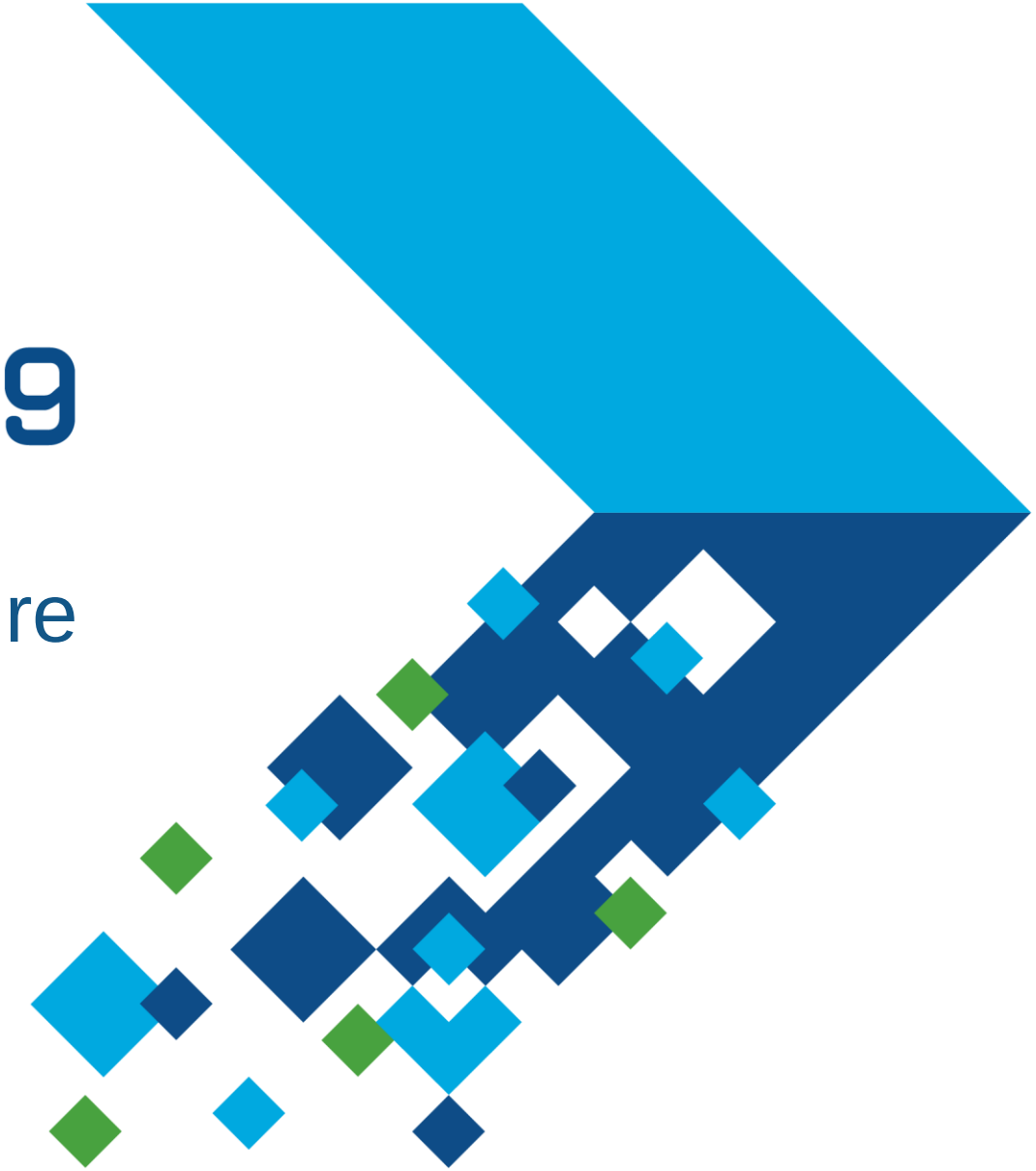
MATLAB EXPO 2019

Making Software Safe and Secure with Team Collaboration

Vaishnavi H R
Application Engineer,Mathworks



Agenda

1. Making Software Safe and Secure

2. Polyspace Static Analysis

3. Team Collaboration with Polyspace

1. Making Software Safe and Secure

“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

Dijstra, “Notes on Structured Programming” (1972)
Hailern, Santhanam, “Software Debugging, Testing, and Verification”, IBM Systems Journal, (2002)

Using Static Analysis to Make Software Safe and Secure

- Find bugs without code execution
  - Code analyzed without running tests
  - Identify bugs and coding rule violations for MISRA, AUTOSAR, CERT
- Prove absence of critical run-time errors
  - Identify code that will never experience errors regardless of run-time conditions
- Complements dynamic testing
  - Used together, you can find more bugs for higher quality code

main.cpp

```
20
21  static bool table_loop(void)
22  {
23      int j = 4;
24
25      // Table of basic element
26      Base* array[] = { new SAnalogic, new Sensor, new Sensor, new SAnalogic };
27
28      for (int i = 4; i >= 0; i--, j--) {
29          array[i-1]->Draw();
30
31          // Error for the 2 last elements: this cast is similar to static_cast
32          // the TypeInfo function only define in SAnalogic
33          if (i % 2)
34              ((SAnalogic*)(array[i-1]))->TypeInfo();
35          else
36              (dynamic_cast<SAnalogic*>(array[i-1]))->TypeInfo();
37      }
```

	Event	File	Scope
1	Iterating on loop	main.cpp	table_loop()
2	This-pointer of TypeInfo is null	main.cpp	table_loop()
3	● Non-terminating loop	main.cpp	table_loop()

● Non-terminating loop (?)

The loop is infinite or contains a run-time error.

Loop fails due to a run-time error (maximum number of iterations: 3).

When Software Safety and Security Matter

- Industries where safety and security matter
  - Automotive, Aerospace, Medical Device, Industrial Machinery

- Governed by functional safety and other standards
  - ISO 26262, DO-178, IEC 62304, IEC 61508
  - MISRA, CERT, AUTOSAR

- Static analysis provides certification credits
  - For standards such as ISO 26262 and DO-178

2. Polyspace Static Analysis

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

Proving Absence of Critical Run-Time Errors

```
float x, y;

...

x = x / (x – y);
```

- How many run-time errors are possible?
  1. Divide by zero
  2. Overflow
  3. Uninitialized variables
- How to test all floating point variable combinations?
- How do you prove that this code will not fail?

Proving Absence of Critical Run-Time Errors

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

✔ **Division by zero** ?

Float division by zero does not occur

operator / on type float 32

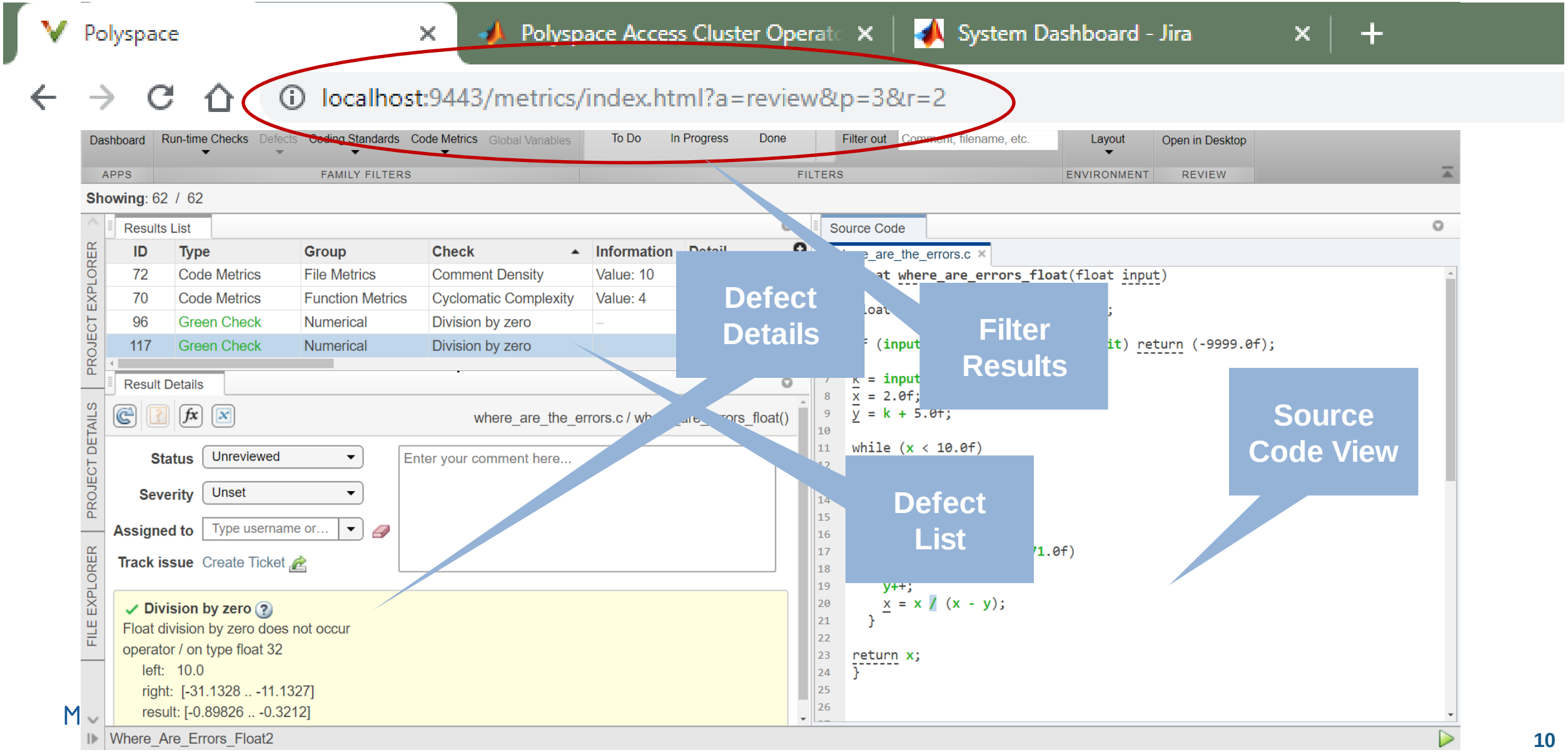
left: 10.0

right: [-31.1328 .. -11.1327]

result: [-0.89826 .. -0.3212]

```
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 }
```

Proving Absence of Critical Run-Time Errors with **Polyspace**



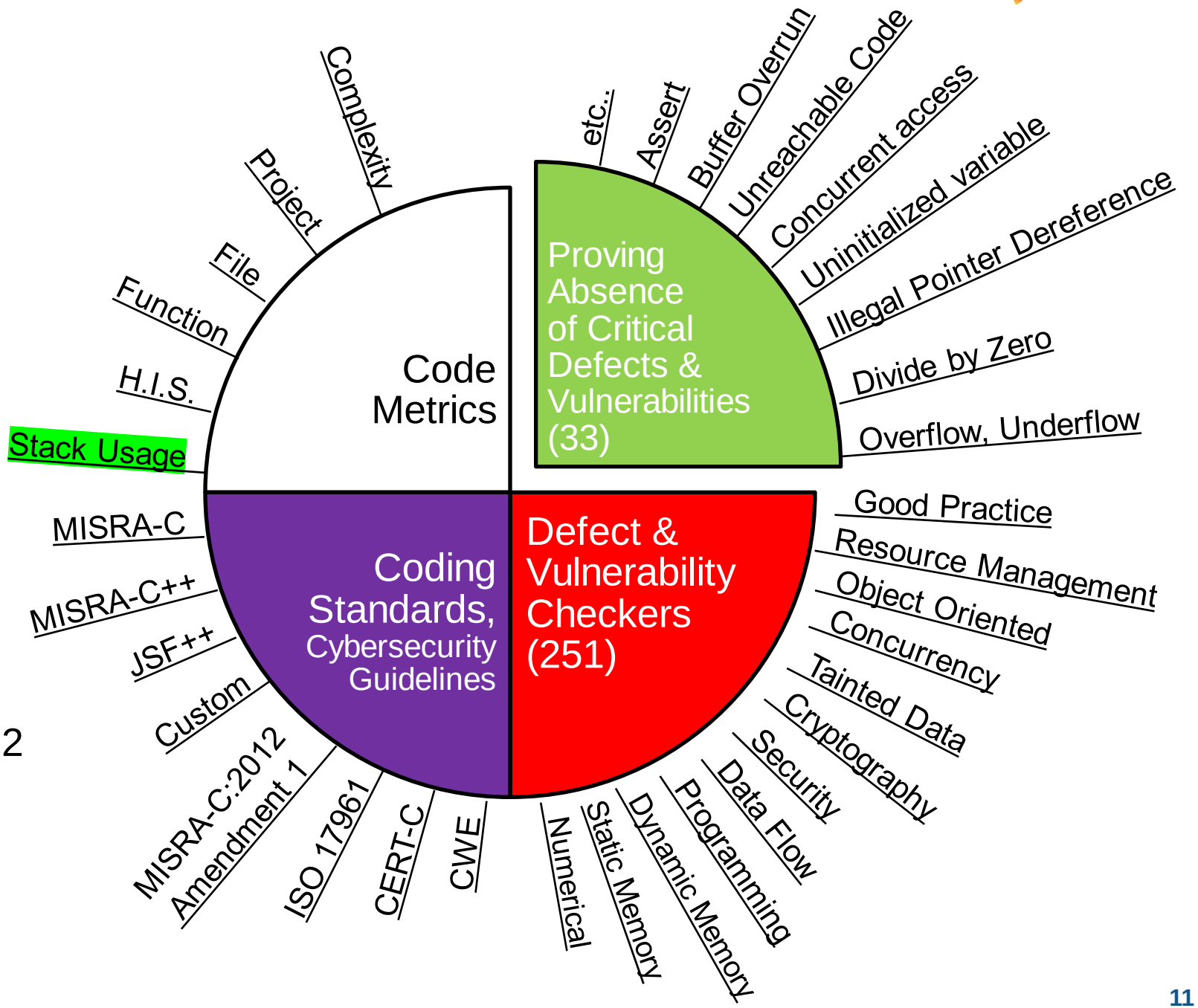
Polyspace Tools

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
- Supports DO-178 and ISO 26262



Polyspace Customer References

Electronic Steering Lock

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

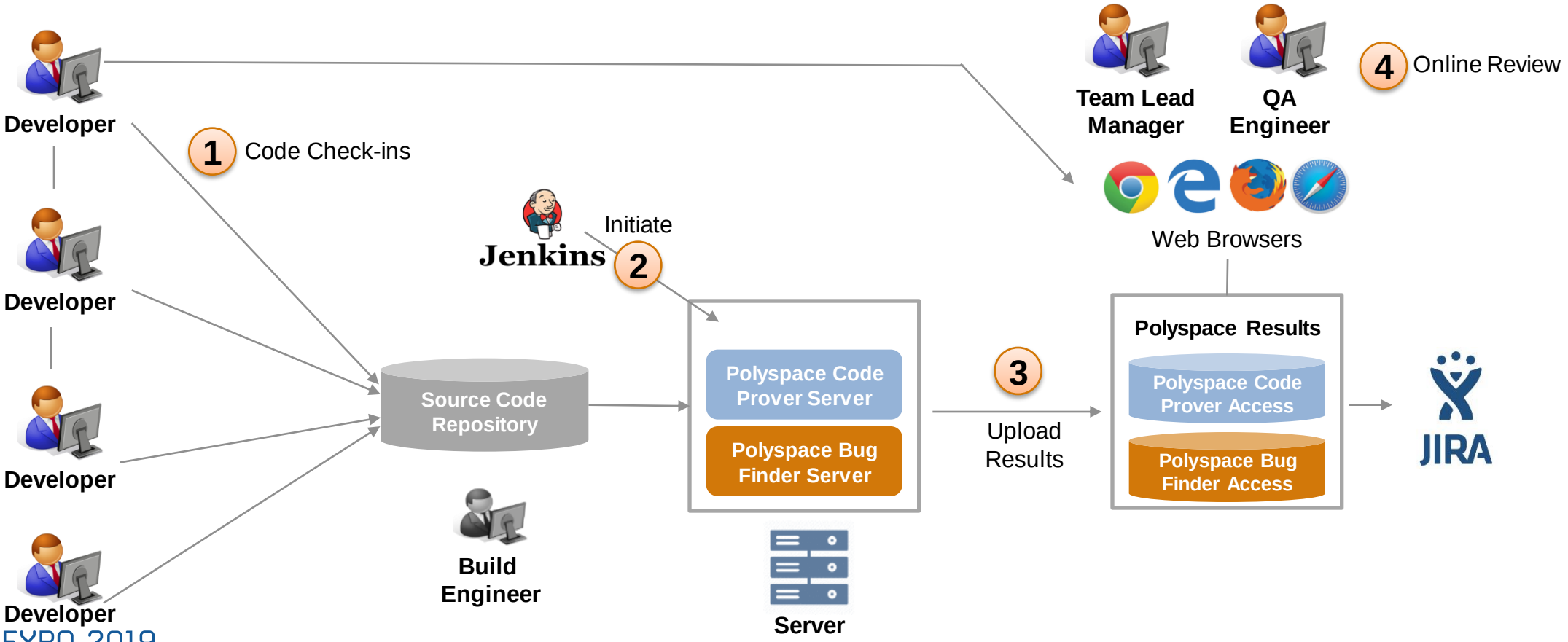
Alenia Aermacchi Develops Autopilot Software for DO-178B Level A Certification

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

3. Team Collaboration with Polyspace

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



Team Collaboration Story

Bob is the Build Engineer
He has configured Polyspace in a Jenkins CI workflow

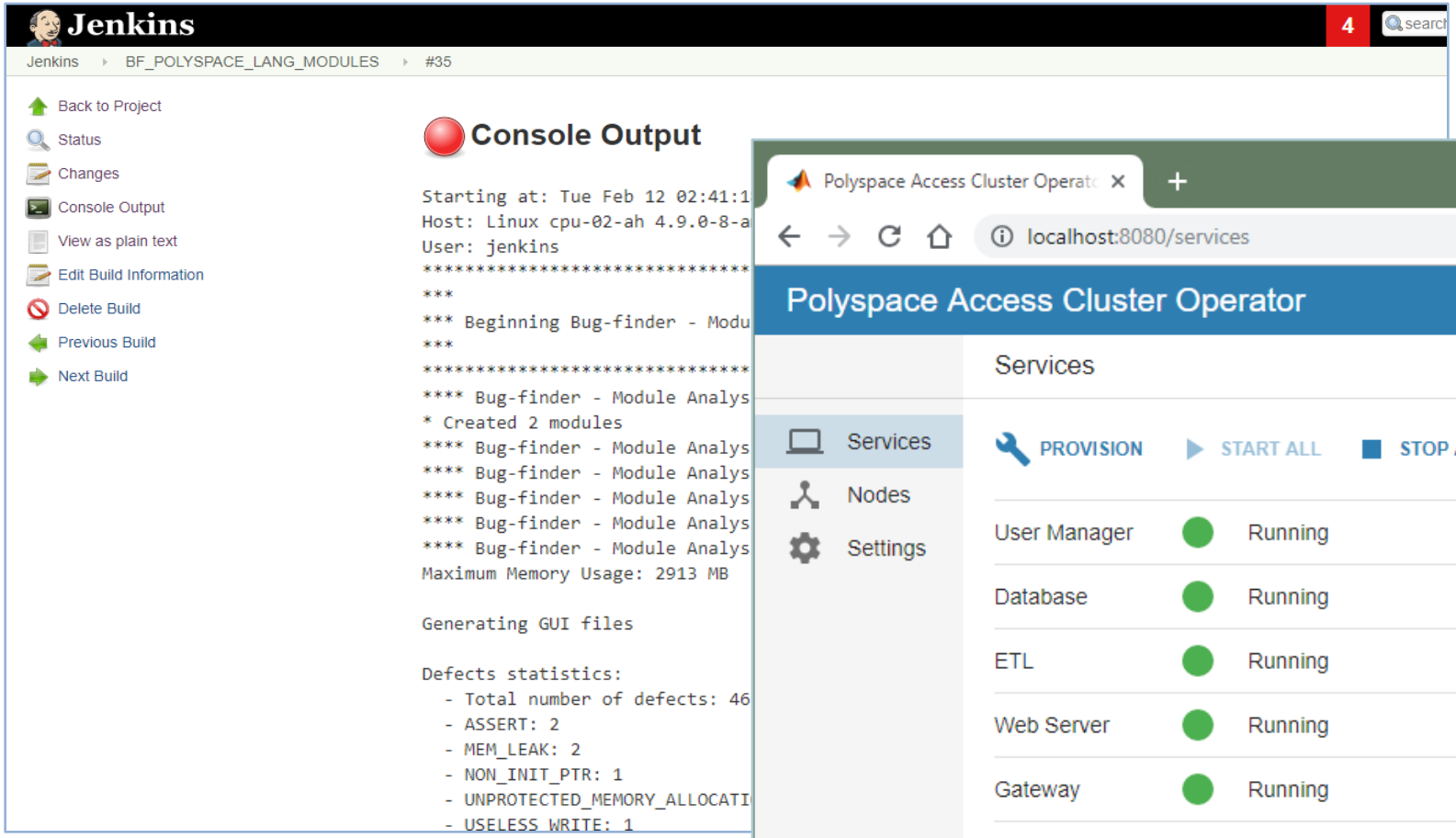
Quinn is a Quality Engineer
She is responsible for triaging software defects

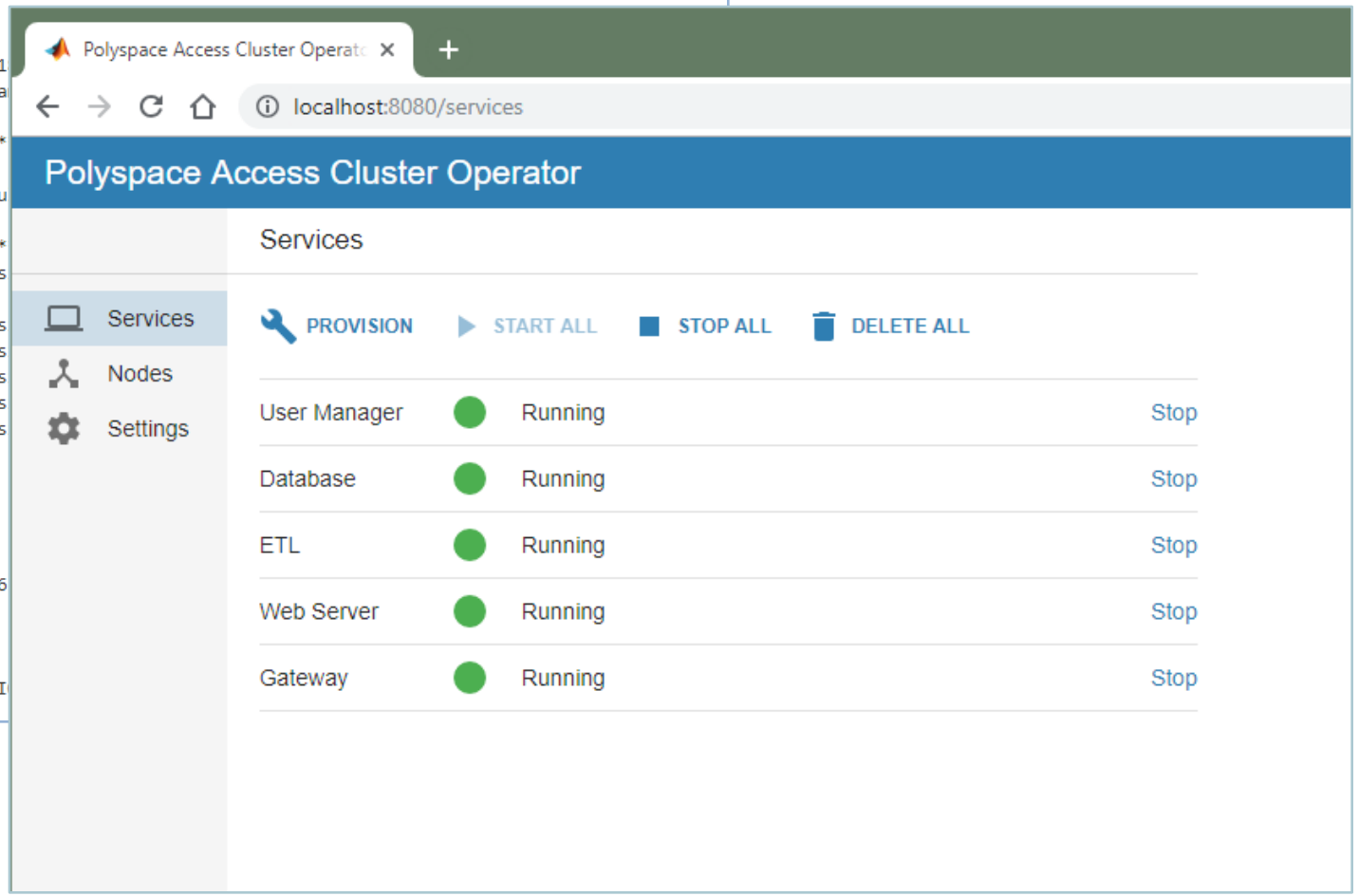
Dara is a software developer
She is responsible for writing code and fixing defects

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

Martin is a project manager
He is responsible for software quality of the project

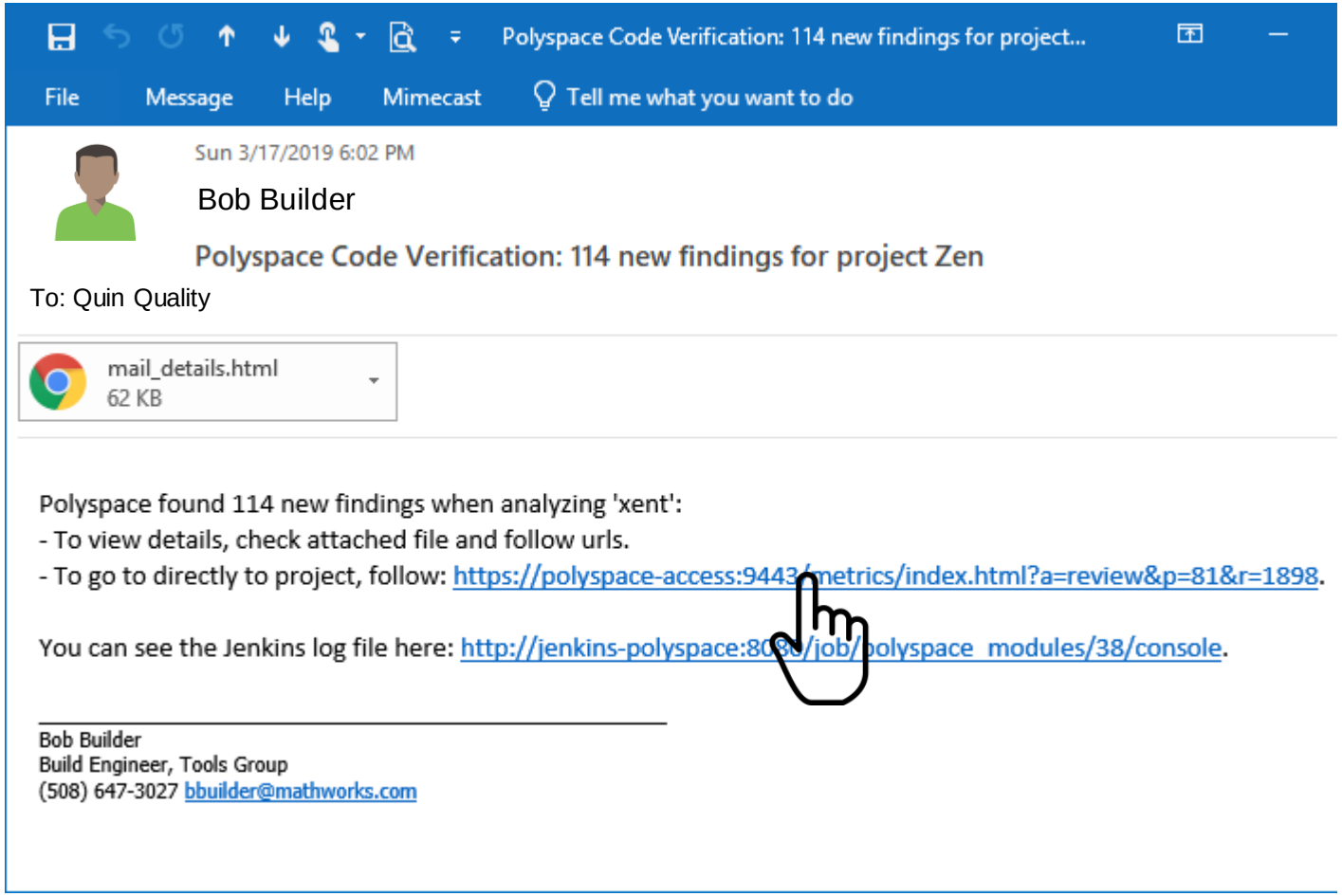
Bob is the Build Engineer
He has configured Polyspace in a Jenkins CI workflow





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 click 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

POLYSPACE®

Sign in to your account

Username

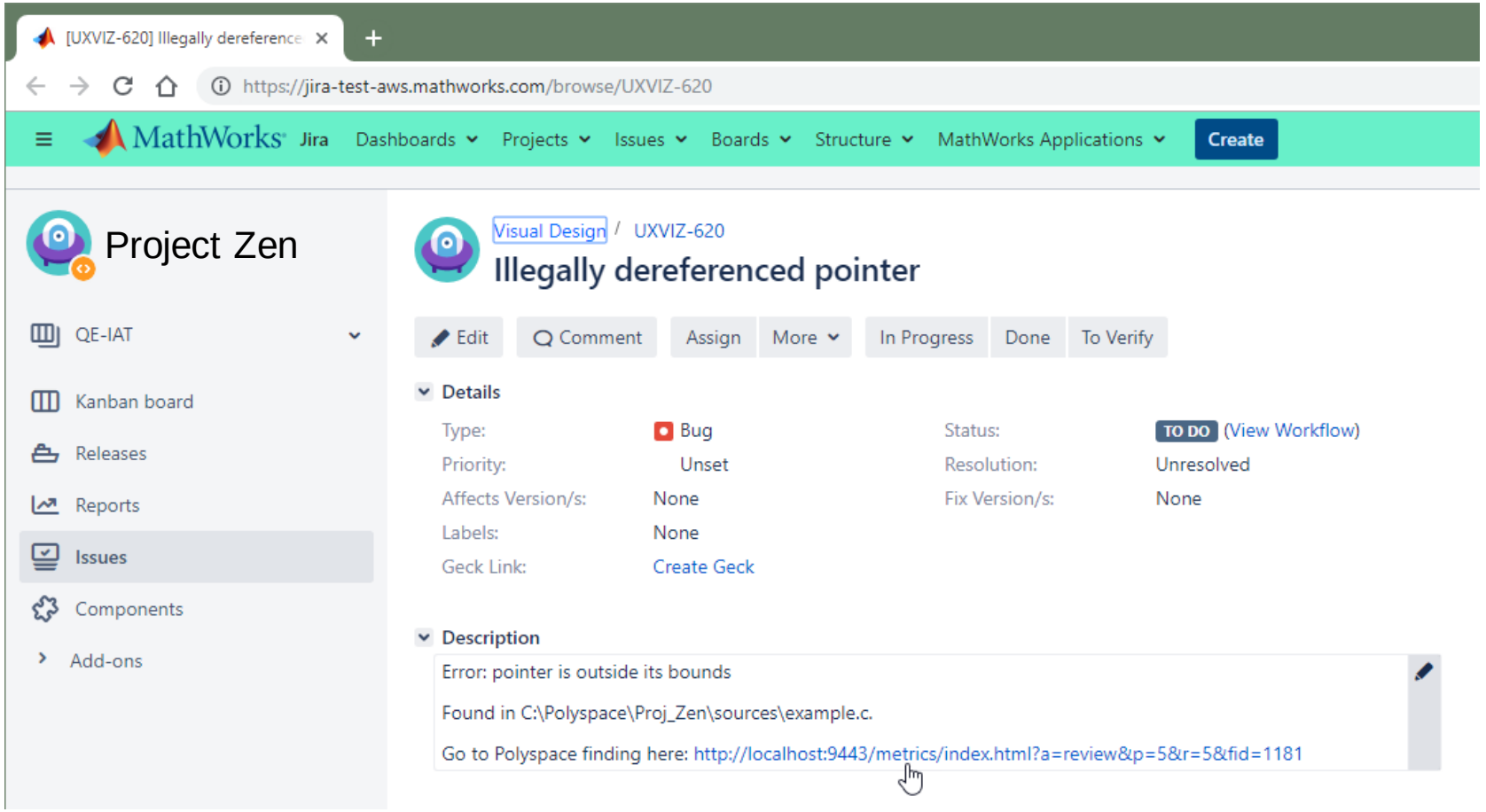
Password

Forgot password?

Sign in

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



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

[UXVIZ-623] Illegally dereference

https://jira-test-aws.mathworks.com/browse/UXVIZ-623

MathWorks Jira Dashboards Projects Issues Boards Structure MathWorks Applications Create Search

Visual Design

QE-IAT

Kanban board
Releases
Reports
Issues
Components
Add-ons

Visual Design / UXVIZ-623

Illegally dereferenced pointer

Edit Comment Assign More In Progress Done To Verify Export

Details

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

Description

Error: pointer is outside its bounds

Found in C:\Polyspace\Proj_Zen\sources\example.c.

Go to Polyspace finding here: http://localhost:9443/metrics/in Click to edit eview&p=6&r=7&fid=3949

Critical run-time error, needs investigation.

Attachments

Drop files to attach, or browse.

People

Assignee: Unassigned
Assign to me
Reporter: Jay Abraham
Watchers: 1 Stop watching this issue

Dates

Created: 2 hours ago
Updated: 2 hours ago

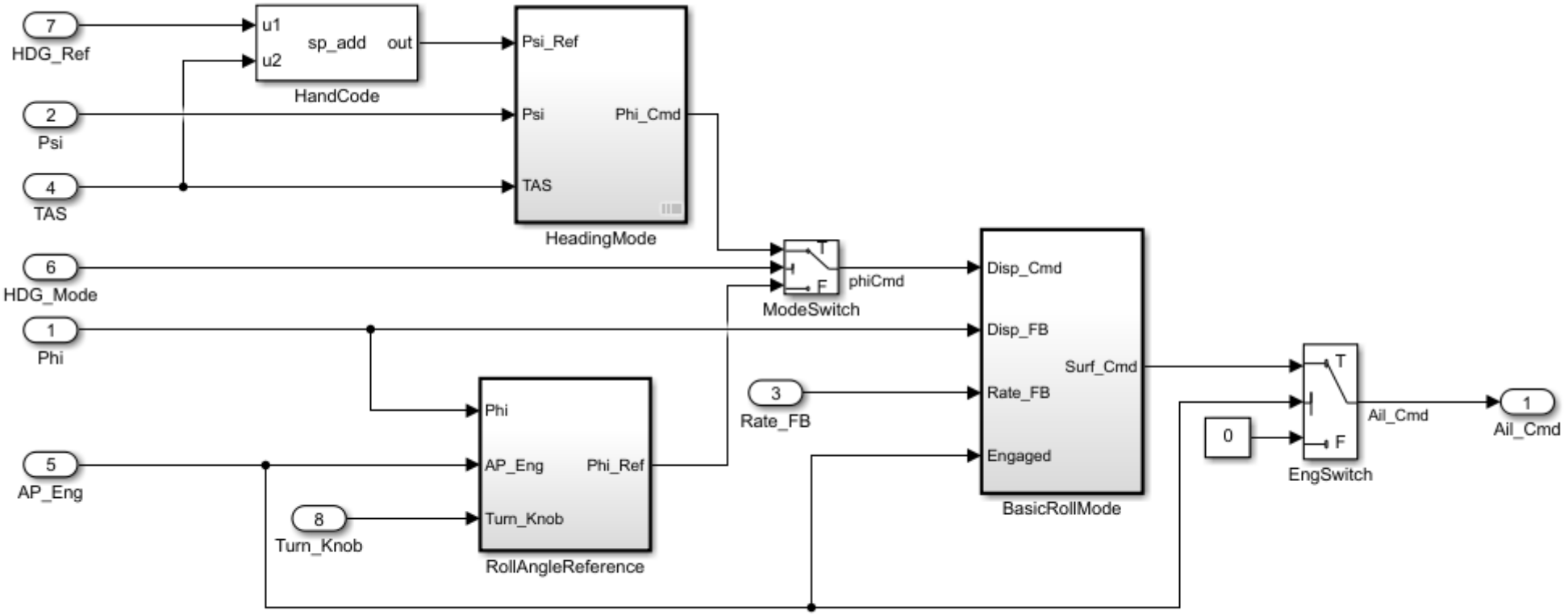
Agile

View on Board

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

Get Help / Give Feedback

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



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

Code Generation Report

Find: Match Case

Contents

- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Code Replacements Report
- Coder Assumptions

Generated Code

[-] **Main file**
- ert_main.c

[-] **Model files**
- rtwdemo_roll.c
- rtwdemo_roll.h

[+] **Shared files (1)**

[+] **Other files (1)**

Code Generation Report for 'rtwdemo_roll'

Model Information

Author	The MathWorks, Inc.
Last Modified By	The MathWorks, Inc.
Model Version	1.162
Tasking Mode	SingleTasking

Configuration settings at time of code generation

Code Information

System Target File	ert.tlc
Hardware Device Type	Intel->x86-64 (Windows64)
Simulink Coder Version	9.1 (R2019a) 23-Nov-2018
Timestamp of Generated Source Code	Wed Apr 10 10:36:32 2019
Location of Generated Source Code	C:\Work\MATLAB\ML_Expo\rtwdemo_roll_ert_rtw\
Type of Build	Model
Memory	Global Memory: 39(bytes) Maximum Stack:

OK Help

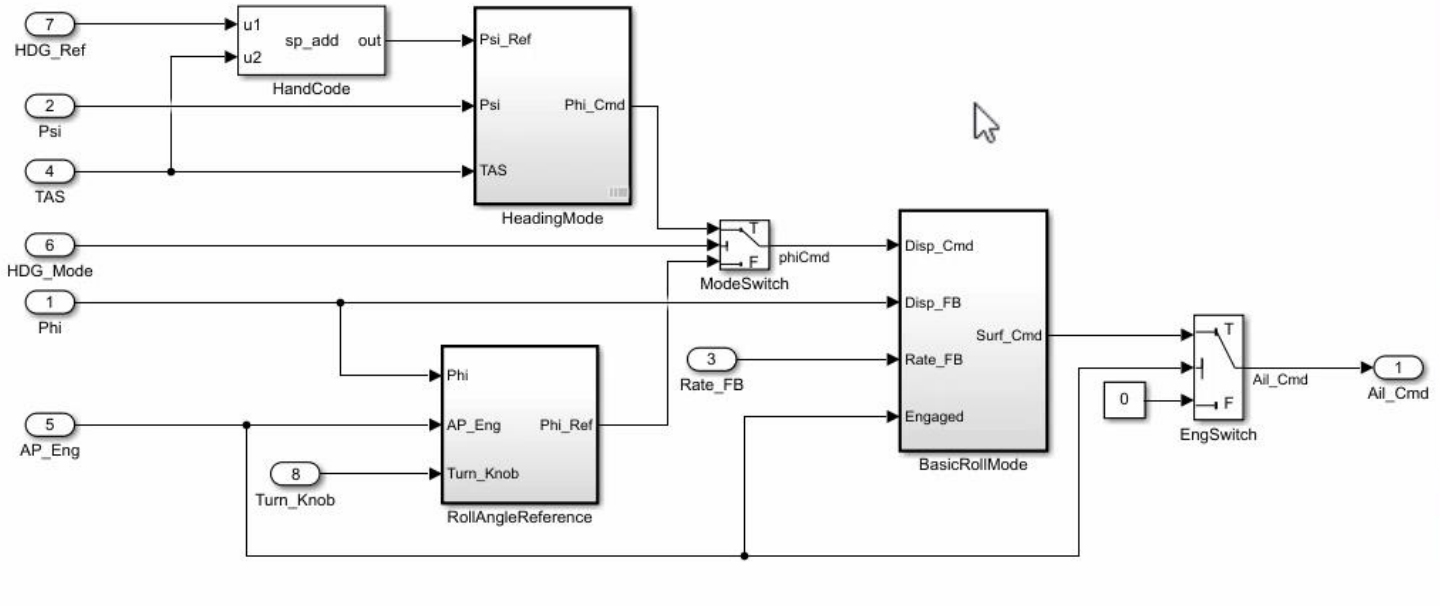
rtwdemo_roll - Simulink

File Edit View Display Diagram Simulation Analysis Code Tools Help

30 Normal

rtwdemo_roll

rtwdemo_roll



Ready 115% FixedStepDiscrete

Martin is a project manager
He is responsible for software quality of the project

Polyspace

localhost:9443/metrics/index.html?a=metrics&p=1

DASHBOARD

jabraham

Project Overview | Run-time Checks | Code Metrics | MISRA C:2012 | Layout | Open in Desktop | Review

DASHBOARDS | ENVIRONMENT | REVIEW

PROJECT EXPLORER

- public
  - Proj_Zen
  - Test_Area

PROJECT DETAILS

Project

Name	public
Tools	Code Prover
Coding Standards	Custom Rules, MISRA C:2012
Number of Runs	6

SUPPORT REPORT

Project Overview | Quality Objectives | Run-time Checks | Code Metrics

Summary — public

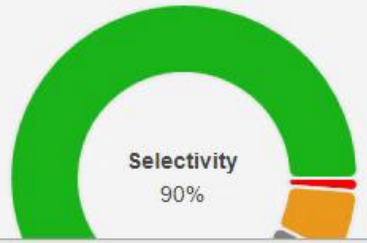
Open Issues

Open	104
New	0
Assigned To Me	0
Unassigned	104

Code Metrics

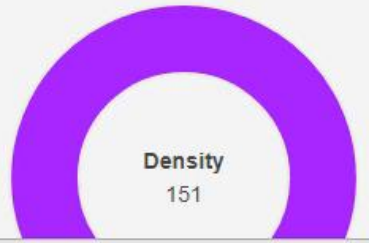
Sub-project(s)	2
Number of Files	7
Number of Lines Without Comment	450
Cyclomatic Complexity	6

Run-time Checks — Open 32



Red	4
Orange	21
Gray	8
Green	300

Coding Standards — Open 68



To Do 68

public

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
- Supports 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.

Polyspace Helps Makes C, C++, and Ada Safe and Secure

Safety	Security
Standards: • DO-178 (aero) • ISO 26262 (auto) • IEC 61508 (industrial) • IEC 62304 (medical) • EN 50128 (rail) • MISRA • AUTOSAR	**Standards:** • CERT-C • CWE • ISO 17961 • MISRA-C:2012 Appendix 1 • Tainted data tracking

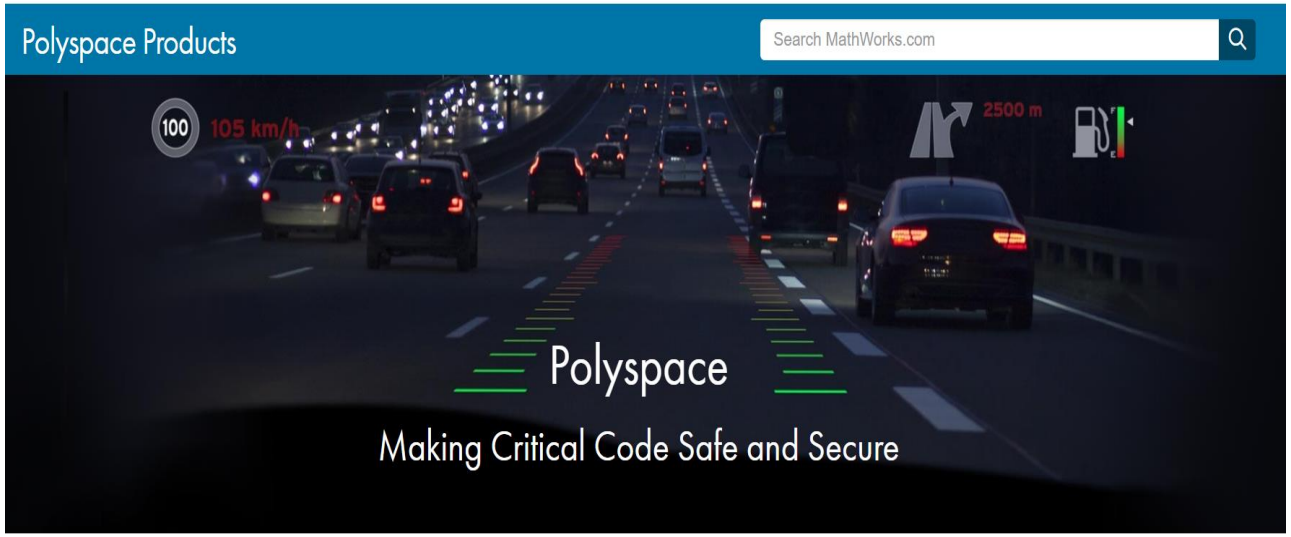
Reliability and **Robustness**
Code Proving • Prove absence of critical runtime errors (or find even the slightest vulnerability) • Exhaustive: all possible inputs, control flows, data flows (no instrumentation, execution, test cases) • Sound: no false negatives

Quality
• Coding Standards • Find Probable Bugs, Defects • Code Metrics • Formal Method: Runtime Behavior, Debugger-like view • Review Scopes / Software Quality Objectives • Simulink Integration: trace issues in generated code back to model

Learn More

Visit MathWorks Code Verification Solution Page:

https://www.mathworks.com/products/polyspace.html



Polyspace® static code analysis products use formal methods to prove the absence of critical run-time errors under all possible control flows and data flows. They include checkers for coding rules, security vulnerabilities, code metrics, and hundreds of additional classes of bugs.

MathWorks® | *Training Services*

Polyspace for C/C++ Code Verification

This two-day course discusses the use of Polyspace Bug Finder™ and Polyspace Code Prover™ to prove code correctness, improve software quality metrics, and ensure product integrity.

Topics include:

- Creating a verification project
- Reviewing and understanding verification results
- Emulating target execution environments
- Handling missing functions and data
- Managing unproven code (color-coded in orange by Polyspace® products)
- Applying MISRA C® rules
- Reporting

Thank You