



# Using Machine Learning to Improve Security Analysis of Source Code

Mark Sherman  
Sept 23, 2020  
Cylab Partners Conference

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213

Copyright 2020 Carnegie Mellon University.

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

**NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.**

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM20-0770

# Research Projects at SEI/ CERT

**Source Code as Natural Language**  
**Combining Source Code Analyzers**  
**Automated Source Code Repair**

# Research Teams

## PIs

Lori Flynn

Will Klieber

Carson Sestili

## Team members

Jennifer Burns

Matt Churilla

Zachary Kurtz

Jiyeon Lee

Derek Leung

Guillermo Marce-  
Santurio

Ruben Martins (SCS)

Mike McCall

## Team members (cont)

Ebonie McNeil

Richard Qin

Will Snavely

Ryan Steele

Robert Stoddard

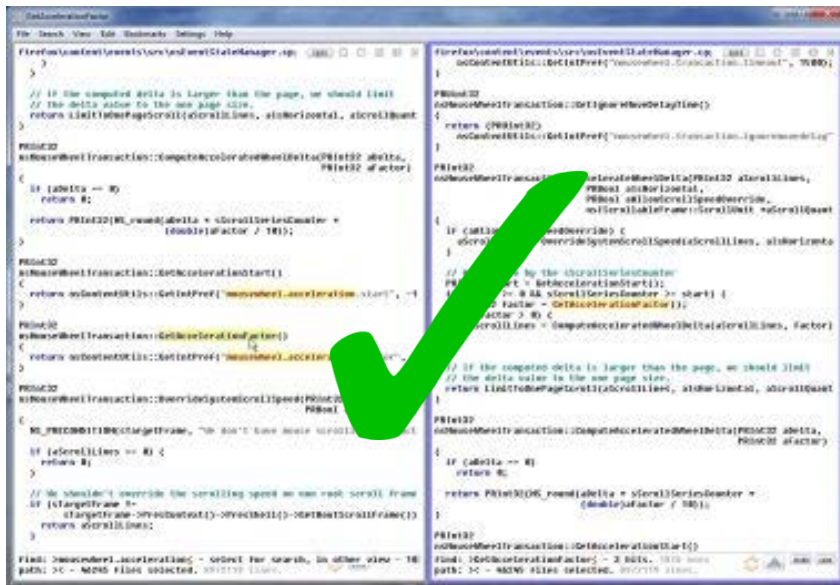
David Svoboda

Nathan VanHoudnos

Joseph Yankel

David Zubrow

# Finding Code Defects – ML that Considers Source Code as Natural Language



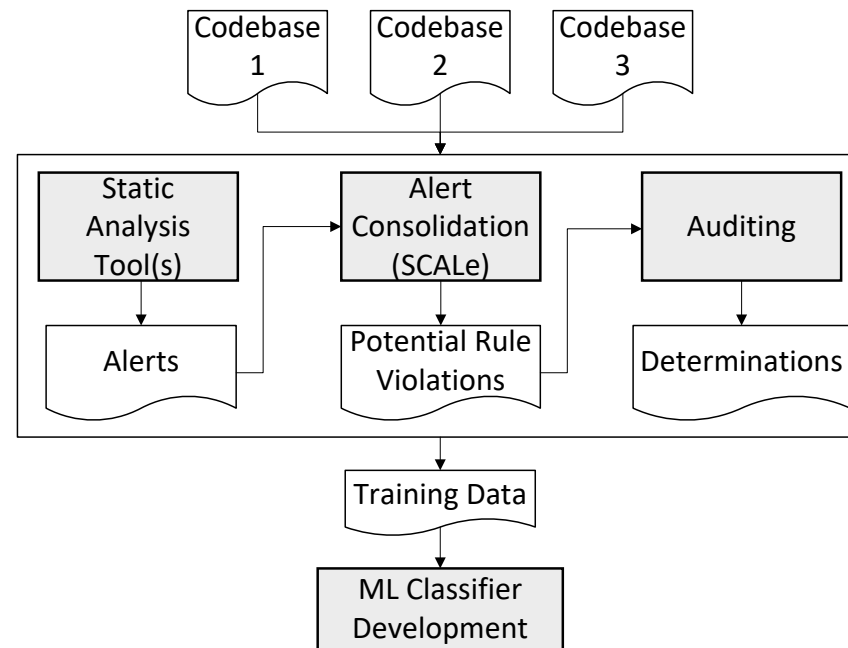
The image shows a screenshot of a code editor with two panes. The left pane contains C++ code with comments and function calls. The right pane contains more C++ code, including a function definition. A large green checkmark is overlaid on the code, indicating a successful analysis or finding.

## Analyze Source Code for Insecure Coding

- Supplements Compiler-style Checking
- Treats Programs Like Natural Language

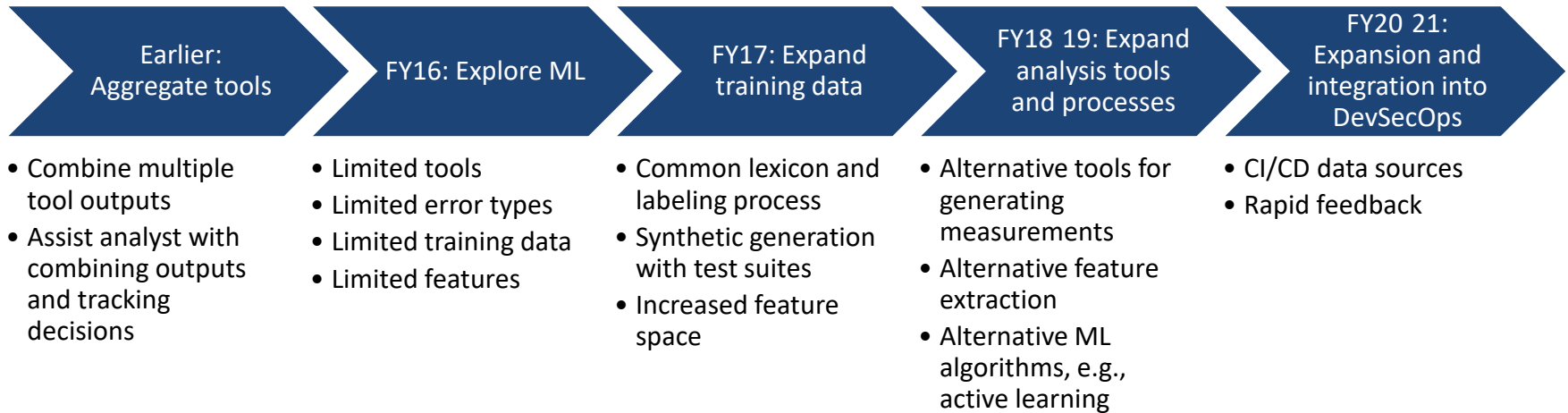
Sources: Carson D. Sestili, William S. Snavey, Nathan M. VanHoudnos, Towards security defect prediction with AI, Sep 12, 2018, [read the article here](#).

# Finding Code Defects Using Machine Learning



[Read the L. Flynn publications at SEI Digital Library here.](#)

# Introduction and Integration of Machine Learning



# Automated Code Repair (ACR) Tool as a Black Box

**Input:** Buildable codebase

**Output:** Repaired source code that is still human-readable and maintainable.

We currently support C code. Support for C++ can likely be added without too much difficulty.

## ACR Tool



[Read Will Klieber, Automated Code Repair to Ensure Memory Safety, Feb 24, 2020, here.](#)



# Why repair of source code instead of as a compiler pass?

## Repair of source code

Easily audited (if desired).

Repairs can easily be tweaked to improve performance, if necessary.

Changes to source code are frequent and easily handled.

Okay to do slow, heavy-weight static analysis; produces a persistent artifact.

## Repair as a compiler pass

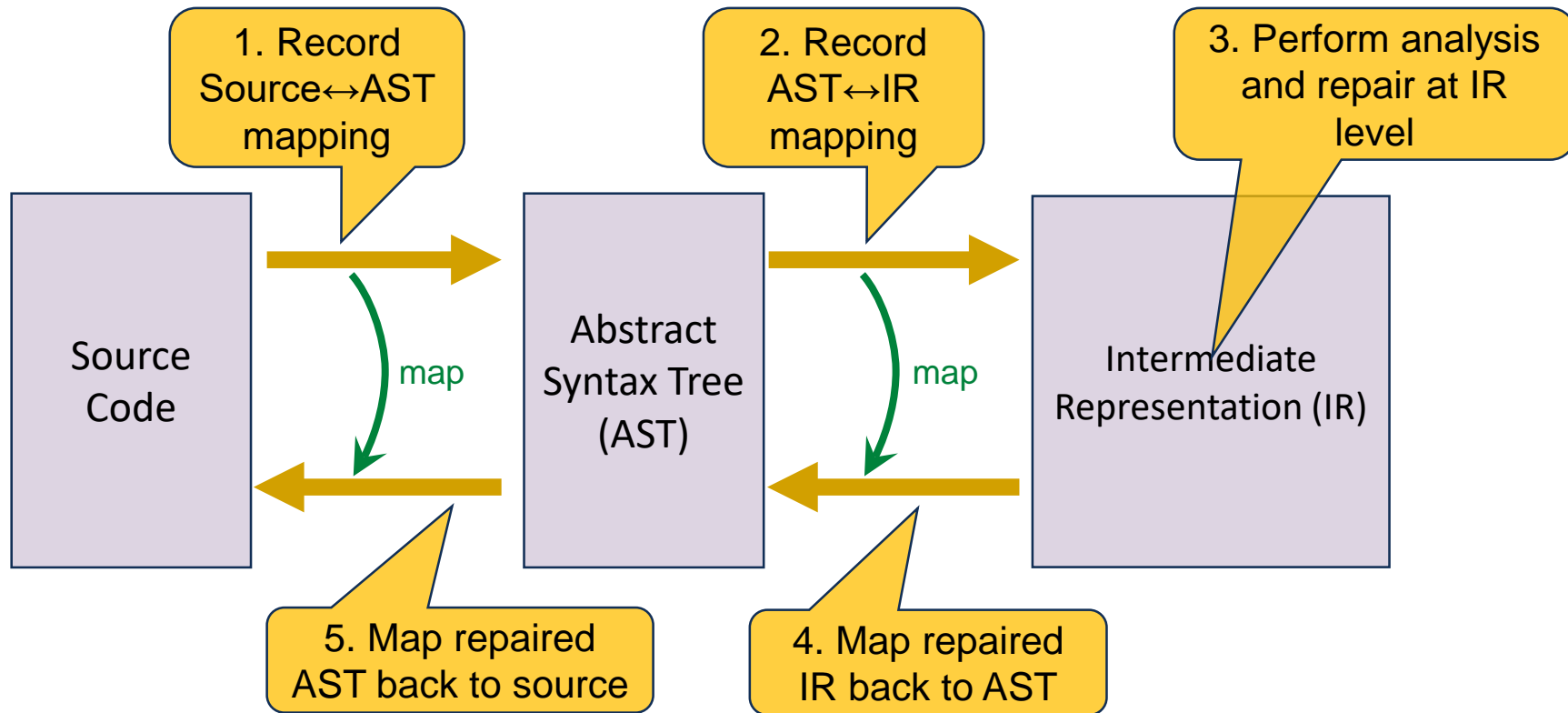
Must trust the tool.

Difficult to remediate performance issues caused by repair.

Changes to the build process may be more difficult and error-prone.

Slowing down every test build is not okay.

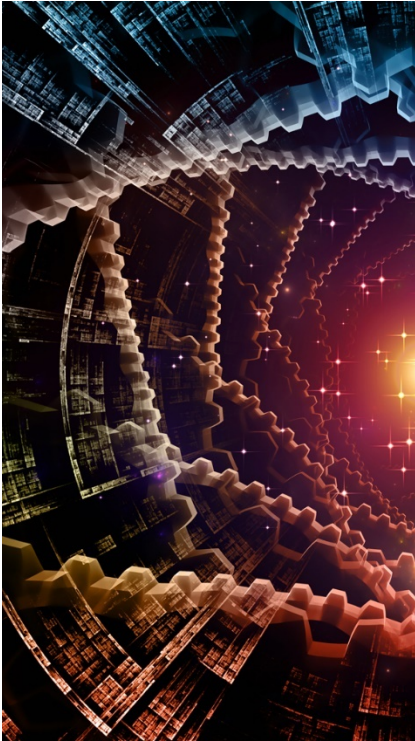
# Source Code Repair Pipeline



# Challenges

- In translating repairs from AST to source, the C preprocessor is main difficulty.
  - Repairs to macro uses
  - Repairs to `#included` code
  - Conditional-compilation directives (`#ifdef`, `#endif`, etc.) inside expressions
- Considerations of whitespace
- The C preprocessor can conditionally include or exclude pieces of code depending on the configuration chosen at compile time.
  - We repair configurations separately and then merge the results such that the final repaired code is correct under all desired configurations.

# Ways to Engage with Us



- Download [software and tools](#)
- Explore [research and capabilities](#)
- Participate in [education](#) offerings
- Attend an [event](#)
- Search the [digital library](#)
- Read the [SEI Year in Review](#)
- [Collaborate](#) with the SEI on a new project

## Software Engineering Institute

Carnegie Mellon University  
4500 Fifth Avenue  
Pittsburgh, PA 15213-3890  
412-268-5800 - Phone  
888-201-4479 - Toll-Free  
412-268-5758 - Fax  
[info@sei.cmu.edu](mailto:info@sei.cmu.edu) - Email  
[www.sei.cmu.edu](http://www.sei.cmu.edu) - Web