

MATLAB EXPO 2019

Making Software Safe and Secure
with Team Collaboration

Static Analysis with Polyspace

Olivier Bouissou



Agenda

1. Why do you need Static Analysis?
2. Polyspace Static Analysis
3. Team Collaboration with Polyspace

1. Why do you need Static Analysis?



Martin



Eric



Bob



Martin is a software developer.
He writes code in C/C++.

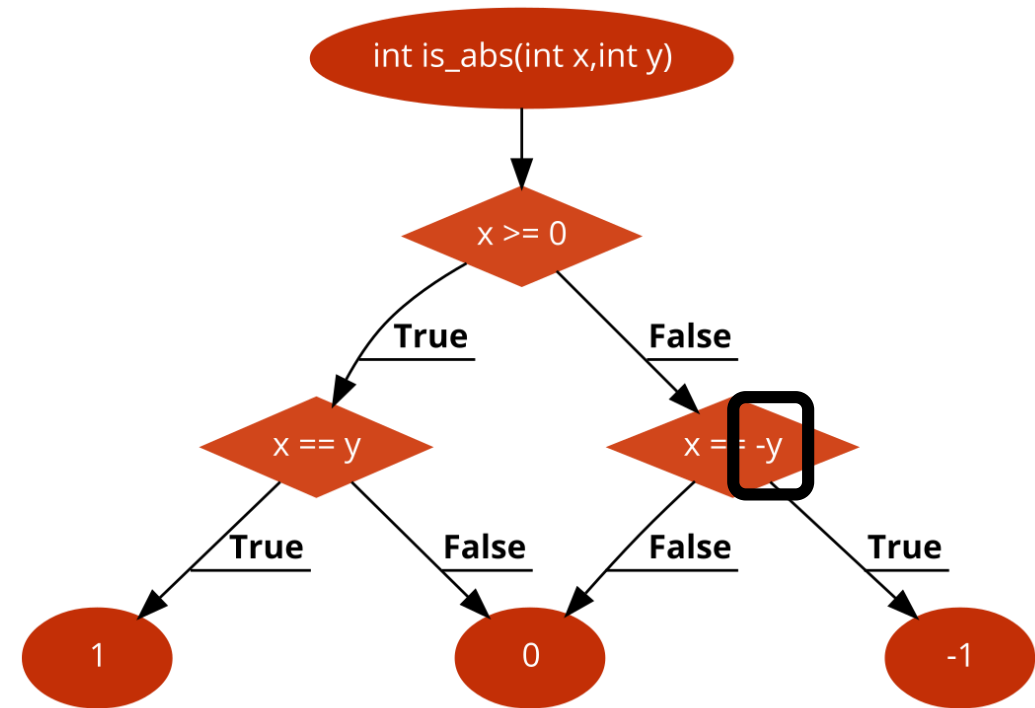
I'm already doing code reviews
and writing unit tests.
Why should I run a static analysis
tool?

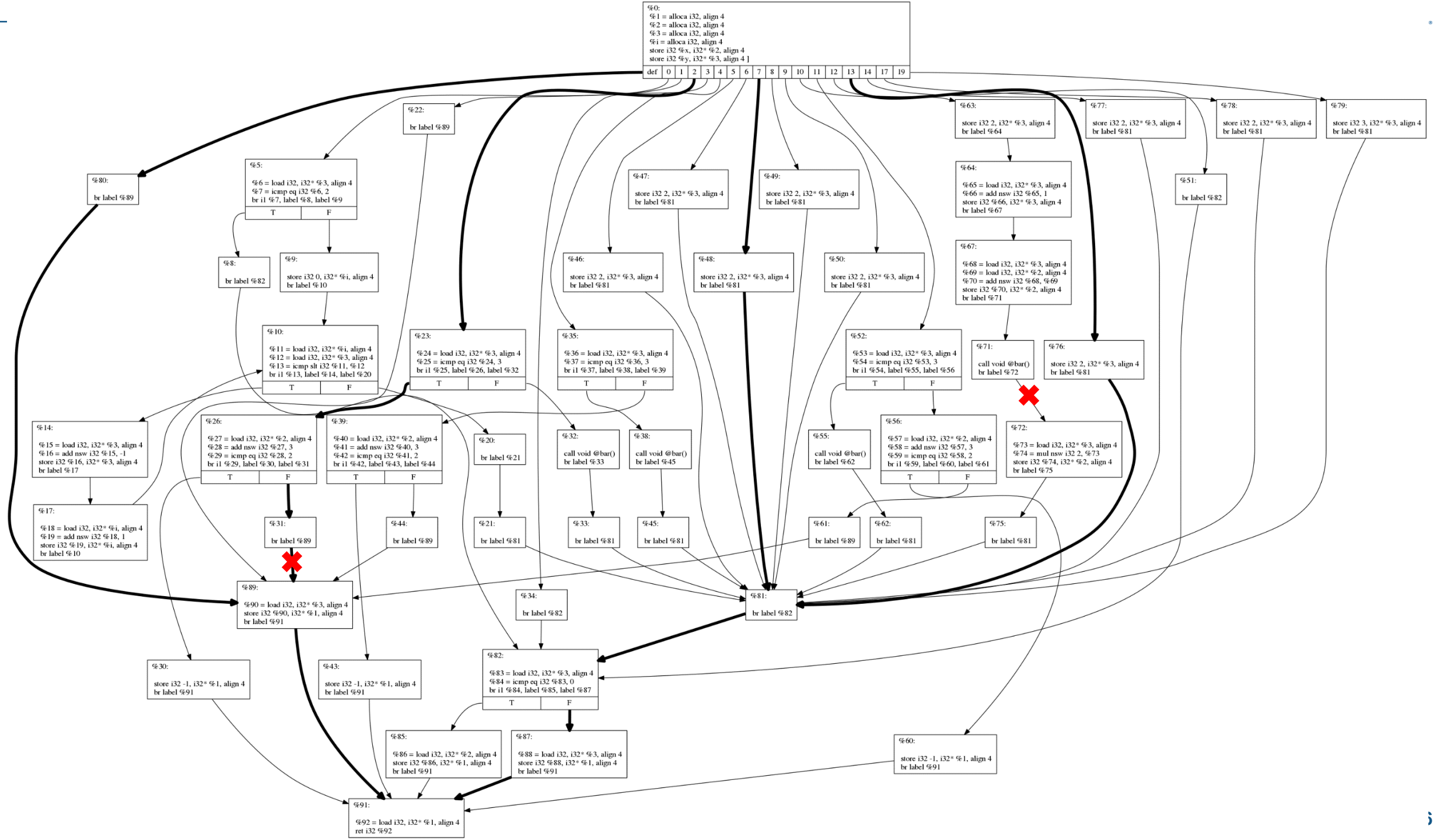
```
1
2 int is_abs(int x,int y) {
3     if (x >= 0) {
4         if (x == y)
5             return 1;
6     }
7     else {
8         if (x == -y)
9             return -1;
10    }
11    return 0;
12 }
```



Martin is a software developer.
He writes code in C/C++.

I'm already doing code reviews
and writing unit tests.
Why should I run a static analysis
tool?



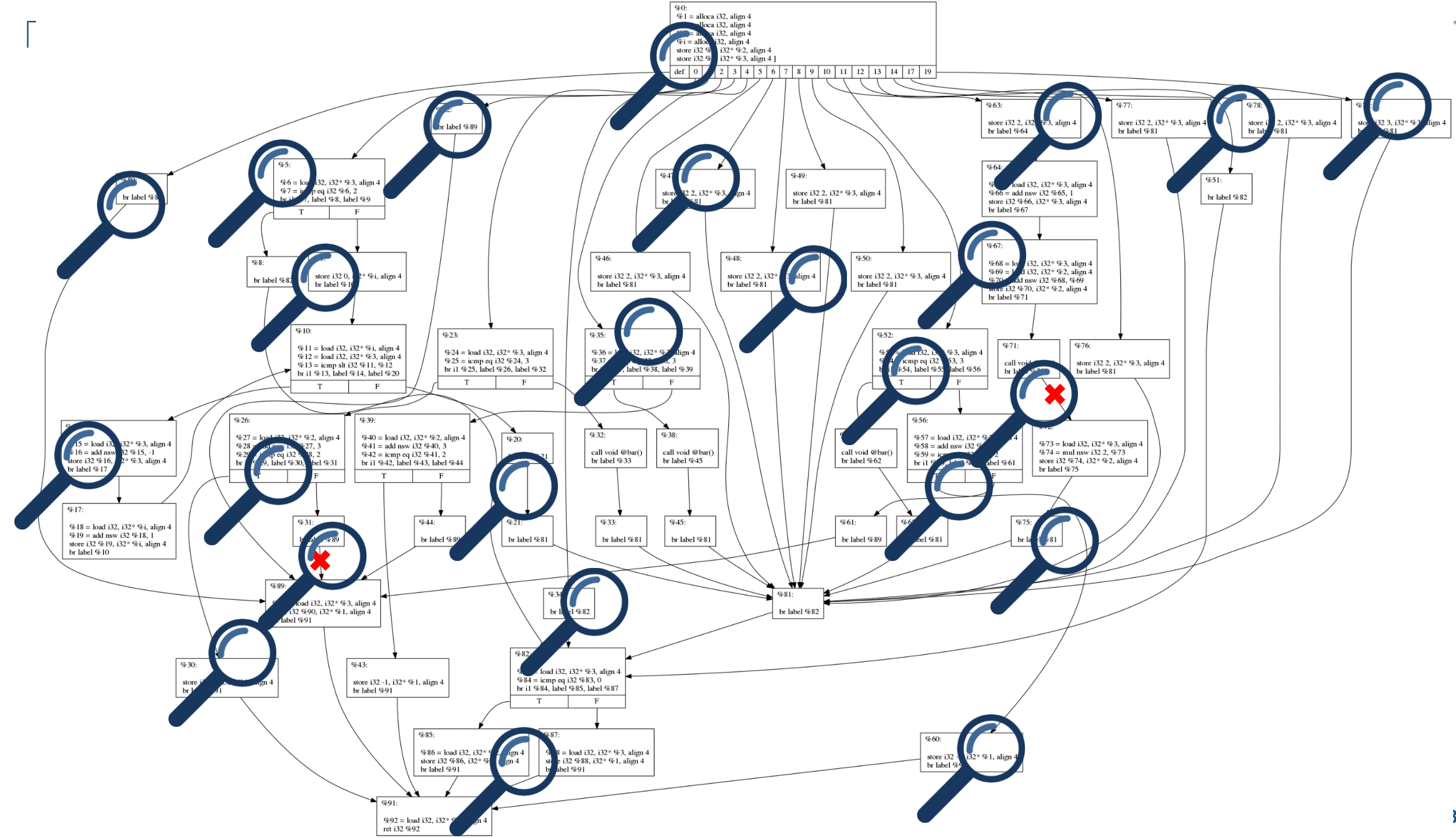


**“Program testing can be used to show the presence of bugs,
but never to show their absence”**

Edsger Dijkstra, Computer Science Pioneer

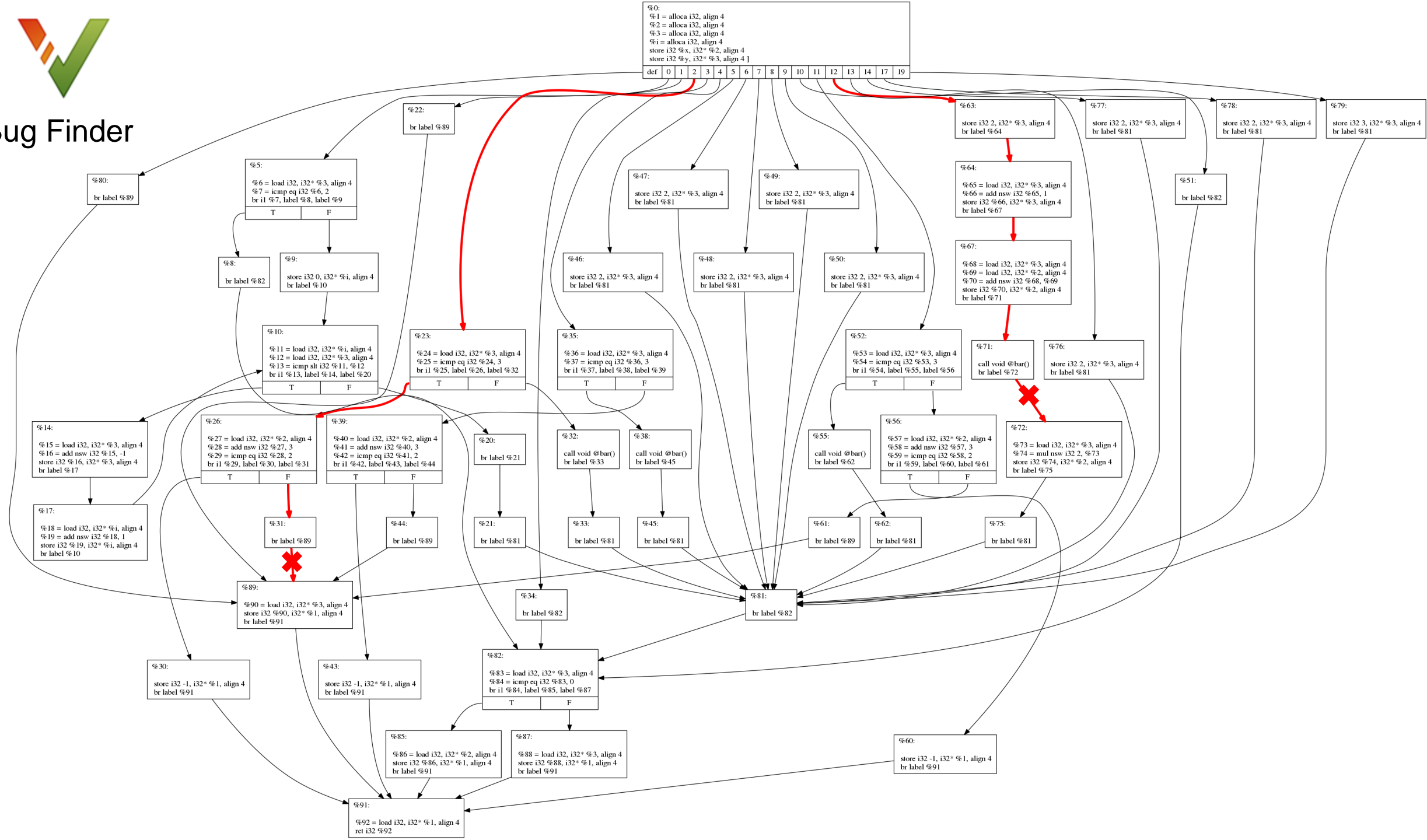
**“Given that we cannot really show there are no more errors
in the program, when do we stop testing?”**

Brent Hailpern, Head of Computer Science



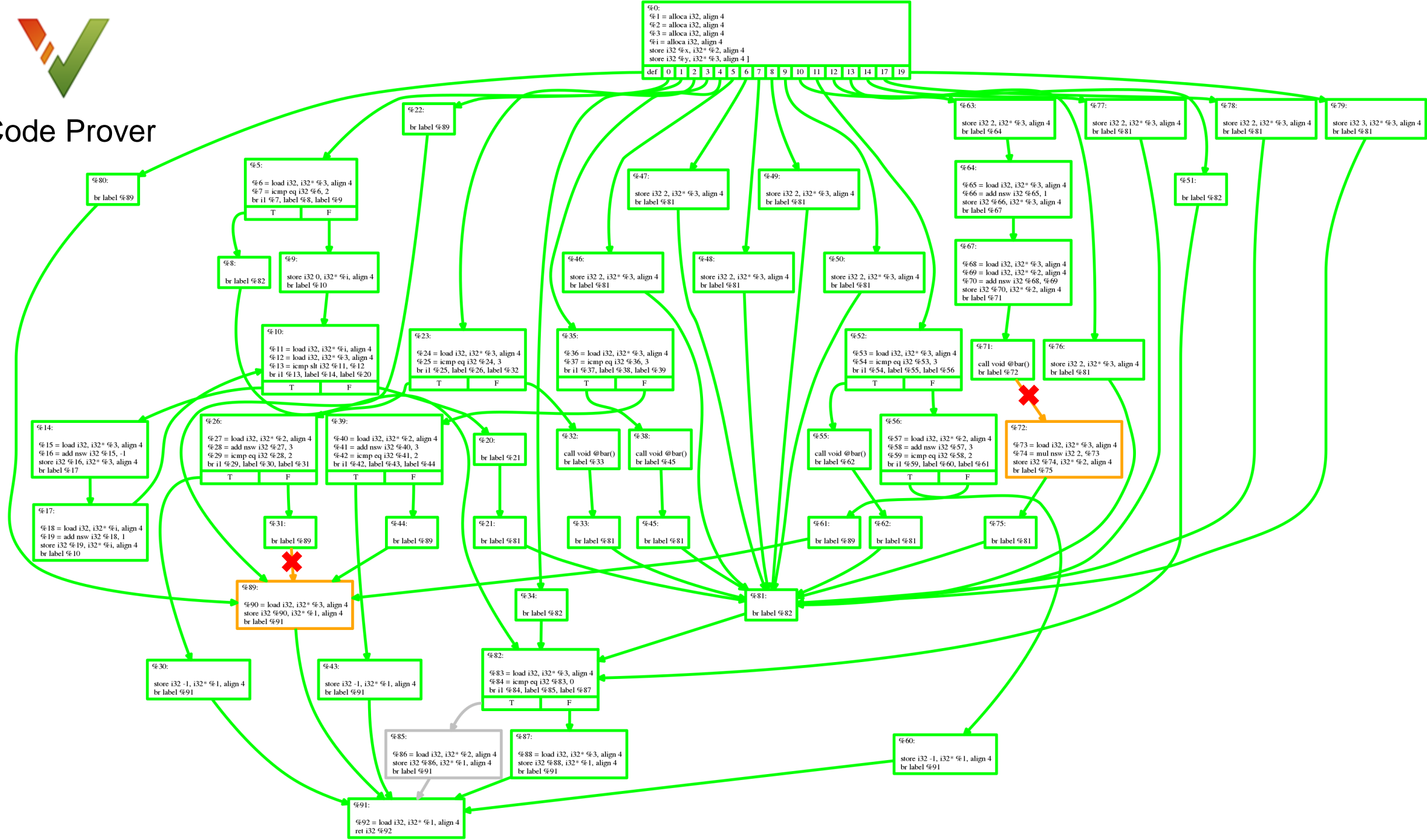


Bug Finder





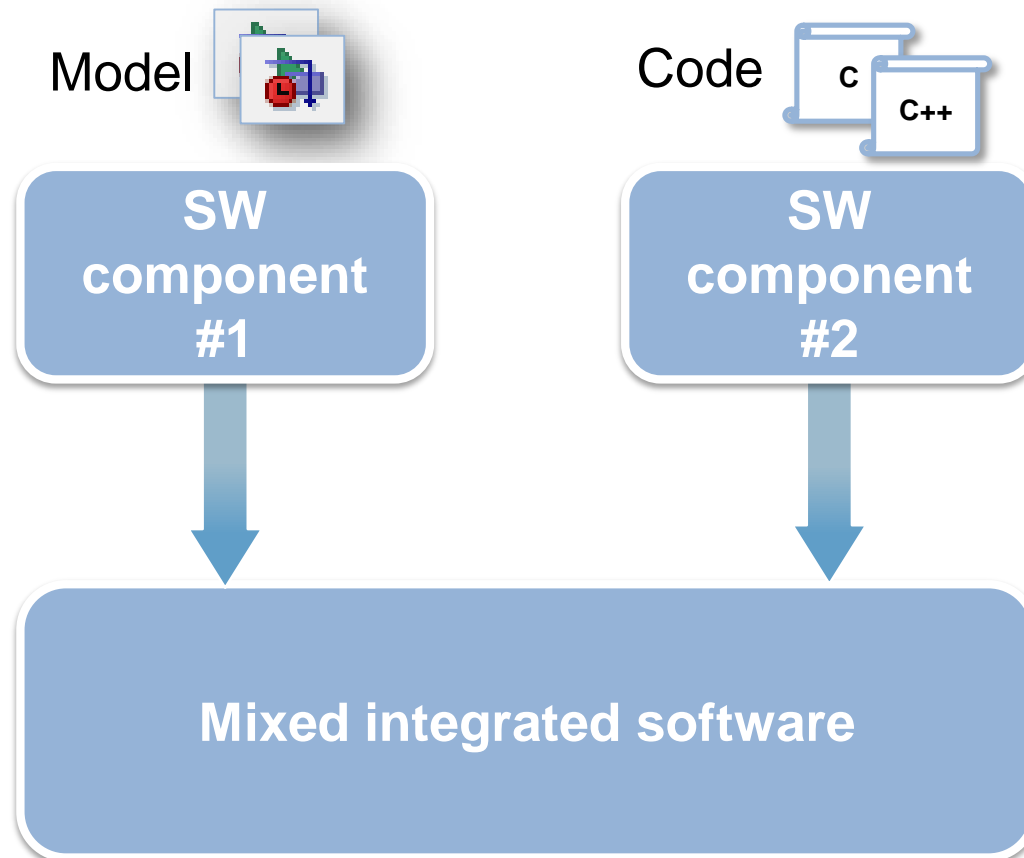
Code Prover





Eric is a Simulink and Embedded Coder user.
He is responsible for generating code from models.

I'm generating my code from Simulink,
and running V&V tools on the models.
Why should I check my software too?





Bob is a software developer.
He is writing software embedded in a pacemaker.

I'm working for a medical company.
Is static analysis useful for me?



Miracor Eliminates Run-Time Errors and Reduces Testing Time for Class III Medical Device Software



KOSTAL Asia R&D Center Receives ISO 26262 ASIL D Certification for Automotive Software



Alenia Aermacchi Develops Autopilot Software for DO-178B Level A Certification

2. Polyspace Static Analysis

For software written in C, C++, and Ada

Proving Absence of Critical Run-Time Errors

```
1 float where_are_errors_float(float input)
2 {
3     float x, y, k, l, limit = 1000.0f;
4
5     if (input < -limit || input > limit) return (-9999.0f);
6
7     k = input / 100.0f;
8     x = 2.0f;
9     y = k + 5.0f;
10
11     while (x < 10.0f)
12     {
13         x++;
14         y = y + 3.141592f;
15     }
16
17     if ((3.0*k + 100.0f) >= 71.0f)
18     {
19
20
21
22
23     return x;
24 }
```

x = x / (x - y);

- How many run-time errors are possible?
 1. Divide by zero
 2. Overflow
 3. Uninitialized variables

Proving Absence of Critical Run-Time Errors

```

1  float where_are_errors_float(float input)
2  {
3      float x, y, k, l, limit = 1000.0f;
4
5      if (input ≤ -limit || input ≥ limit) return (-9999.0f);
6
7      k = input / 100.0f;
8      x = 2.0f;
9      y = k + 5.0f;
10
11     while (x < 10.0f)
12     {
13         x++;
14         y = y + 3.141592f;
15     }
16
17     if ((3.0*k + 100.0f) ≥ 71.0f)
18     {
19         y++;
20         x = x / (x - y);
21     }
22
23     return x;
24 }

```

What Polyspace infers:

$$k \in [-10, 10]$$

$$y = k + 5$$

$$y = k + 5 + (x - 2) * 3.141592$$

$$x = 10$$

$$k \geq (71 - 100)/3 \longrightarrow y \geq 20.466$$

Proving Absence of Critical Run-Time Errors

```
1 float where_are_errors_float(float input)
2 {
3     float x, y, k, l, limit = 1000.0f;
4
5     if (input ≤ -limit || input ≥ limit) return (-9999.0f);
6
7     k = input / 100.0f;
8     x = 2.0f;
9     y = k + 5.0f;
10
11     while (x < 10.0f)
12     {
13         x++;
14         y = y + 3.141592f;
15     }
16
17     if ((3.0*k + 100.0f) ≥ 71.0f)
18     {
19         y++;
20         x = x / (x - y);
21     }
22
23     return x;
24 }
```

Proven mathematically by
Polyspace that run-time error
will not occur

✓ Division by zero ?

Float division by zero does not occur
operator / on type float 32

left: 10.0

right: [-31.1328 .. -11.1327]

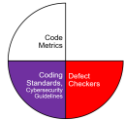
result: [-0.89826 .. -0.3212]

Mathematical proof via the Abstract Interpretation framework

- Very generic theory that ensures **soundness**, **automaticity** and **scalability**.
- **Soundness**
 - Captures all possible executions of the program
 - A green check proves that all executions are safe from Run Time Error
- **Automaticity**
 - No user intervention is required to guide the analysis
- **Scalability**
 - Technique scales up to large software with very complex dataflow

Polyspace Tools

Bug Finder

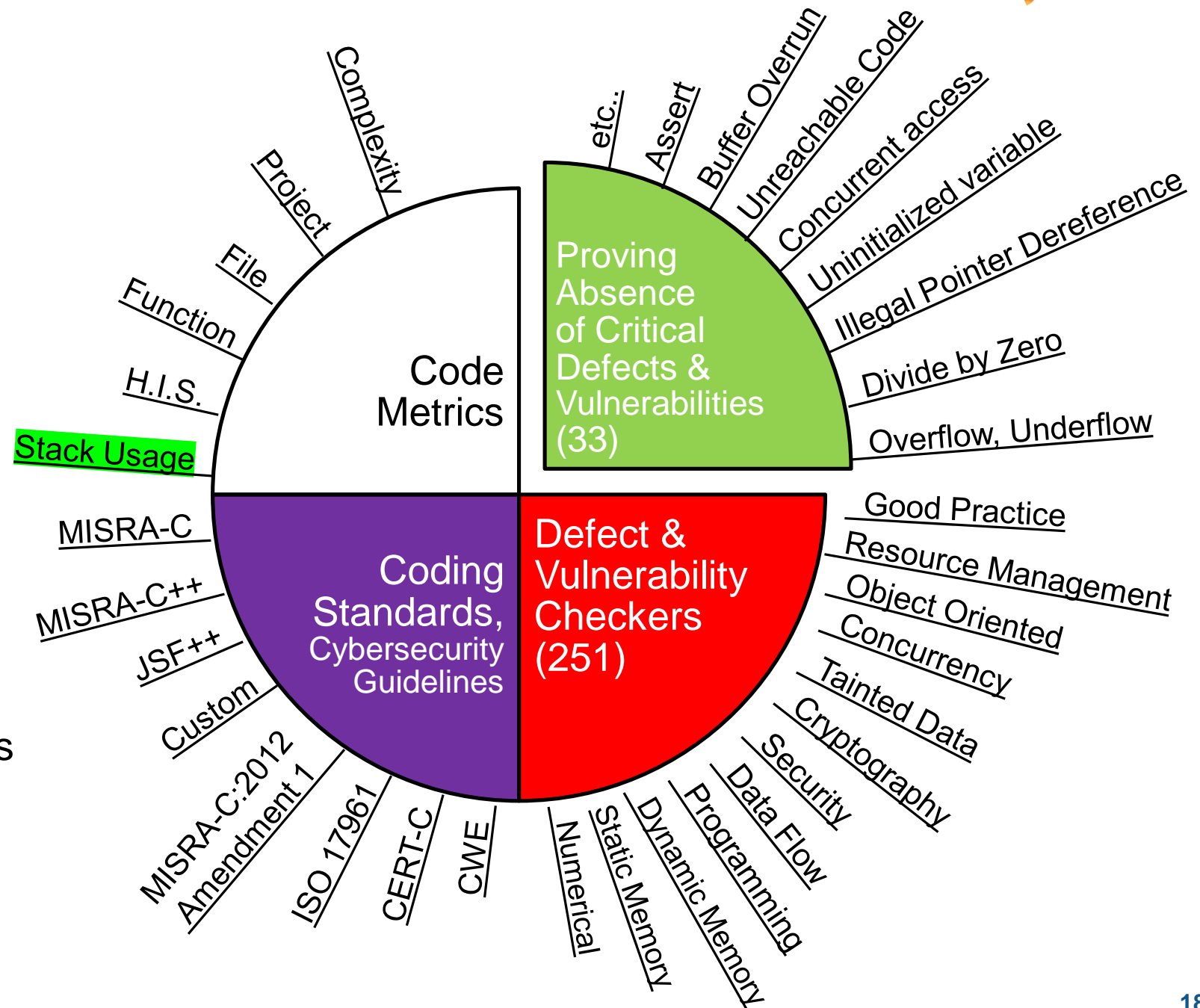


- Produce code metrics
- Check coding standards
- Find defects and vulnerabilities

Code Prover



- Proves code Safe and Secure
- 33 most critical run-time checks
- Helps getting certification credits (DO-178, ISO 26262, ...)



3. Team Collaboration with Polyspace



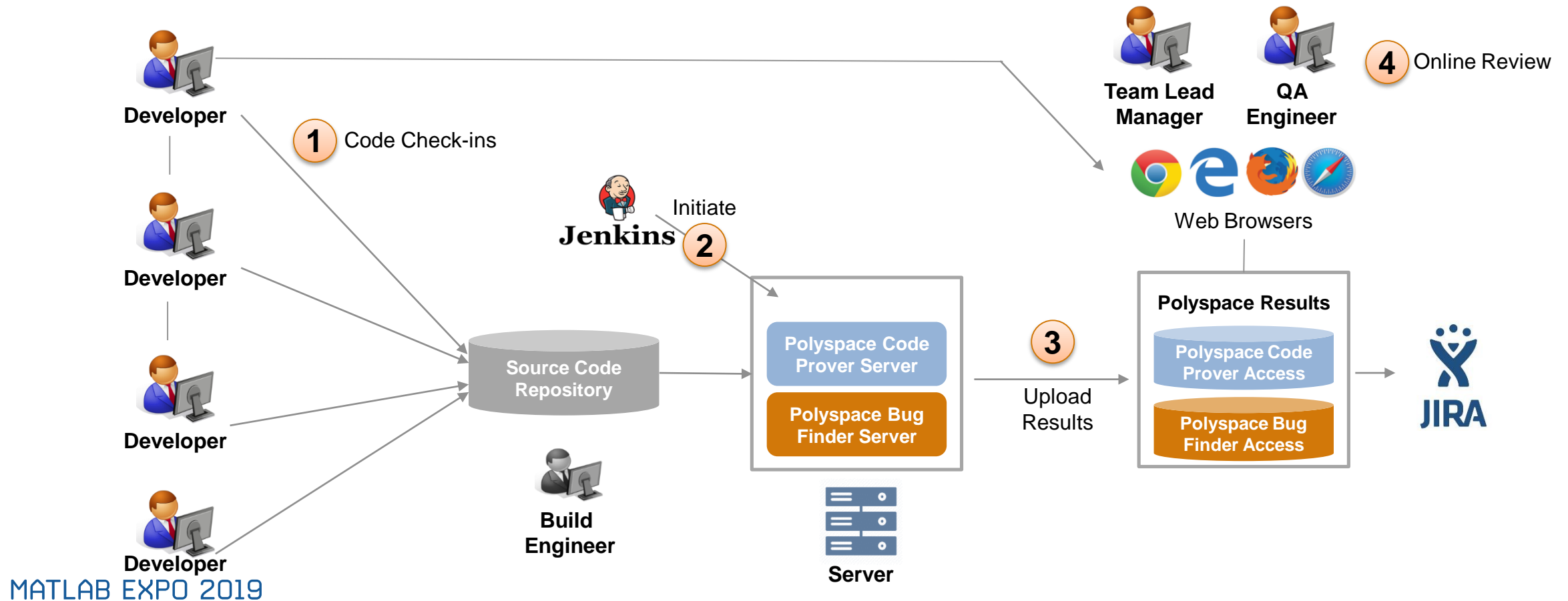
Quinn



Dara

Workflow with New Polyspace Products in R2019a

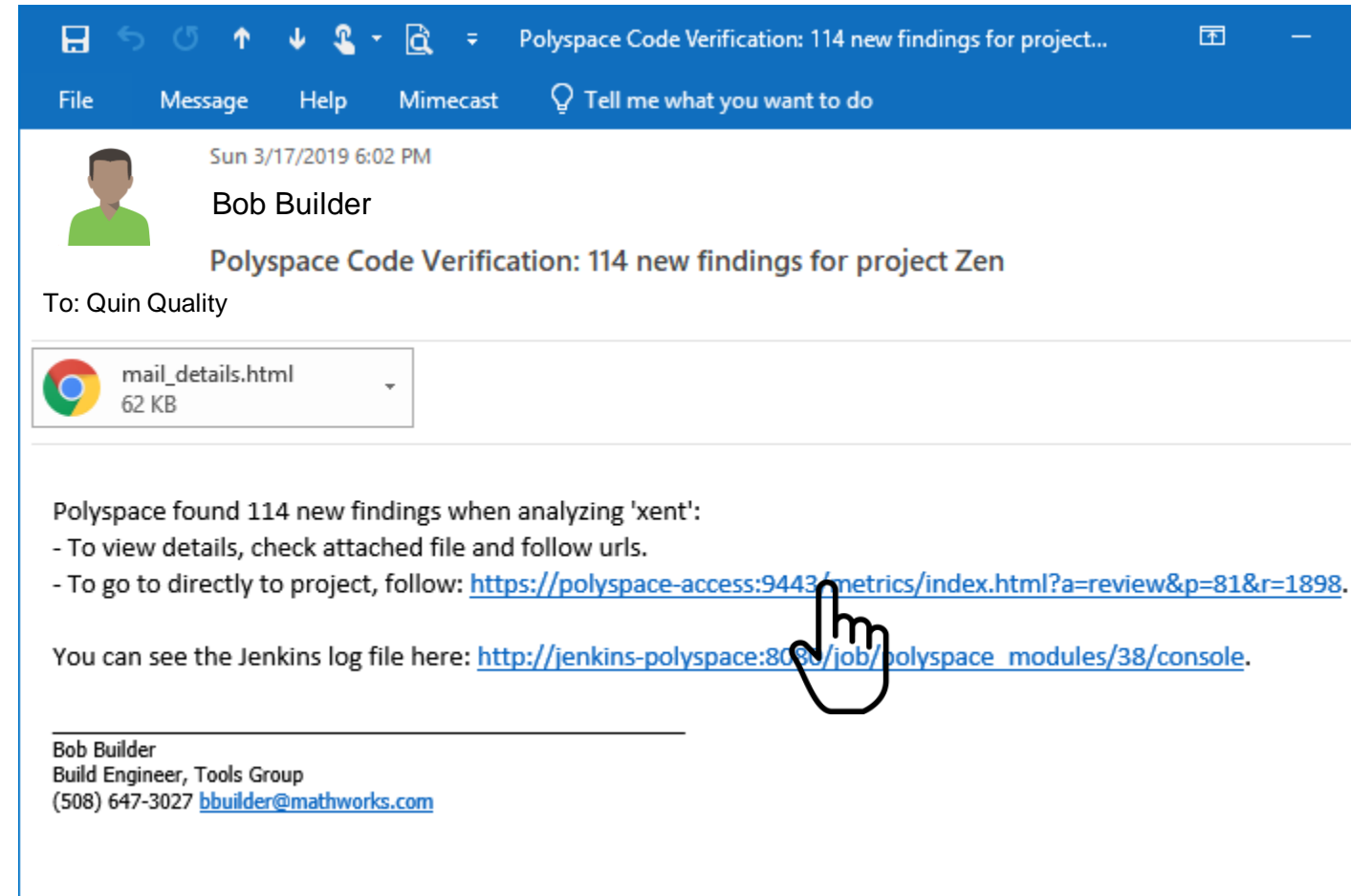
1. Developers check-in code into repository, Build Engineer has configured Jenkins to run Polyspace analysis
2. Jenkins initiates Polyspace analysis run on the server (periodically or at program milestones)
3. Once Polyspace analysis run concludes, results are uploaded to Polyspace Access
4. Team Lead/Manager, QA, Developers use web browser to review results, open Jira defects, monitor quality metrics





Quinn is a Quality Engineer
She is responsible for triaging software defects

- She received an email notification from last night's Jenkins initiated Polyspace analysis
- The email indicates several findings were found in her project
- She clicks on the link in the email to view the findings in Polyspace Access






Quinn is a Quality Engineer
She is responsible for triaging software defects

Sign In

localhost:9443/authn/signin?&title=/static/images/polyspace_title_logo.svg&continue=%2F



Sign in to your account

[Forgot password?](#)

Sign in

1984-2019 The MathWorks, Inc.



Dara is a software developer
She is responsible for writing code and fixing defects

- Dara has been assigned 2 defect tickets in Jira
- She opens the first JIRA ticket and clicks the Polyspace Access link

The screenshot shows a web browser window with the URL `https://jira-test-aws.mathworks.com/browse/UXVIZ-620`. The page header includes the MathWorks logo and navigation links for Jira, Dashboards, Projects, Issues, Boards, Structure, and MathWorks Applications. A 'Create' button is visible in the top right.

The main content area displays the 'Project Zen' sidebar on the left, which includes links for QE-IAT, Kanban board, Releases, Reports, Issues (selected), Components, and Add-ons.

The central panel shows the details of a Jira ticket titled 'Illegally dereferenced pointer' under the 'Visual Design' project. The ticket is assigned to 'Dara' and has a status of 'TO DO'. The ticket details include:

- Type: Bug
- Priority: Unset
- Affects Version/s: None
- Labels: None
- Geck Link: Create Geck
- Status: TO DO (View Workflow)
- Resolution: Unresolved
- Fix Version/s: None

The Description field contains the following text:

Error: pointer is outside its bounds
Found in C:\Polyspace\Proj_Zen\sources\example.c.
Go to Polyspace finding here: <http://localhost:9443/metrics/index.html?a=review&p=5&r=5&fid=1181>



Dara is a software developer
She is responsible for writing code and fixing defects

Polyspace

localhost:9443/metrics/index.html?a=review&p=6&r=7&fid=3949

REVIEW

Dashboard Run-time Checks Defects Coding Standards Code Metrics Global Variables

To Do In Progress Done

Show only Comment, filename, etc. Filter out Comment, filename, etc. Layout Open in Desktop

Showing: 1 / 1260 Finding ID

PROJECT EXPLORER

- public
 - Proj_Zen
 - Proj_Zen (Code Prover)
 - Test_Area

PROJECT DETAILS

Project

Name Proj_Zen (Code Prover)

Language C

FILE EXPLORER

- C:\Polyspace\Proj_Zen\sources

SUPPORT REPORT

Results List

Family	ID	Type	Group	Check
● *	3949	Red Check	Static memory	Illegally dereferenced p...

Result Details Contextual Help

example.c / Pointer_Arithmetic()

Status To Investigate

Severity High

Assigned to Type username or...

Track issue UXVIZ-623

Critical run-time error, finds investigation.

Source Code

example.c main.c

```
108 p++;
109 }
110
111 /* Set array based on bus status and oil pressure */
112 if (get_bus_status() >= 0) {
113     if (get_oil_pressure() >= 0) {
114         *p = 5;
115     } else {
116         *p = 0;
117     }
118 }
119
120 i = get_oil_pressure();
121
122 if (i >= 0) {*(p - i) = 10;}
123
124 if ((0 < i) && (i <= 100)) {
125     p = p - i;
126     *p = 5;
127 }
128
129
130
131
```


4. Summary

Summary

- Use Polyspace to achieve high quality software with reduced testing effort
 - Prove that your code will not cause safety hazards or security issues
- Polyspace fits software development workflows
 - Jenkins for build automation and Jira for bug tracking
- Support team-based collaboration
 - Results published for web browser based review by developers and quality engineers
 - Dashboards to show quality metrics for project and safety managers