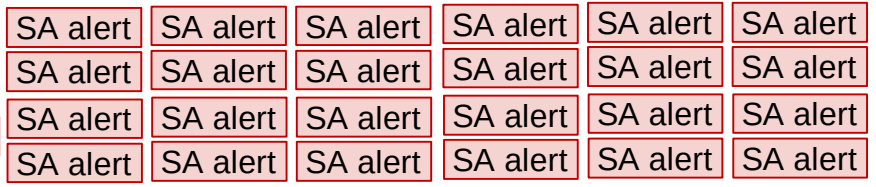


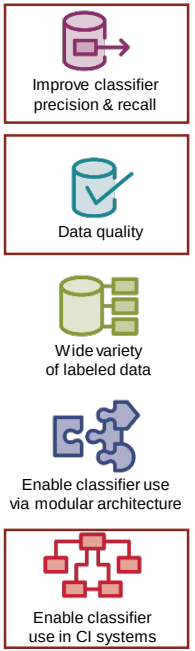
Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

Rapid Adjudication of Static Analysis Alerts During CI



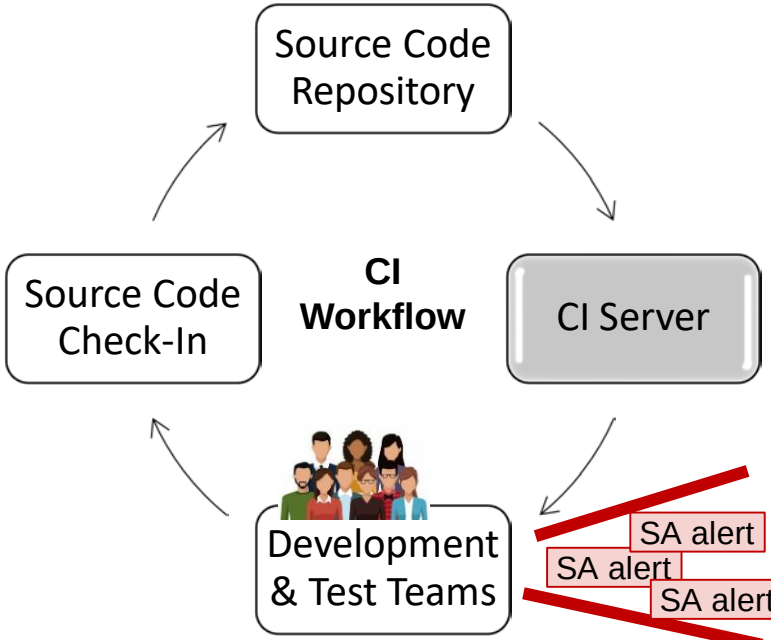


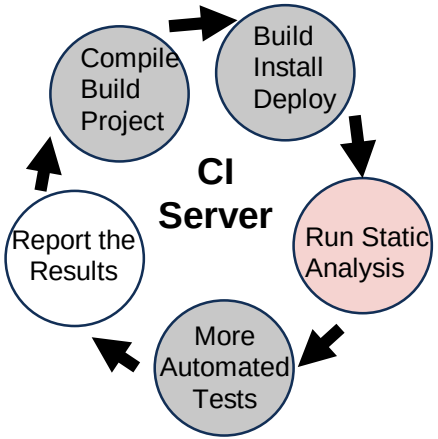


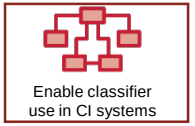


Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

Rapid Adjudication of Static Analysis Alerts During CI

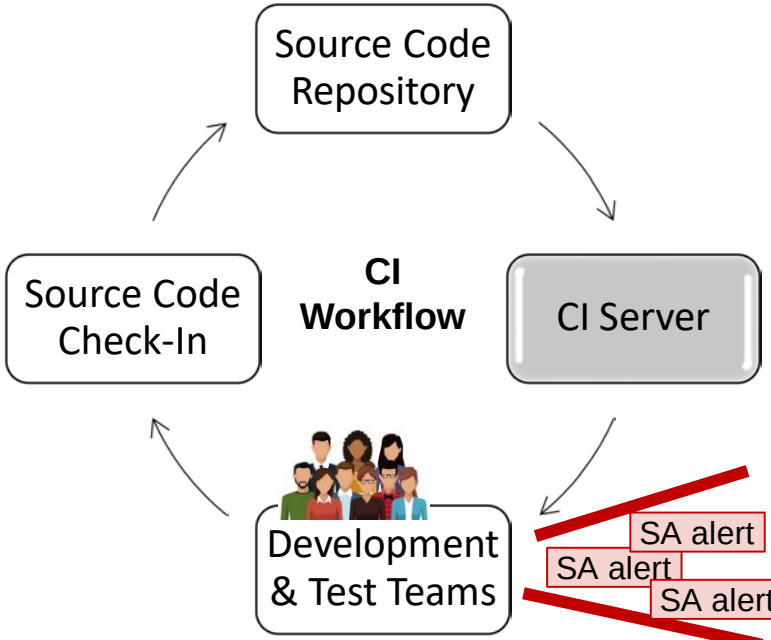


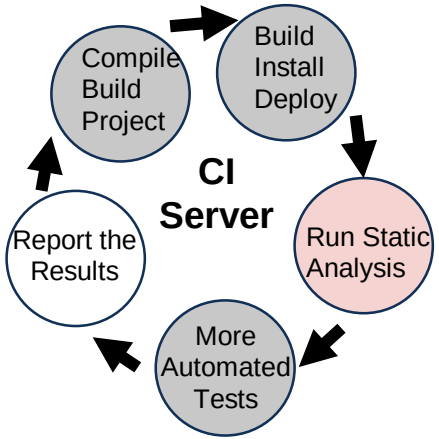


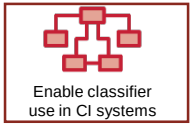


Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

Rapid Adjudication of Static Analysis Alerts During CI

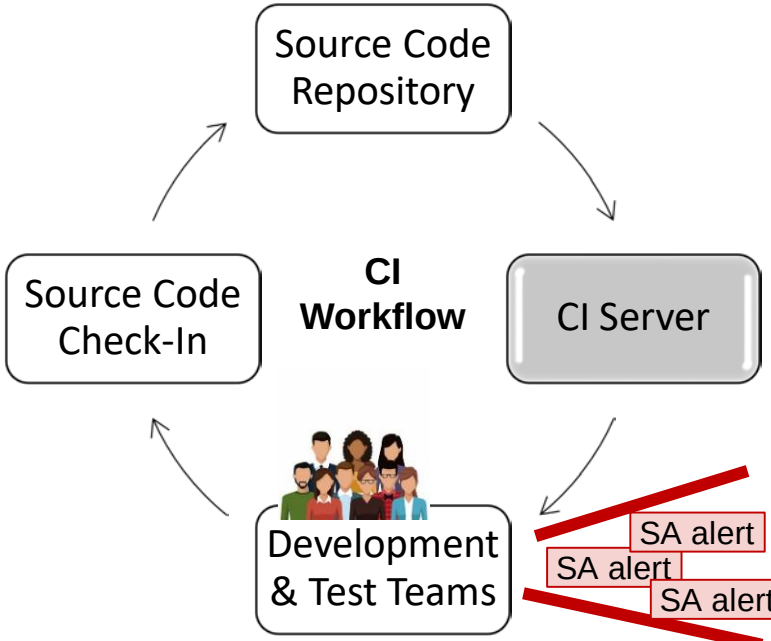


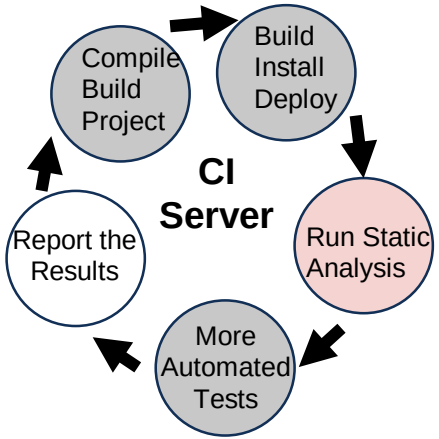




Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

Rapid Adjudication of Static Analysis Alerts During CI





Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

Rapid Adjudication of Static Analysis Alerts During CI

DoD is moving to CI/CD but doesn't have a solution to this problem

Problem: It takes too much time to adjudicate alerts from static analysis tools during continuous integration (CI).

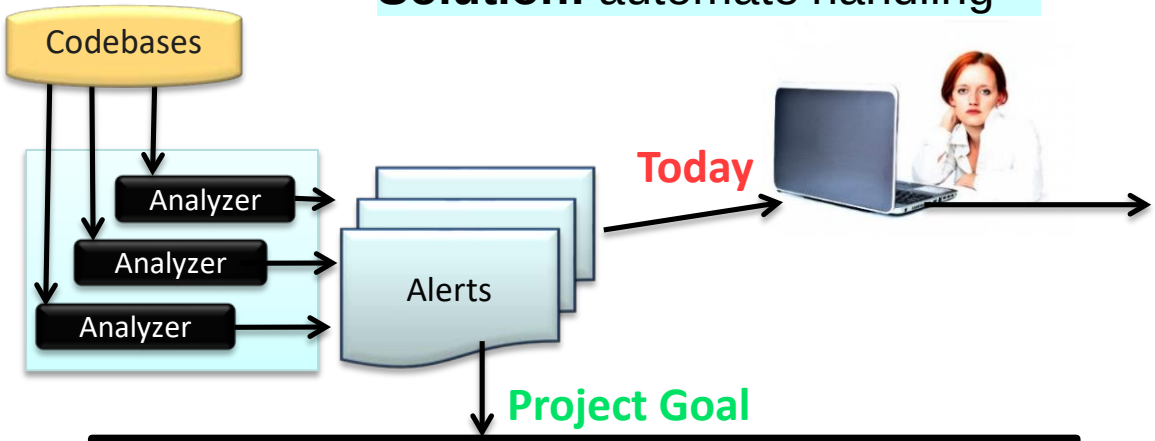
Static analysis (SA) is incompletely integrated in CI development projects in the DoD, and the selection of SA tools is limited to those with very few false positives.

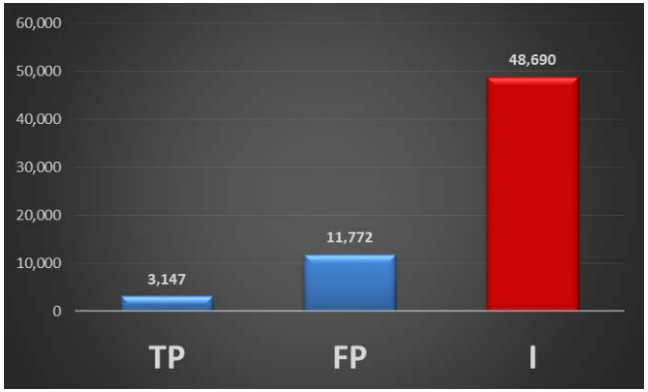
Current practice is too labor-intensive. We will automate it.

Classifiers

Problem: too many alerts
Solution: automate handling

Alert: an SA warning (with a tool checker ID, line #, filepath, message)
AlertCondition: an alert mapped to a code flaw taxonomy item (e.g., CWE-190)
Meta-alert: mapped to by the set of alertConditions that differ only by checker ID.
We do adjudication and classification at the meta-alert level





Project Goal

Classification algorithm development for CI systems, that precisely and with high recall, classifies 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)

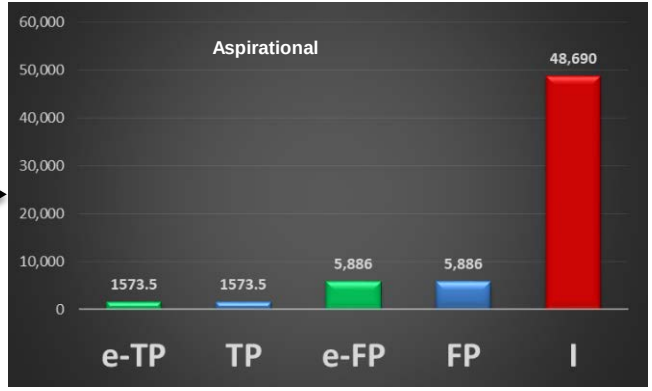
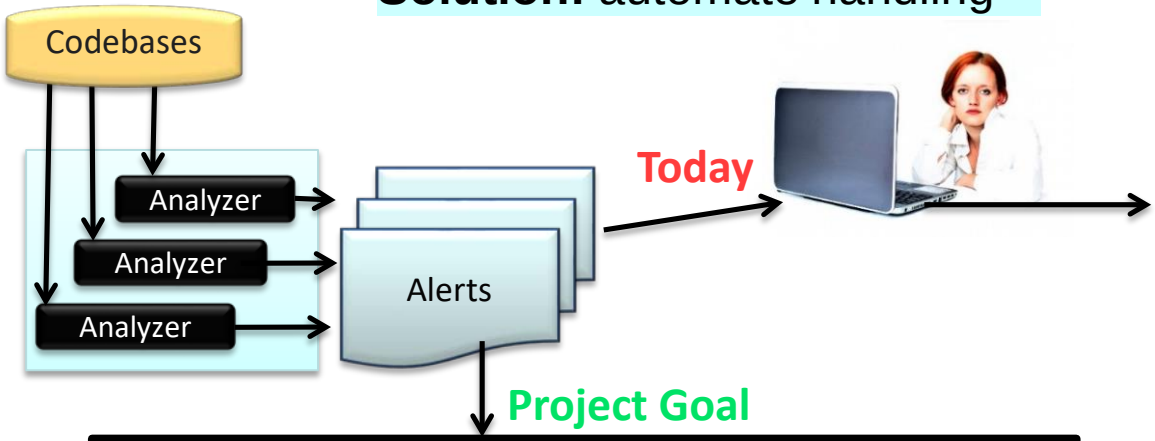


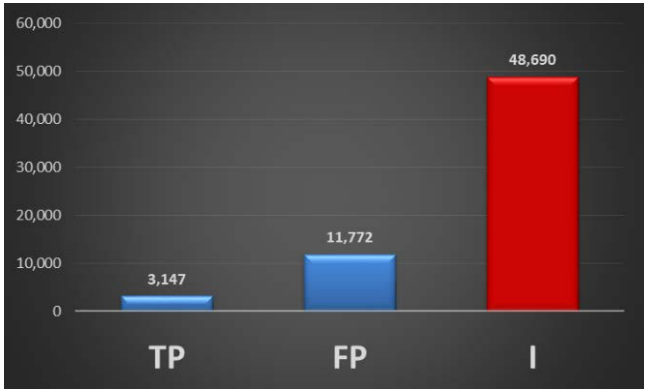
Image of woman and laptop from http://www.publicdomainpictures.net/view-image.php?image=47526&picture=woman-and-laptop "Woman And Laptop"

Classifiers

Problem: too many alerts
Solution: automate handling

Alert: an SA warning (with a tool checker ID, line #, filepath, message)
AlertCondition: an alert mapped to a code flaw taxonomy item (e.g., CWE-190)
Meta-alert: mapped to by the set of alertConditions that differ only by checker ID.
We do adjudication and classification at meta-alert level





Project Goal

Classification algorithm development for CI systems, that precisely and with high recall, classifies 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)

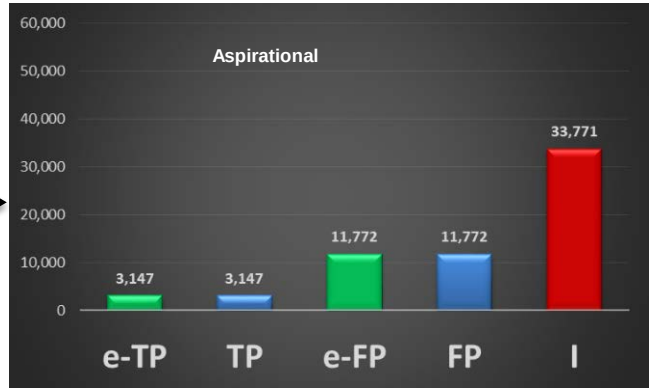


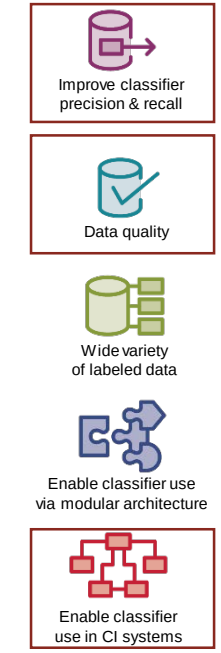
Image of woman and laptop from http://www.publicdomainpictures.net/view-image.php?image=47526&picture=woman-and-laptop "Woman And Laptop"

Approach

DoD is moving to CI/CD but doesn't have a solution to this problem

Enable adjudicating static analysis (SA) meta-alerts quickly during CI

- Automated classifier use to reduce manual effort making SA meta-alert adjudications
- Research required for effective classifier use in multi-tool CI/CD systems
  - Improve correctness of meta-alert adjudication cascading between code versions (data quality)
  - Use of features intended to improve classifier precision and recall
    - Semantic features
    - More features: mean time to deploy, build system data, repository logfile, developer+org ID [1], function, file path



Alert Classification in CI: State of Practice

No existing CI/CD-integrated tool for multiple static analyses using classifiers OR precise cascading

Sophisticated SA tools enable CI integration. Classifiers missing or unconfirmed.

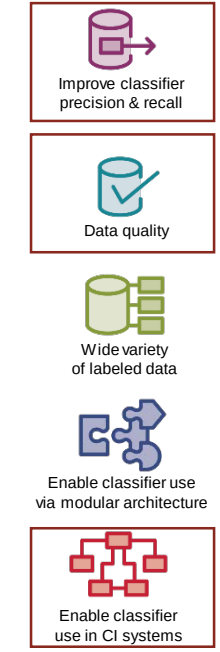
- AlertConditions for particular flaws *may* halt a CI stage
- If use classifiers, not advertised (Coverity, Fortify, etc.)
- We have helped one vendor starting to research classifiers since Nov'18
- There is no known multi-tool aggregator that uses classifiers other than SEI SCALe
- No classifiers in cost-free open-source tools like Cppcheck and GCC

Some single-tool alertCondition-adjudication cascading between code versions

- Proprietary methods for proprietary tools

There is no multi-tool aggregator that cascades using semantic analysis

- Some aggregators (including SEI SCALe) use `diff` to cascade



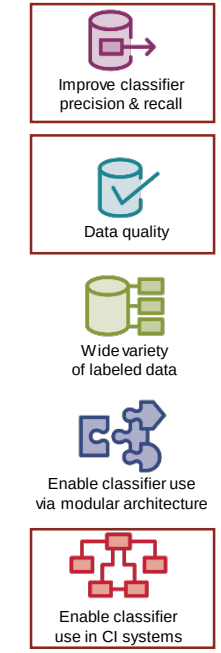
Meta-alert Classification in CI: State of Practice re. Tools

NSA CAS benchmarked SA tools on test suites. Recent and older [1] [2] studies

- Tool performance varies, best tools proprietary
- Multiple tools:
  - Cover many code flaws (individual tools cover different flaw sets)
  - Problem of too many alerts worse with multiple tools

Our strategy: use classifiers to deal with the large number of alerts from multiple tools

- Cost savings and/or increased code security from automated adjudication
- May enable practical strategic mix of free tools with proprietary for affordable coverage of many code flaws



[1] National Security Agency (NSA) Center for Assured Software (CAS). CAS Static Analysis Tool Study – Methodology. 2011. http://samate.nist.gov/docs/CAS_2011_SA_Tool_Method.pdf.
[2] National Security Agency (NSA) Center for Assured Software (CAS), CAS Static Analysis Tool Study – Methodology. 2012. http://samate.nist.gov/docs/CAS%202012%20Static%20Analysis%20Tool%20Study%20 Methodology.pdf.

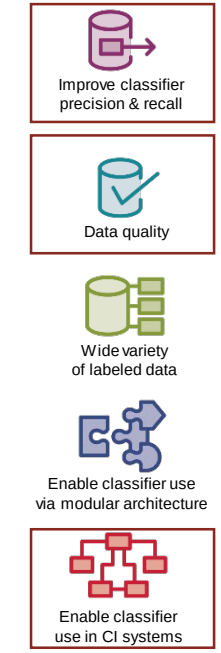
Two Methods of Alternative Incomplete Approaches

Methods:

1. Adjudicate very few alert types in CI
2. Run SA automatically in CI but don't adjudicate during CI

DoD examples:

- Project X (anonymized)
  - SA alertConditions do not break commits or merges during development
  - High/critical un-addressed alertConditions prevent release
  - One goal of SEI work on this project is to make security analyses more integrated with production CI/CD
- Project Y (anonymized)
  - Two proprietary SA tools in CI/CD pipeline, run all checkers (per-tool) every build
  - Some new alertConditions break build: select conditions require review, but ignore many other conditions
- Collaborator G (anonymized FY16 collaborator)
  - Used single SA tool, with cascading but no classifiers
  - Made manual adjudications on subset of code flaw conditions



Meta-alert Classification in CI: Technical Underpinnings

Build on previously-developed SEI tools and research advances:

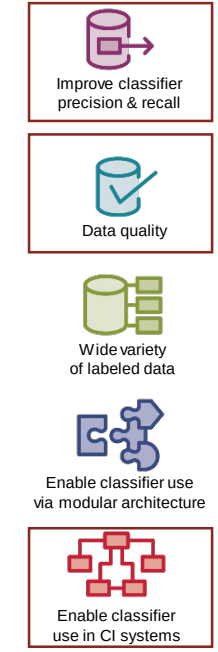
- SCAIFE prototype modular system to do meta-alert classification and prioritization
- CI/CD systems: DevSecOps tools and techniques (Hasan Yasar)

Build on (Iowa State Univ.) Prof. Wei Le's previous work for C:

- Matching defects from one code version to others
- Similar idea but specific to meta-alert matching, for C++ and Java
  - Templates in C++
  - Polymorphism and exception handling must be handled for C++ and Java

Classifier research

- Semantic and other "CI/CD" features

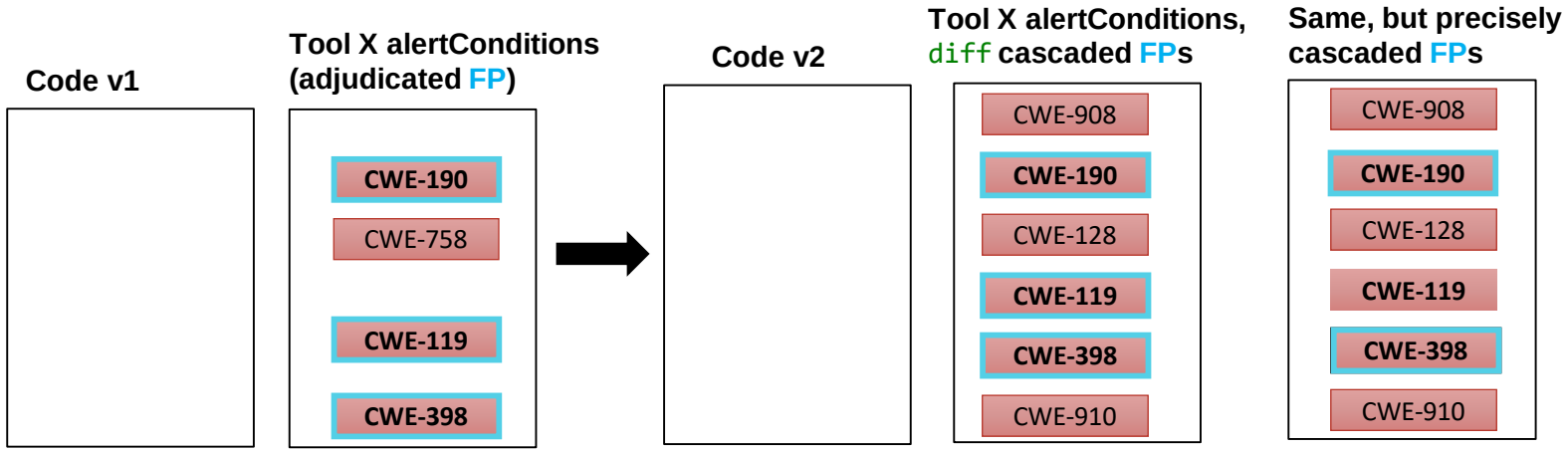


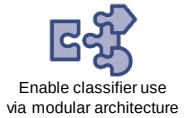
Precisely Cascading Adjudications from Previous Code Versions

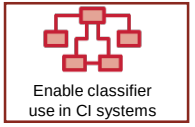
1. Build on Prof. Wei Le's previous work for C
2. Instead of matching patch suitability, determine if meta-alert adjudication should be cascaded
3. Develop a new static analysis to match flaws in different versions of Java or C++ code
4. Test programs for correctness. Test on open source codebases, compare to `diff` cascading to measure improvement.

Challenges

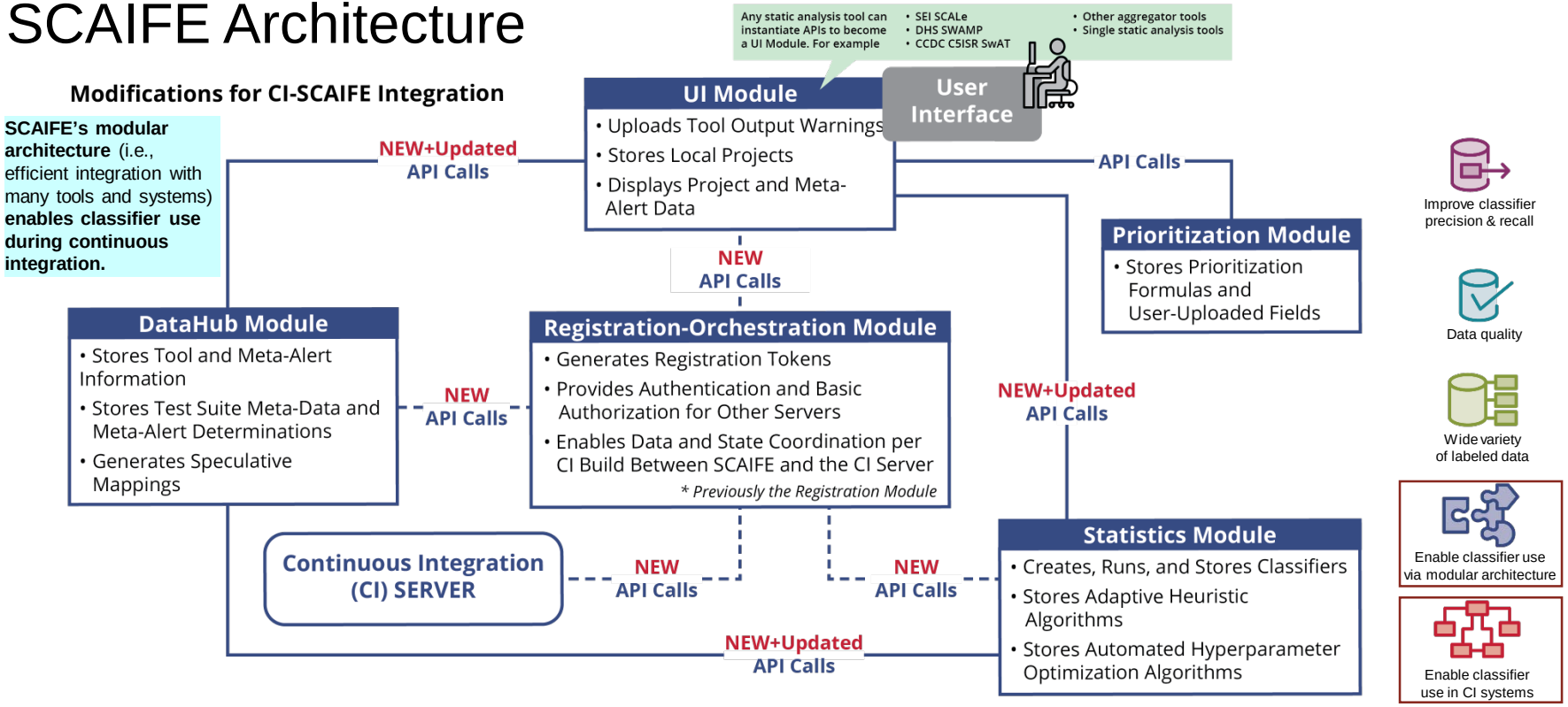
- Templates in C++
- Polymorphism and exception handling must be handled for C++ and Java
- Algorithm must be fast to work in CI system







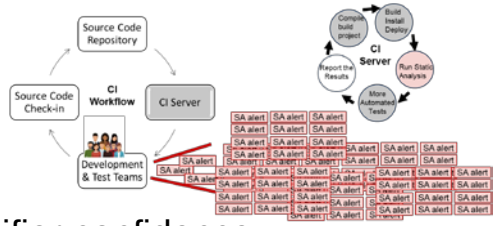
SCAIFE Architecture



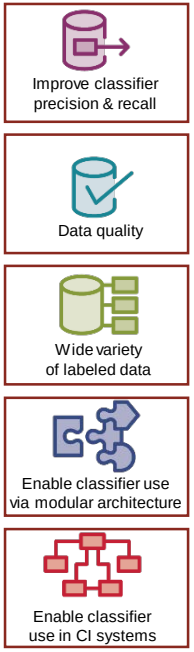
L. Flynn, E. McNeil, and J. Yankel. "How to Instantiate SCAIFE API Calls: Using SEI SCAIFE Code, the SCAIFE API, Swagger-Editor, and Developing Your Tool with Auto-Generated Code." SEI Technical Manual. July 2020.

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

Integrated CI-SCAIFE Design Highlights



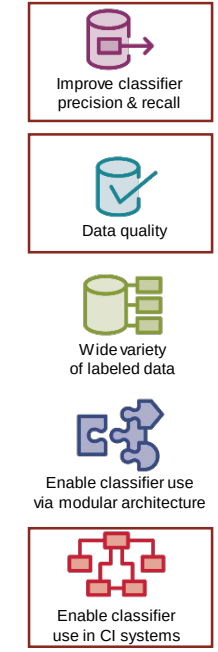
- **SCAIFE-fail or SCAIFE-pass**
  - Fail: If any critical condition meta-alert lacks a cascaded FP ***and*** classifier confidence FP is less than the threshold.
  - Pass: All other cases
- **The CI build only passes if SCAIFE *and* other tests all pass**
- **Complex design aspects:** tracking all build data through SCAIFE, enabling
  non-build data to improve the classifier simultaneously, making it all fast
- Project specifies critical build conditions (e.g., CWE-190 and INT31-C)
- Project specifies confidence threshold (e.g., 90%) for classifier predictions
- CI sends build data to SCAIFE
  - Code change commit data and associated tool output
- SCAIFE cascades adjudications
- SCAIFE classifies remaining non-adjudicated meta-alerts



Meta-alert Classification in CI: Impact

If this project is successful:

- Organizations that develop tools and analyze code
  - Cut number of alerts manually adjudicated in half (save $$s), double adjudicated meta-alerts (more security at same cost), or some mix of cost-savings and increased adjudication
  - By integrating with CI, catch and fix more SA-identified flaws early in development, saving money
  - Use precise cascader developed in this project to improve code security analyses.
  - Use other code and algorithms developed in this project (e.g., SCAIFE system, API, and classification/active learning) to enable practical meta-alert classification in their systems
- Targeted on-ramps for transition:
  - Research project collaborators
  - Discussions started with SEI engineers on DoD contract projects
    - One project could analyze double the SA meta-alerts with the same effort
    - Another project could integrate SA meta-alert adjudication in their CI

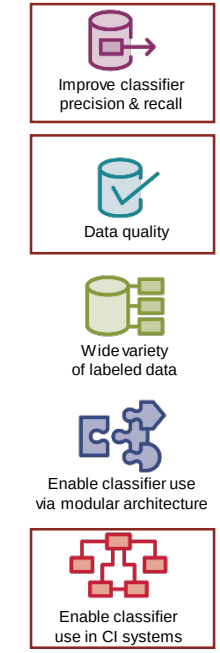


SA Classification in CI: Relevance/Impact for General DoD State of the Practice

Enable the DoD to more efficiently address SA meta-alerts in CI/CD time constraints, by halving time to manually adjudicate meta-alerts for the same level of security.

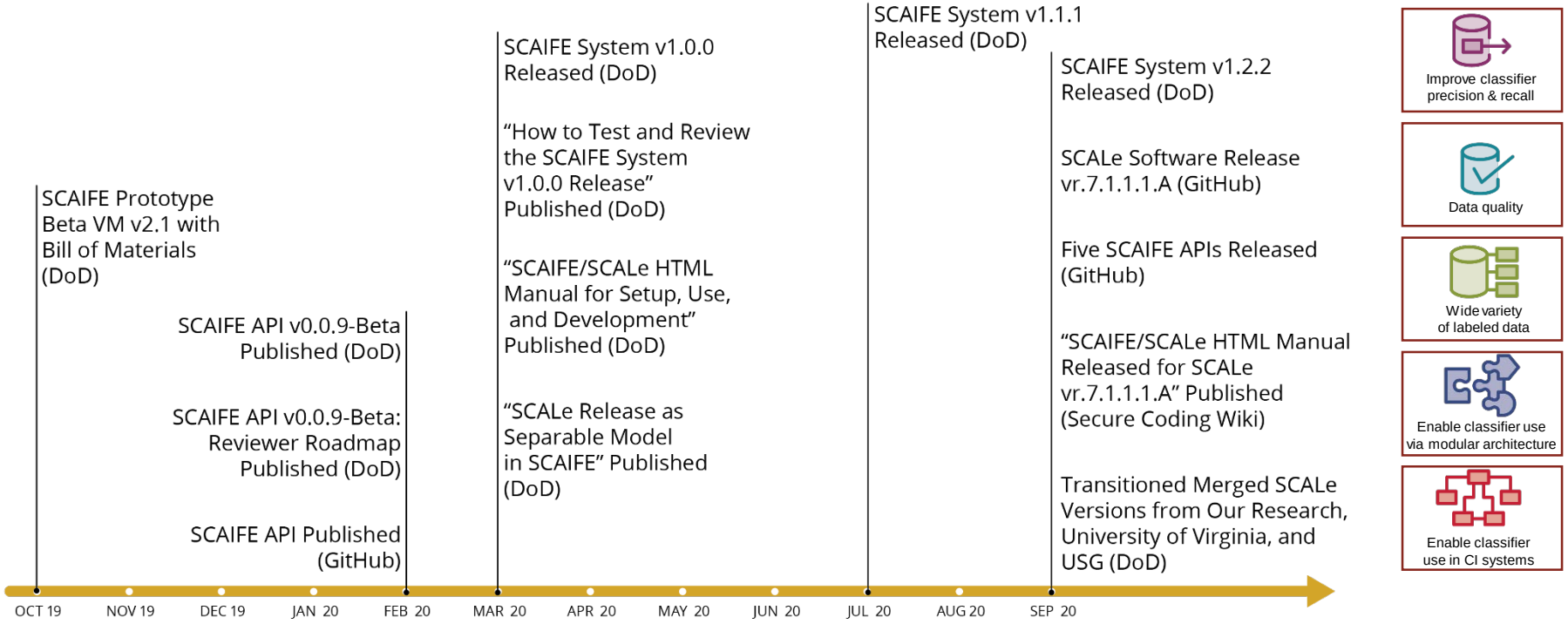
Envisioned classifier-use scenario in Authorization to Operate (ATO):

- DoD Program PMO must provide evidence how software risks managed
  - PMO needs ATO by Authorizing Official
  - How to do this for CI/CD systems is pretty much being developed + experimented, now
    - Possibly CATO (Continuous ATO) option
      - CWEs and other flaw conditions might be required to adjudicate meta-alerts and fix TPs
- We envision this classifier-use scenario in CATOs:
  - CATO covers more code flaw conditions
  - Meta-alerts classified expected-False would not require manual adjudication
  - Even if condition not mentioned in a CATO, classifier use frees more adjudication effort



FY20: Select Code/API Artifacts

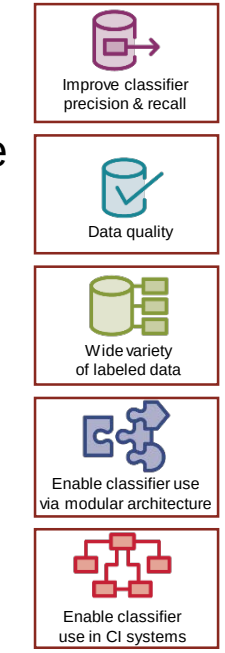
- DoD can get full implementation
- SCALe + SCAIFE API publicly-published (Sept 2020 versions)
- Significant CI integration; to be completed in FY21



Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

FY20 Select Artifacts (New Detail or Item) –1

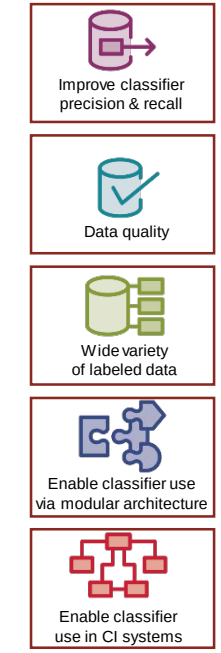
- (Oct 2019 and Feb, April, and Sept 2020) GitHub publication of SCAIFE API versions https://github.com/cmu-sei/SCAIFE-API
- (04/2/20) Published the open dataset “RC_Data” for classifier research to the SEI CERT Secure Coding webpage “Open Dataset RC_Data for Classifier Research”. Database with static analysis alerts from open-source tools, adjudications, code metrics, and more for two codebases.
- (06/18/20) Presentation “Automated Classifiers to Adjudicate Static Analysis Alerts: Challenges, Progress, and Next Steps” (Lori Flynn, Stephen Adams, and Tim Sherburne) to DoD’s DEVCOM Cyber Community of Interest.
- (06/25/20) Presentation “Automated Classifiers to Adjudicate Static Analysis Alerts: Challenges, Progress, and Potential Collaborations with NASA IV&V” (L. Flynn) to leaders of the NASA IV&V Static Code Analysis Working Group (SCAWG).



Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

FY20 Select Artifacts (New Detail or Item) –2

- (07/8/20) Technical manual "How to Instantiate SCAIFE API Calls: Using SEI SCAIFE Code, the SCAIFE API, Swagger-Editor, and Developing Your Tool with Auto-Generated Code" (L. Flynn, E. McNeil, and J. Yankel) Instructions for three types of SCAIFE System code access: (1) none, (2) access to SCALe code, or (3) full access.
- (07/13/20) Auto-generated Java client code for the five SCAIFE API modules for a DoD collaborator, to help them quickly start to instantiate SCAIFE API calls from their tool
- (09/14/2020) Blog post "Managing Static Analysis Alerts with Efficient Instantiation of the SCAIFE API into Code and an Automatically Classifying System" by Lori Flynn
- (Sept. 2020) SCALe code at https://github.com/cmu-sei/SCALe/tree/scaife-scale
- (Sept. 2020) Test data generated with 'diff' cascading, for comparison to precise cascading
- (09/22/2020) Presentation "Rapid Adjudication of Static Analysis Meta-Alerts During Continuous Integration", Software Assurance Community of Practice (SwA CoP).
- (09/30/2020) Special Report "DoD Collaborators: FY20 Deliveries, Validation, & Next Steps"



Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

Static Analysis Classification Research FY16-20

Invitation to Collaborate

DoD Orgs that do CI Development: Invitation to Test

I need DoD collaborators that do CI development, to test our tooling

- Current collaborators test but not doing CI
- Full system implementation release currently limited to DoD
- CI testing does not have to include data sharing (next slide)
- If interested please contact me lflynn@cert.org

Deployment and testing supported by project

- release system containerized and with configuration files (ports, URLs, names) to ease integration in wide variety of systems
- comes with much documentation, we've extended that a lot in last year per collaborator feedback
- Part of FY21 project specifically is for helping collaborators use the system

All: Might you be able to help us get labeled data?

Effort to label data on particular open-source codebases

SCALe (scaife-scale branch) on GitHub can be used to do the adjudication and store results

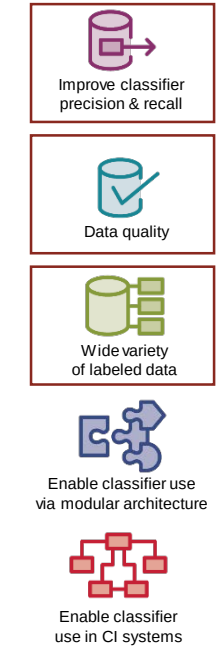
Even better, SEI can provide full SCAIFE system to DoD orgs (includes SCALe + classification etc.)

Auditing self-training support via published materials (next slide)

Possibly your own stored archives, sanitized before sharing

High-quality manually labeled data would help us improve our DoD sponsored classification research.

If our research succeeds, the improved classification techniques and data will help your orgs to secure your code and save money.



Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

Self-Training Resources for Auditing Meta-Alerts

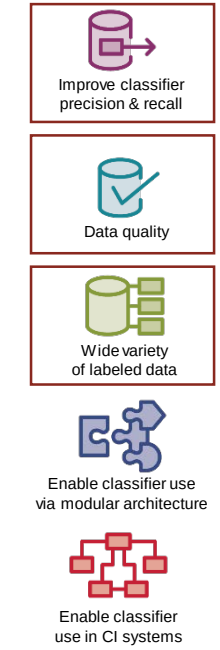
- Paper "Static Analysis Alert Audits: Lexicon & Rules" (D. Svoboda, L. Flynn, W. Snavely) IEEE SecDev
- Presentation "Hands-On Tutorial: Auditing Static Analysis Alerts Using a Lexicon and Rules" (L. Flynn, D. Svoboda, W. Snavely) https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=505451
- Webcast (1 hour video, hands-on SCALe use): "Improve Your Static Analysis Audits Using CERT SCALe's New Features" by L. Flynn. (The SCAIFE System includes the SCALe tool, as a separable part of SCAIFE.) https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=538843 (video) and https://resources.sei.cmu.edu/asset_files/Presentation/2018_017_101_532198.pdf (slides)
- Video "Rapid Construction of Accurate Automatic Alert Handling System" Nov. 2019 https://youtu.be/dwYbhgko3to
- Slides "Rapid Construction of Accurate Automatic Alert Handling System" Nov. 2019 https://resources.sei.cmu.edu/asset_files/Presentation/2019_017_001_635435.pdf

It will increase the quality of data if your team studies definitions of the code flaw types ("conditions") they will inspect static analysis meta-alerts for, as defined in a formal code flaw taxonomy.

For this classification research, the taxonomies currently of the most interest are:
- MITRE CWE https://cwe.mitre.org/data/index.html
- CERT coding rules for C: https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard
- CERT coding rules for Java: https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java
- CERT coding rules for C++: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682

The SCALe (scaife-scale branch) GitHub release includes a SCAIFE/SCALe HTML manual with extensive information about how to use the SCAIFE and SCALe systems to adjudicate (aka 'audit) static analysis meta-alerts.

Goal: Enable **practical** automated classification, for more secure software & lower cost/effort



Static Analysis Classification:
Line-Funded Research FY16-20

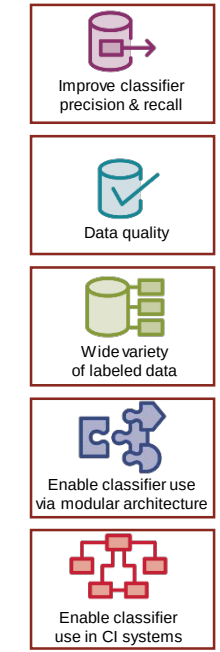
Impacts Time Frame

AI Engineering-Related Topics

- Robust Systems: V&V, Tools & Process, Secure Coding
- Data, Devices, and Computing: Scalability, Performance and Evaluation

Project Impacts Time Frame

NEAR	MID	FAR
Public can use/review SCAIFE API and SCALe* module. DoD collaborators will further test SCAIFE to provide data and feedback integrate their tools using the API The FY20-21 research project incorporates continuous integration (CI) into architecture design.	More collaborators (DoD and non-DoD) to test SCAIFE with CI. Design improvements for transition include classification precision latencies bandwidth/disk/memory use business continuity scalability	A wide variety of systems will do automated meta-alert classification, using SCAIFE System SCAIFE API Goal: Provide better software security, or less time and cost for the same security (DoD and non-DoD).

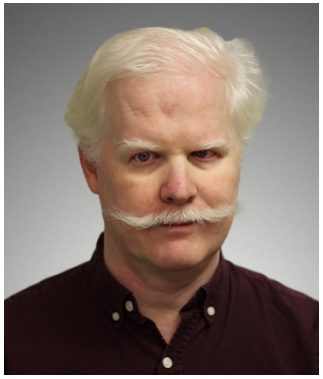


** Version of SCALe used in SCAIFE System implementation*

Goal: Enable **practical** automated classification, for more secure software & lower cost/effort

FY20 Project Team





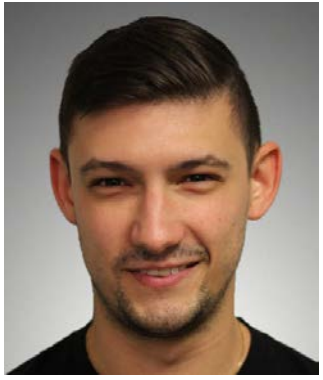

Dr. Lori Flynn

Ebonie McNeil

David Svoboda

Matt Sisk






Hasan Yasar

Joseph Yankel

Shane Ficorilli

David Shepard

Thanks + Contact Info

Thank you for listening!

Questions?

Feedback and potential collaborations are welcome, here's my contact info:

Lori Flynn, PhD

Senior Software Security Researcher

lflynn@sei.cmu.edu

Carnegie Mellon University

Software Engineering Institute

4500 Fifth Avenue

Pittsburgh, PA 15213-2612