Carnegie Mellon University
Software Engineering Institute

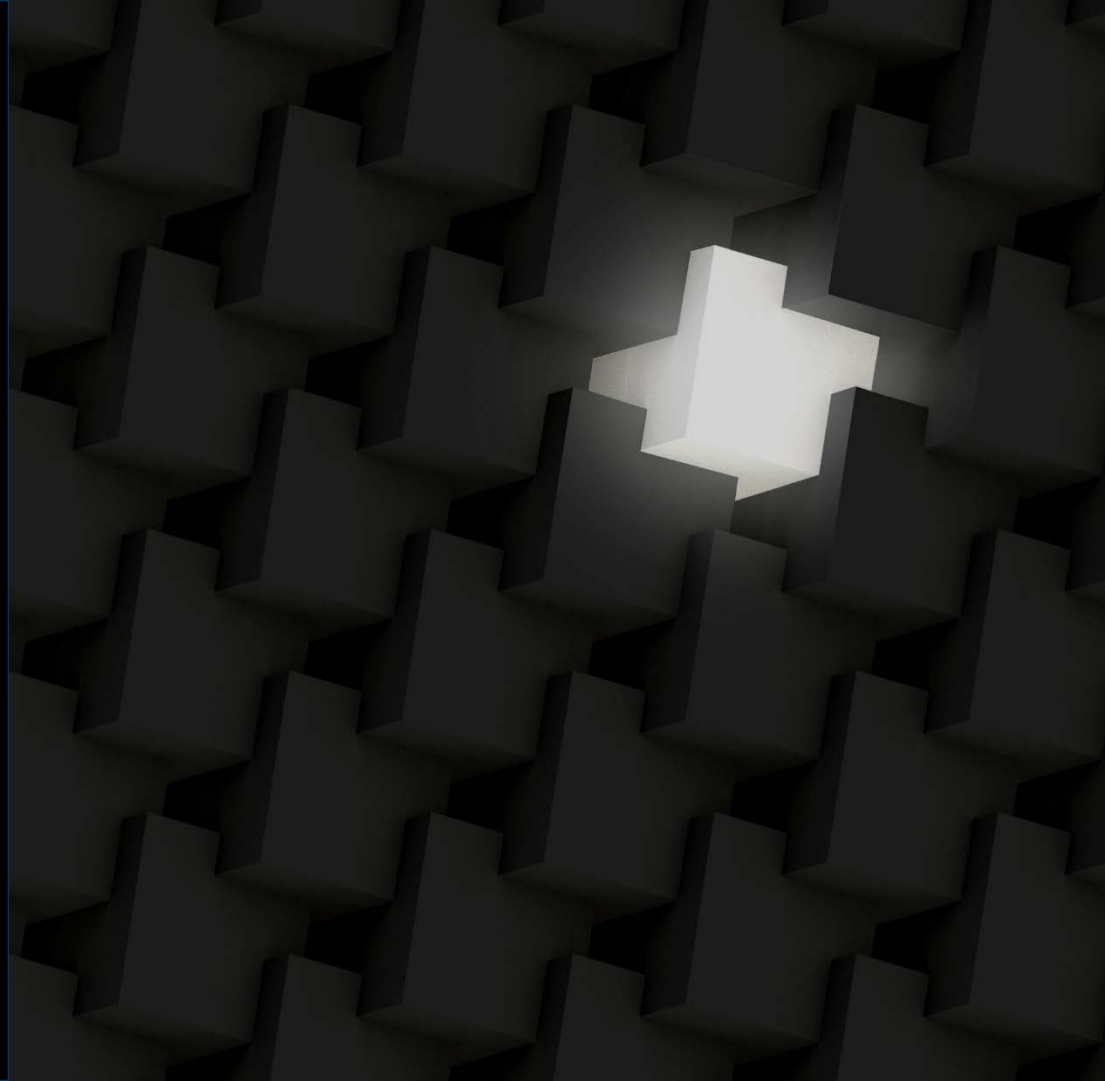
# RESEARCH REVIEW 2020

## Rapid Adjudication of Static Analysis Alerts During CI

Presenter: Dr. Lori Flynn (PI)

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





# 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.



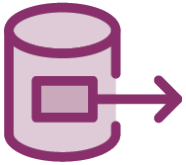
DM20-0858

# FY20 Line-Funded Project

- Today's presentation focuses on our FY20 project.
- This project builds on tools and techniques from previous projects.
- For an overview of my FY16-FY20 projects, refer to my Tuesday Research Review presentation.

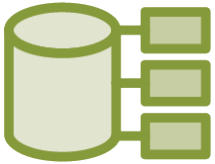
**FY20 Project Main Focus**

## Highlighted Research Focus Areas

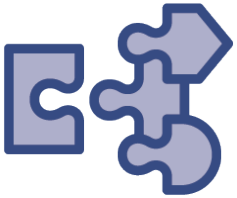


Improve classifier precision & recall

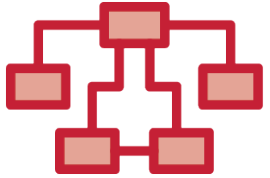
Data quality



Wide variety of labeled data



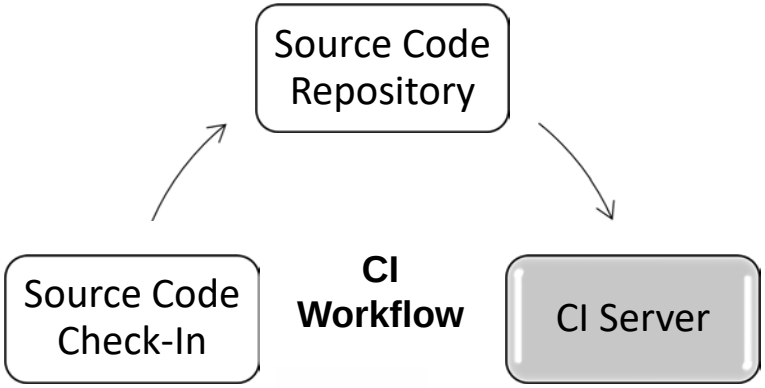
Enable classifier use via modular architecture

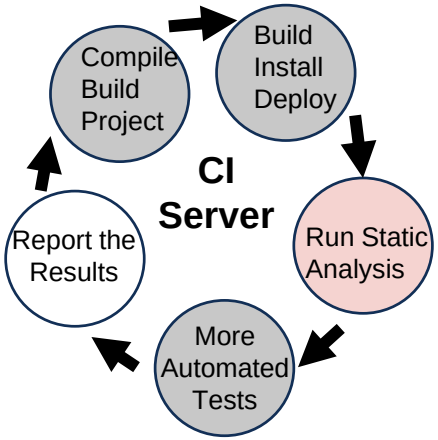


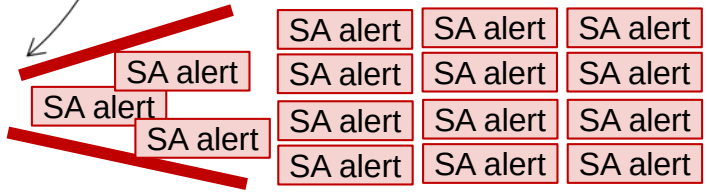
Enable classifier use in CI systems

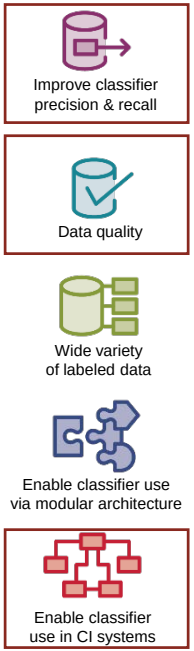
Goal: Enable **practical** automated classification for more secure software and lower cost/effort.

# Rapid Adjudication of Static Analysis (SA) Meta-Alerts During CI



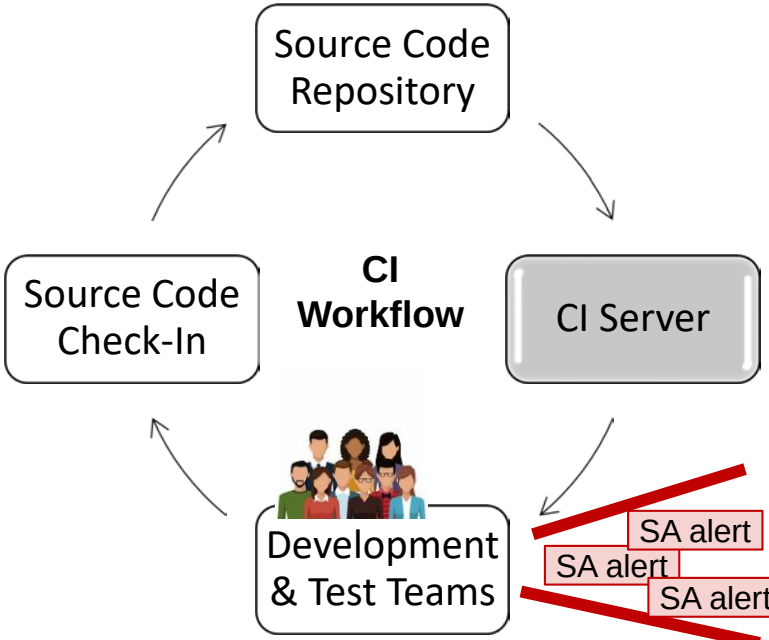



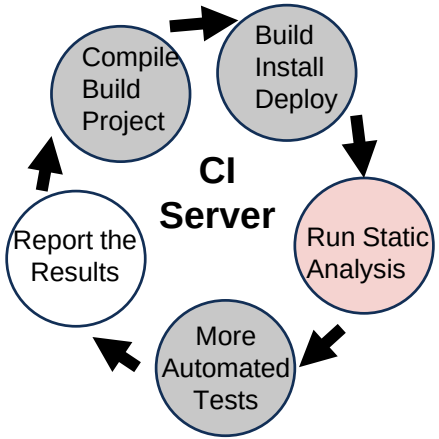



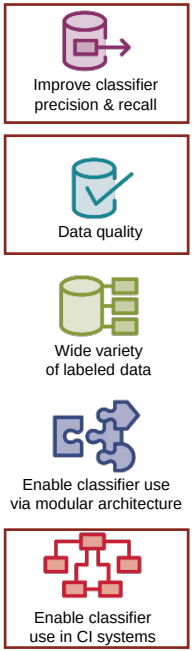

**Alert***:* An SA warning (with a tool checker ID, line #, filepath, message)
**AlertCondition**: An alert mapped to a code flaw taxonomy item (e.g., CWE-190)
**Meta-alert***:* mapped to by the set of alertConditions that differ only by checker ID.
**We do adjudication and classification at the meta-alert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

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

# Rapid Adjudication of Static Analysis Meta-Alerts During CI



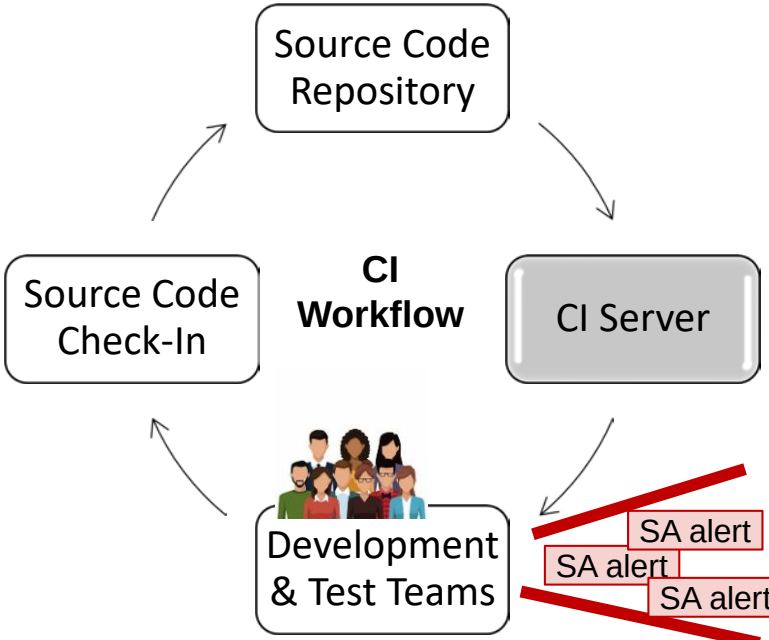



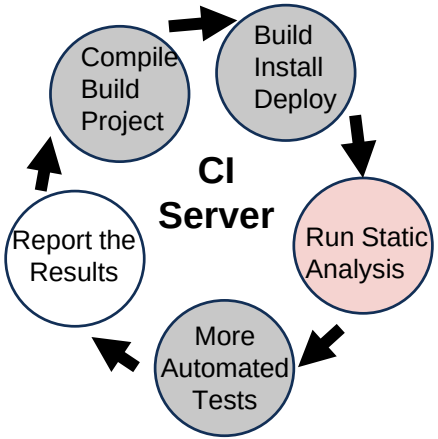




SA alert SA alert SA alert SA alert SA alert SA alert
SA alert SA alert SA alert SA alert SA alert SA alert
SA alert SA alert SA alert SA alert SA alert SA alert
SA alert SA alert SA alert SA alert SA alert SA alert

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

# Rapid Adjudication of Static Analysis Meta-Alerts During CI

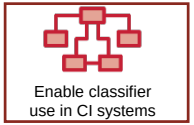




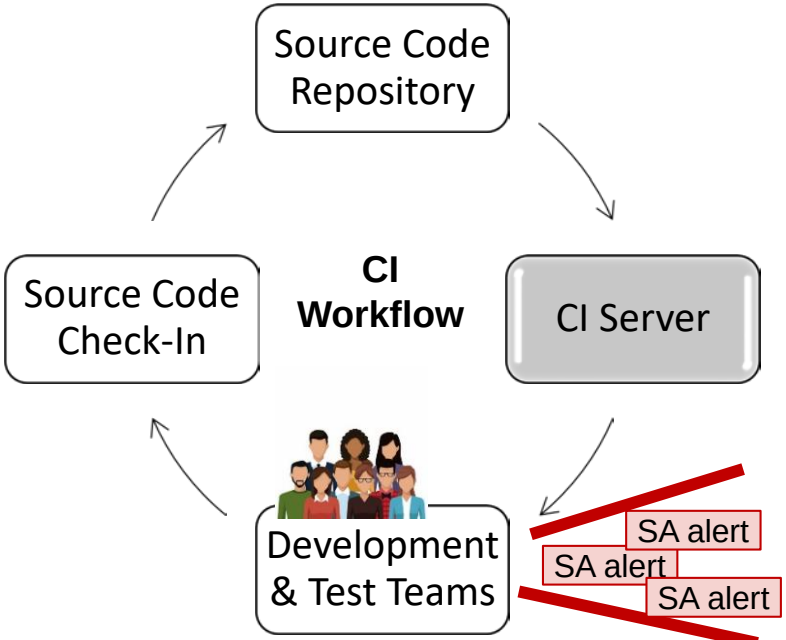


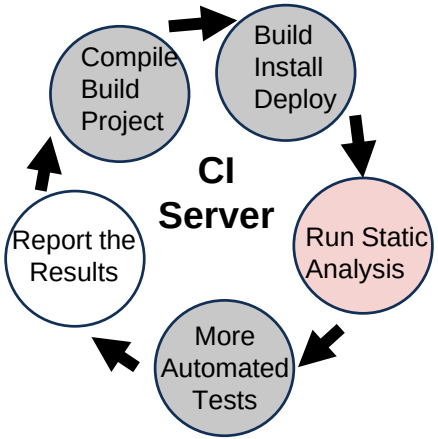





Goal: Enable **practical** automated classification for more secure software and lower cost/effort.

# Rapid Adjudication of Static Analysis Meta-Alerts During CI



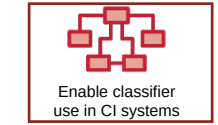





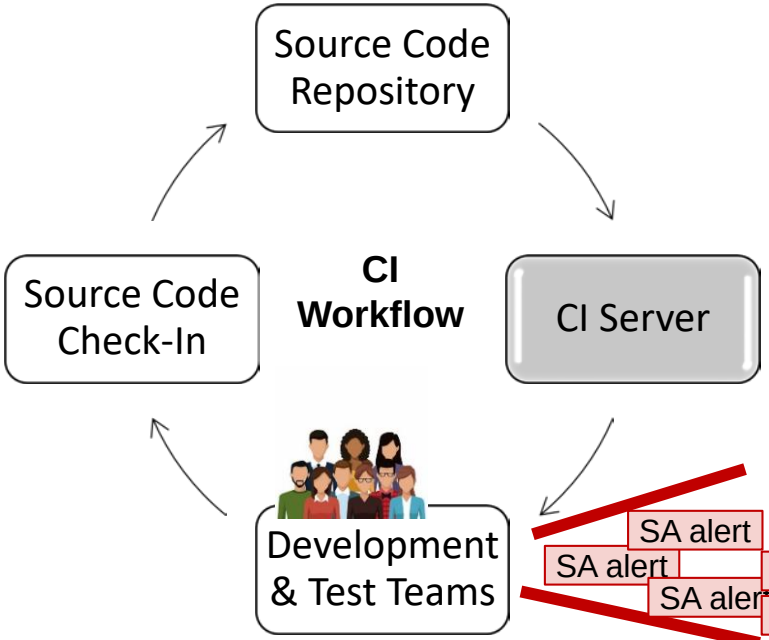
Goal: Enable **practical** automated classification for more secure software and lower cost/effort.

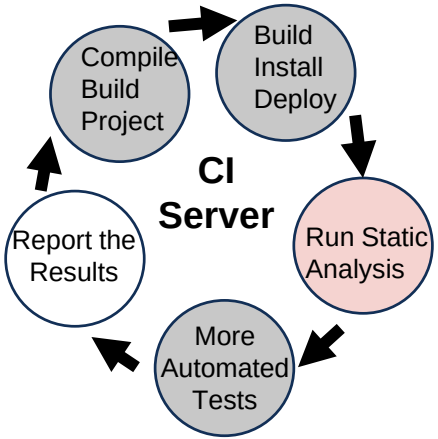






# Rapid Adjudication of Static Analysis Meta-Alerts During CI





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




# Rapid Adjudication of Static Analysis Alerts During CI

The 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 into CI development projects in the DoD, and the selection of SA tools is limited to those with very few false positives.

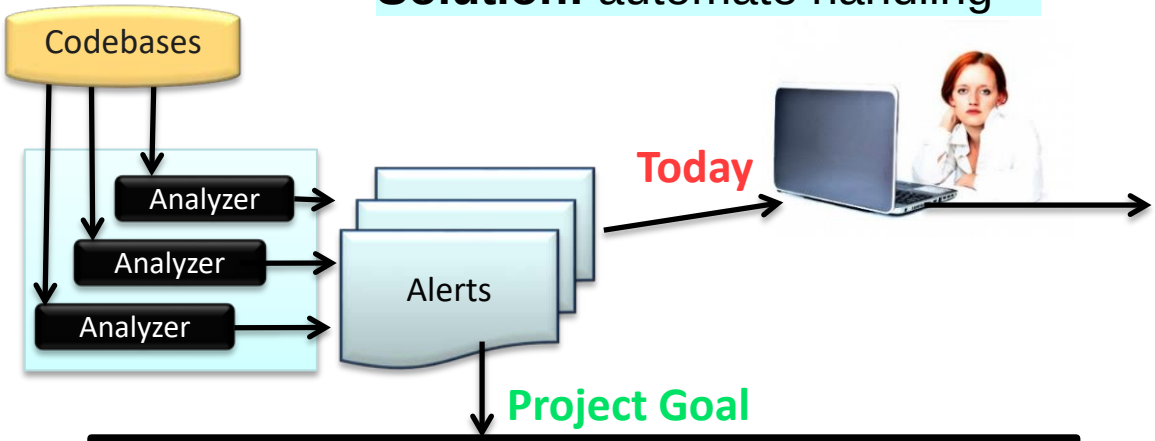
The current practice is too labor intensive. We will automate it.

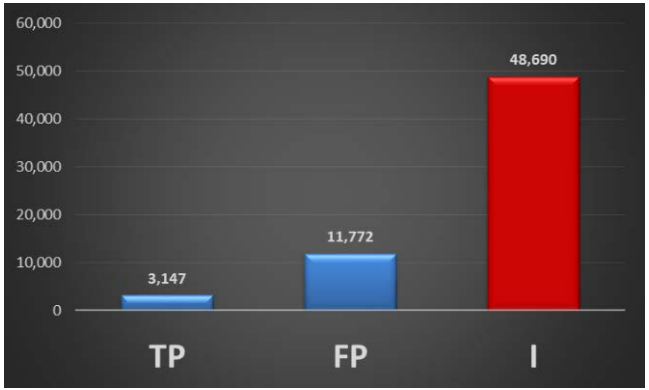
Goal: Enable **practical** automated classification for more secure software and lower cost/effort.

# 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)

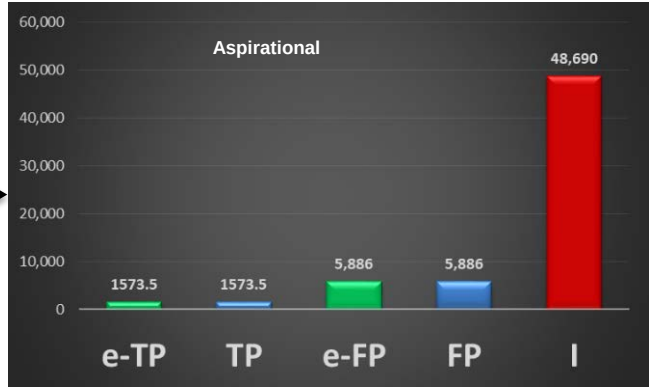
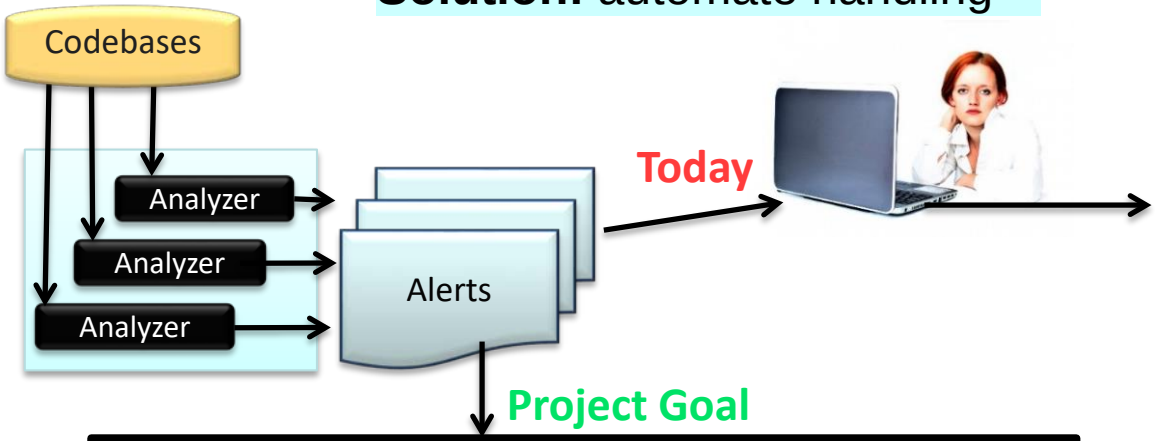


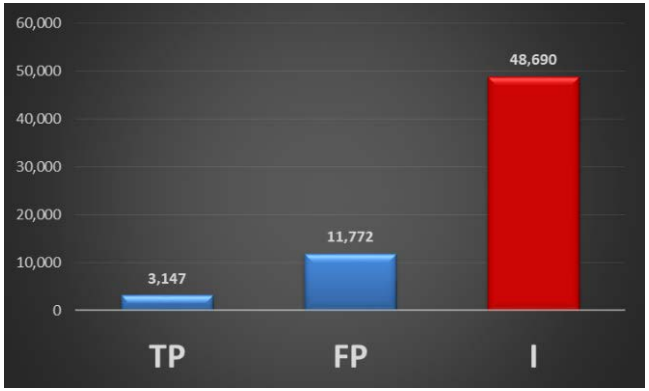

Image of woman and laptop from http://www.publicdomainpictures.net/view-image.php?image=47526&picture=woman-and-laptop "Woman And Laptop"

# 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)

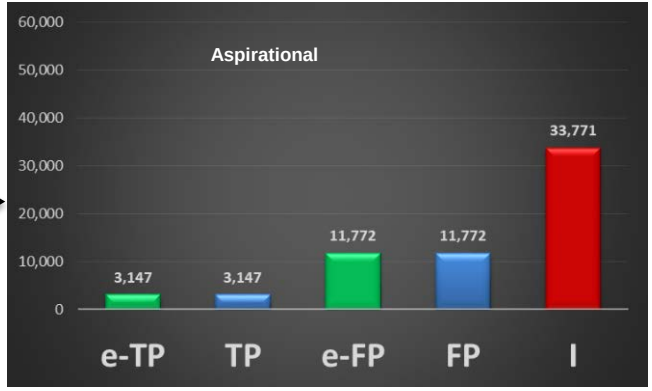



Image of woman and laptop from http://www.publicdomainpictures.net/view-image.php?image=47526&picture=woman-and-laptop "Woman And Laptop"

# Approach

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

Our approach enables SA meta-alerts to be adjudicated quickly during CI:

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

[1] Cheirdari, Foteini and George Karabatis. "Analyzing False Positive Source Code Vulnerabilities Using Static Analysis Tools." Workshop on Big Data for CyberSecurity. December 2018.

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

# Approach: Cascading and Classification

## 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.

## Classification

- It predicts if an SA meta-alert is true or false based on statistics/AI.
- It is different from SA tool prediction of severity/risk/cost. (All SA meta-alerts are predicted to be true by SA tools.)

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

# RESEARCH REVIEW 2020

Rapid Adjudication of Static Analysis Alerts During CI

## State of the Art/Practice

# SA Classification in CI: State of Practice

There is no existing CI/CD-integrated tool for multiple static analyses using classifiers OR precise cascading.

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

- AlertConditions for particular flaws *may* halt a CI stage.
- SEI SCALe is the only multi-tool aggregator that uses classifiers.
- There are no classifiers in open source tools like Cppcheck and GCC.

There is some single-tool alertCondition-adjudication cascading between code versions.

- Proprietary methods are used in proprietary tools.

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

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

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

# SA Classification in CI: State of Practice—Tools

NSA CAS benchmarked SA tools on test suites. Findings from recent and older [1, 2] studies include the following:

- Tool performance varies; the best tools are proprietary.
- There are pros and cons to using multiple tools:
  - Multiple tools cover many code flaws; individual tools cover different flaw sets.
  - Using multiple tools introduces too many alerts.

**Our Strategy**: Our approach uses classifiers to deal with the large number of alerts from multiple tools.

**Benefits**: Automated adjudication results in **cost savings** and/or increased **code security**. This approach **may enable a practical strategic mix of** free and proprietary **tools to affordably cover many code flaws**.

[1] http://samate.nist.gov/docs/CAS_2011_SA_Tool_Method.pdf
[2] http://samate.nist.gov/docs/CAS%202012%20Static%20Analysis%20Tool%20Study%20 Methodology.pdf

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

# Two Alternate Incomplete Methods to Use SA in CI

**Methods**

1. Adjudicate very few alert types in CI.
2. Run SA automatically in CI, but do not 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.
- Project Y (Anonymized)
  - Two proprietary SA tools in the CI/CD pipeline run all checkers (per tool) on every build.
  - Some new alertConditions break the build. Select conditions require review, but they ignore many other conditions.
- Collaborator G (Anonymized FY16 Collaborator)
  - The collaborator used a single SA tool with cascading but without classifiers.
  - They made manual adjudications on a subset of code flaw conditions.

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

# RESEARCH REVIEW 2020

Rapid Adjudication of Static Analysis Alerts During CI

## Technical Approach and Progress

# Meta-Alert Classification in CI: Technical Underpinnings

We build on previously developed SEI tools and research advances:

- SCALe SA aggregator and meta-alert auditing system
- Test suites used in novel ways for classifier development
- SCAIFE prototype modular system used to do meta-alert classification and prioritization
- CI/CD systems: DevSecOps tools and techniques (Hasan Yasar)

We plan to build on the previous work of Prof. Wei Le, Assistant Professor of CS at Iowa State University:

- 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 features (CI/CD, etc.)

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

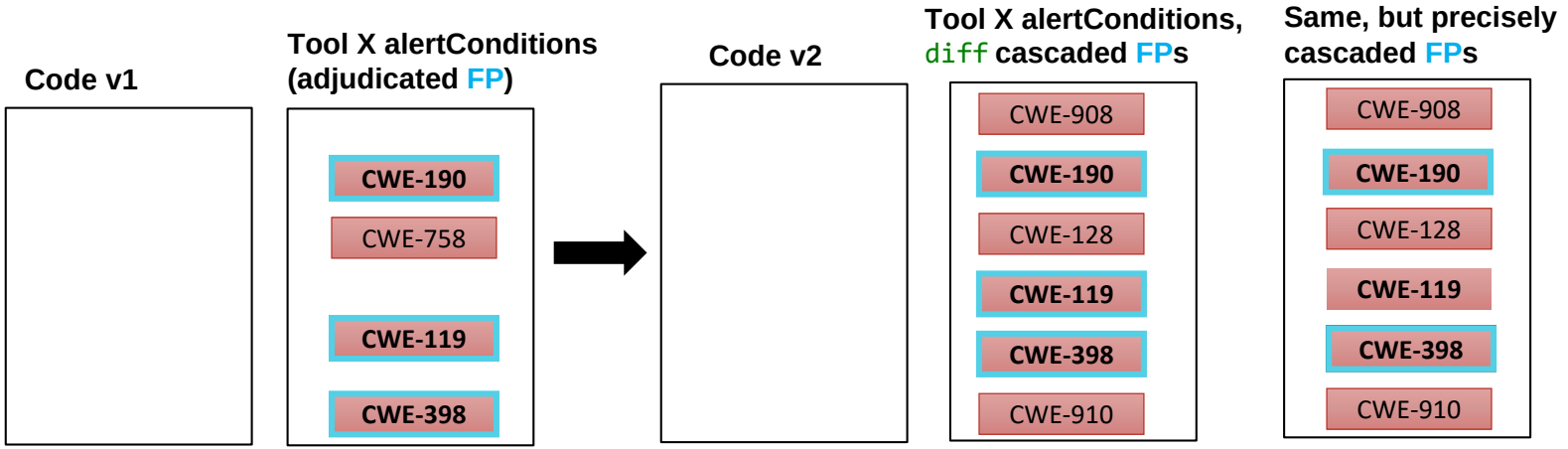
Goal: Enable **practical** automated classification for more secure software and lower cost/effort.

# 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 method that matches 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.

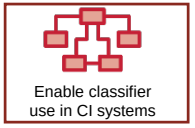
**Challenges**

- Templates in C++
- Polymorphism and exception handling must be handled for C++ and Java
- An algorithm that must be fast to work in the CI system







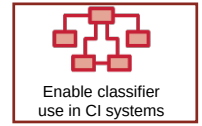



# SA Classification in CI: Technical Approach with Additional Features

1. Generate semantic features using and building a deep belief network (DBN).
2. Apply a dimensionality reduction technique (e.g., principal component analysis) to minimize feature space that is occupied by traditional features.
3. Train existing classifiers (e.g., gradient boosting decision trees) using a combination of traditional and semantic features.
4. Improve classification precision and recall by applying and developing better ML models.
5. Use research techniques such as the following to transfer knowledge for this domain:
   - from artificial test-suite data to natural program data
   - across different codebases of natural programs
6. **Test precision** and **recall** for labeled data and time, counting manually vs. automatically adjudicated.








## Challenges

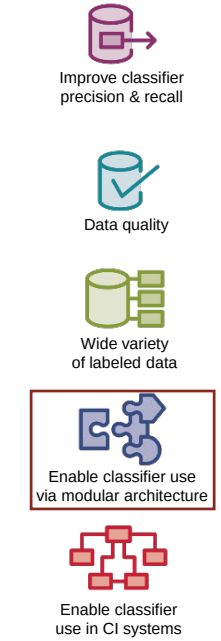
- Incorporate semantic features into the existing ML pipeline without degrading system performance (e.g., time to build classifier).
- Published transfer learning approaches include inductive and transductive transfer learning. Our work is closer to the latter, but it fits neither definition completely. It may be necessary to (1) adjust existing techniques or (2) create a new transfer learning approach.

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

# SCAIFE Definitions

SCAIFE is **a modular architecture that enables static analysis meta-alert classification** and 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 for more secure software and lower cost/effort.

# SCAIFE Architecture FY19

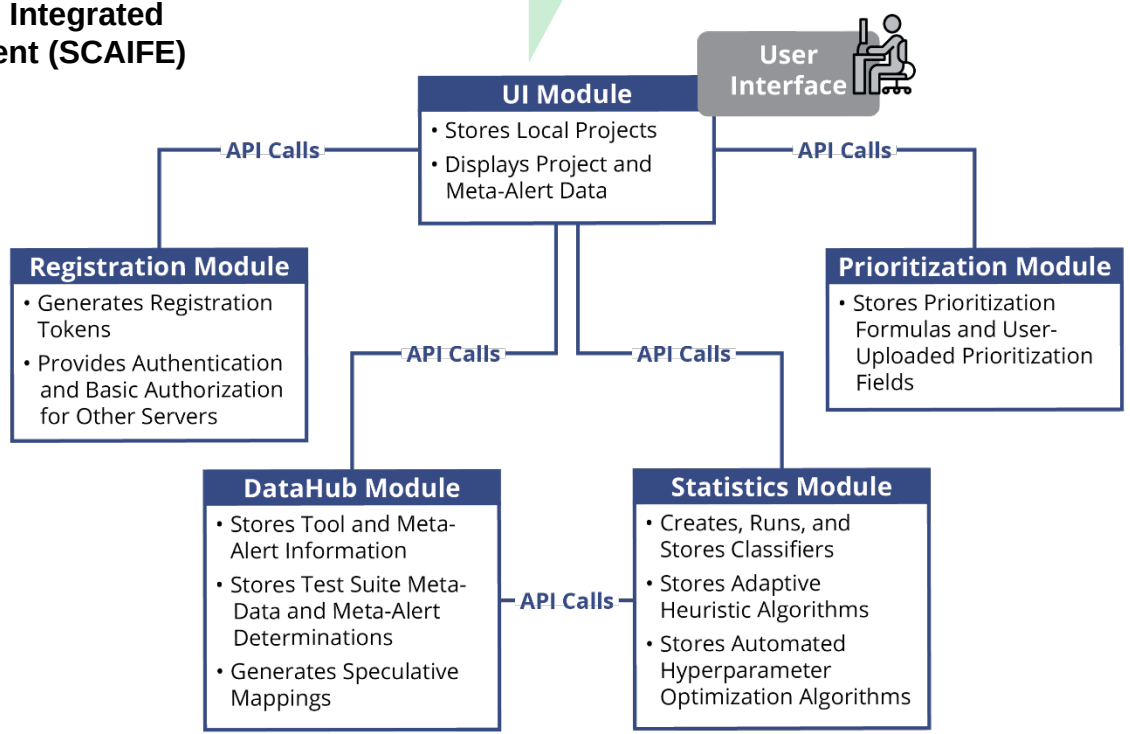
**Any static analysis tool can instantiate APIs to become a UI Module. For example**

- 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.



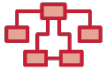


Improve classifier precision & recall


Data quality


Wide variety of labeled data

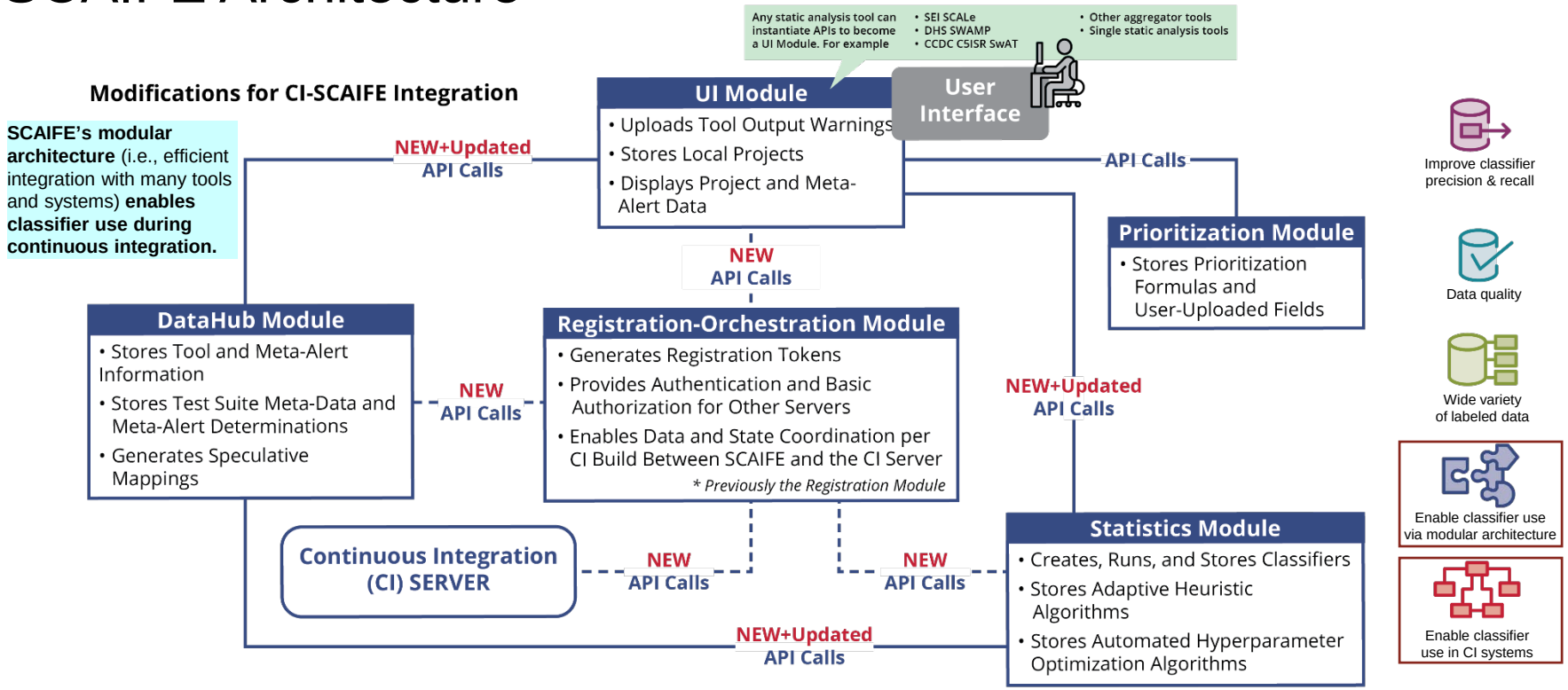

Enable classifier use via modular architecture


Enable classifier use in CI systems

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.

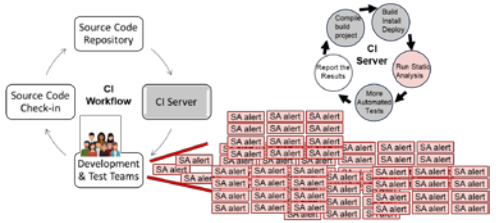
# SCAIFE Architecture FY20




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 is defined as any critical condition meta-alert that lacks a cascaded FP ***and*** classifier confidence FP is less than the threshold.
  - Pass is defined as all other cases.
- **The CI build passes only if SCAIFE *and* all other tests pass.**
- **There are complex design aspects:** (1) tracking all build data through SCAIFE, (2) enabling non-build data to improve the classifier simultaneously, and (3) making it all fast.
- The project specifies critical build conditions (e.g., CWE-190 and INT31-C).
- The project specifies the confidence threshold (e.g., 90%) for classifier predictions.
- CI sends build data (e.g., code change commit data and associated tool output) to SCAIFE.
- SCAIFE cascades adjudications.
- SCAIFE classifies remaining non-adjudicated meta-alerts.

Improve classifier precision & recall

Data quality

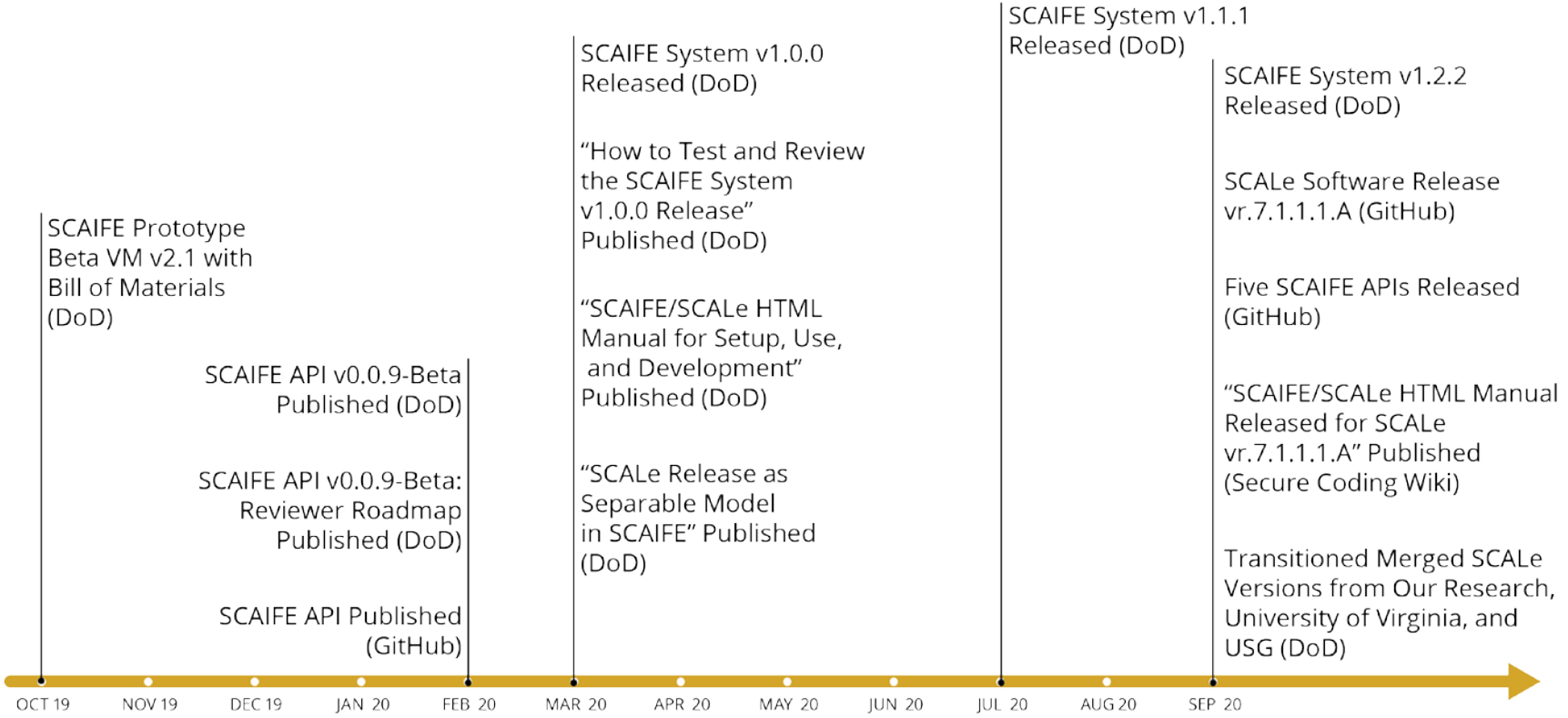
Wide variety of labeled data

Enable classifier use via modular architecture

Enable classifier use in CI systems

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

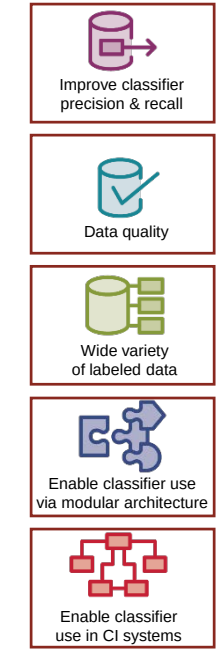
# FY20: Select Code/API Artifacts




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

# FY20 Select Artifacts (New Detail or Item) –1

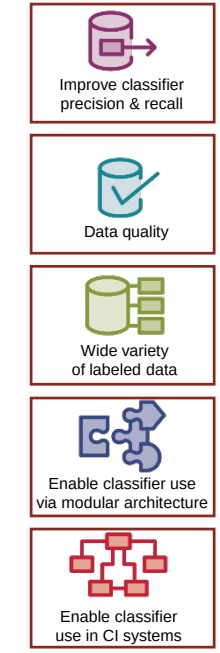
- (Oct 2019 and Feb, Apr, and Sep 2020) GitHub publication of SCAIFE API versions https://github.com/cmu-sei/SCAIFE-API
- (Apr 2020) Published the open dataset "RC_Data" for classifier research to the SEI CERT Secure Coding webpage "Open Dataset RC_Data for Classifier Research"; database with static analysis alerts from open-source tools, adjudications, code metrics, and more for two codebases
- (Jun 2020) 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
- (Jun 2020) 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 and lower cost/effort.

# FY20 Select Artifacts (New Detail or Item) –2

- (Jul 2020) 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
- (Jul 2020) 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
- (Sep 2020) Blog post "Managing Static Analysis Alerts with Efficient Instantiation of the SCAIFE API into Code and an Automatically Classifying System" by Lori Flynn
- (Sep 2020) Presentation "Using AI to Find Security Defects in Code / Build More Secure Software" at the Defense Science & Technology Agency (DSTA) Workshop
- (Sep 2020) Presentation "Rapid Adjudication of Static Analysis Meta-Alerts During Continuous Integration," Software Assurance Community of Practice (SwA CoP)
- (Sep 2020) SCALe code at https://github.com/cmu-sei/SCALe/tree/scaife-scale
- (Sep 2020) Test data generated for 'diff' cascading for precise cascading comparison




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

# Progress Detail: FY20 Collaborations (DoD Anonymized)

**DoD Collaborator A and the University of Virginia (UVA)** tested multiple SCAIFE versions. Our SCALe version was merged with SCALe-UVA new features for collaborator A's use.

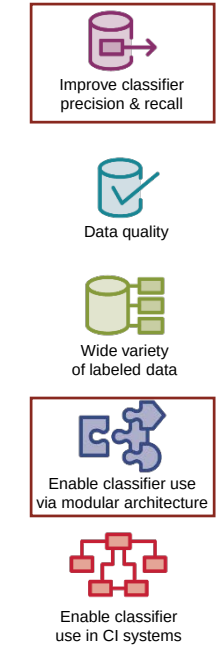
**DoD Collaborator B** tested the SCAIFE release. The SCAIFE API implementation is in-progress (registration of API as a SCAIFE UI Module).

**DoD Collaborator C** tested multiple versions of SCAIFE.

**DoD Collaborator D** provided technical assistance and training material. Testing started in September.

**DoD Collaborator E** added BoM per their requirement; testing starts in FY21.

**The SWAMP** (Mordgridge and UW Madison) collaborator provided SCAIFE API feedback. They discussed possible SWAMP+ SCAIFE API integration, but because of SWAMP funding issues, this integration is not likely.




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

# RESEARCH REVIEW 2020

Rapid Adjudication of Static Analysis Alerts During CI

## Relevancy and Impact

# Measures of Success –1

Organizations that develop tools and analyze code will realize the following cost-saving benefits:

- Manually adjudicated alerts are cut in half (i.e., saves money).
- Adjudicated meta-alerts are doubled (i.e., get more security at same cost).
- Combinations of cost savings and increased adjudication are realized.

By integrating with CI, more SA-identified flaws can be caught and fixed early in development (i.e., saves money).

Using a precise cascader developed in this project, organizations can improve their code security analyses.

Using other code and algorithms developed in this project (e.g., SCAIFE system, API, and classification/active learning) enables practical meta-alert classification in their systems.

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

# Measures of Success –2

Targeted on-ramps for transition include the following:

- Research project collaborators
- Discussions with SEI engineers on DoD contract projects
  - One project could analyze twice the SA meta-alerts with the same effort.
  - Another project could integrate SA meta-alert adjudication in their CI.

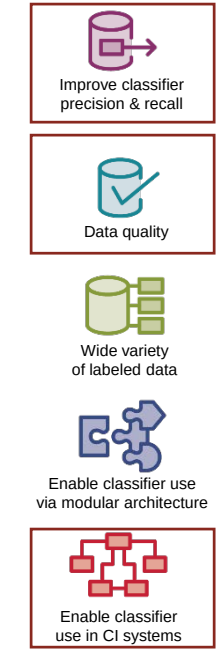
Goal: Enable **practical** automated classification for more secure software and lower cost/effort.

# 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 the time to manually adjudicate meta-alerts for the same level of security.

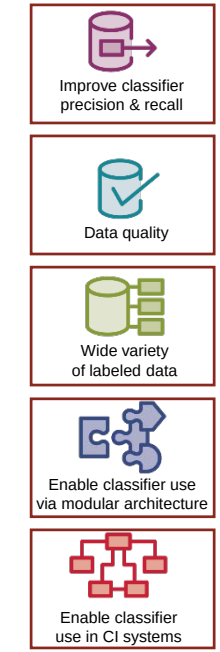
**Envisioned classifier-use scenario in Authorization to Operate (ATO)**

- DoD Program PMO must provide evidence on how software risks managed
  - PMO needs ATO by Authorizing Official
  - How to do this for CI/CD systems is being developed and tested, 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.



# Project Impacts Time Frame

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

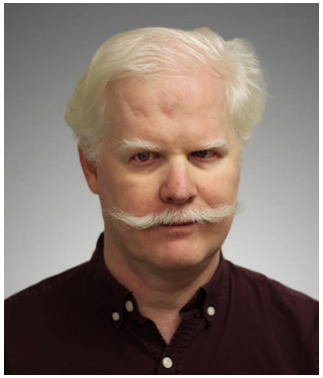



** Version of SCALe used in SCAIFE System implementation*

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

# FY20 Project Team





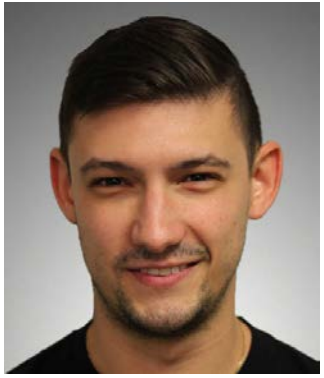

Dr. Lori Flynn

Ebonie McNeil

David Svoboda

Matt Sisk






Hasan Yasar

Joseph Yankel

Shane Ficorilli

David Shepard

# For More Information

## Contact Us

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

**Phone**: 412/268.5800 | 888.201.4479

**Web**: www.sei.cmu.edu

**Email**: info@sei.cmu.edu