MATLAB EXPO 2019

Accelerating embedded software verification

with Polyspace static code analysis

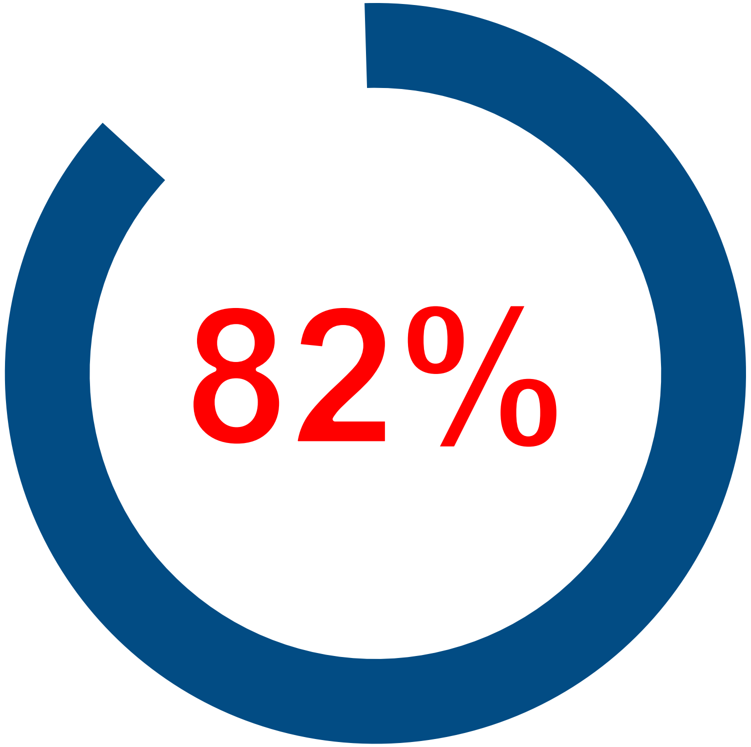
Stefan David

Agenda

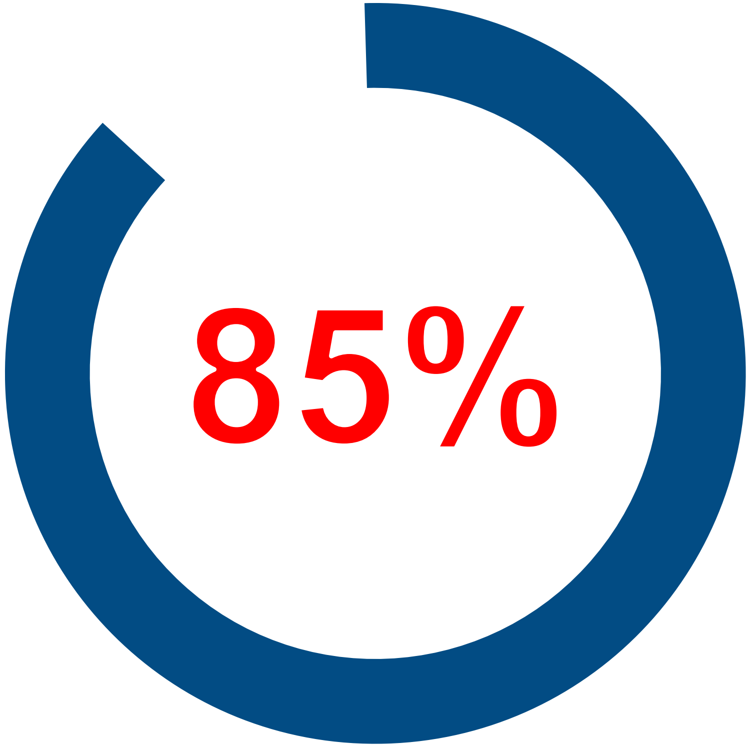
1. Making Software Safe and Secure
2. Polyspace Static Analysis
3. Team Collaboration with Polyspace

1. Making Software Safe and Secure

Security is on consumers' minds



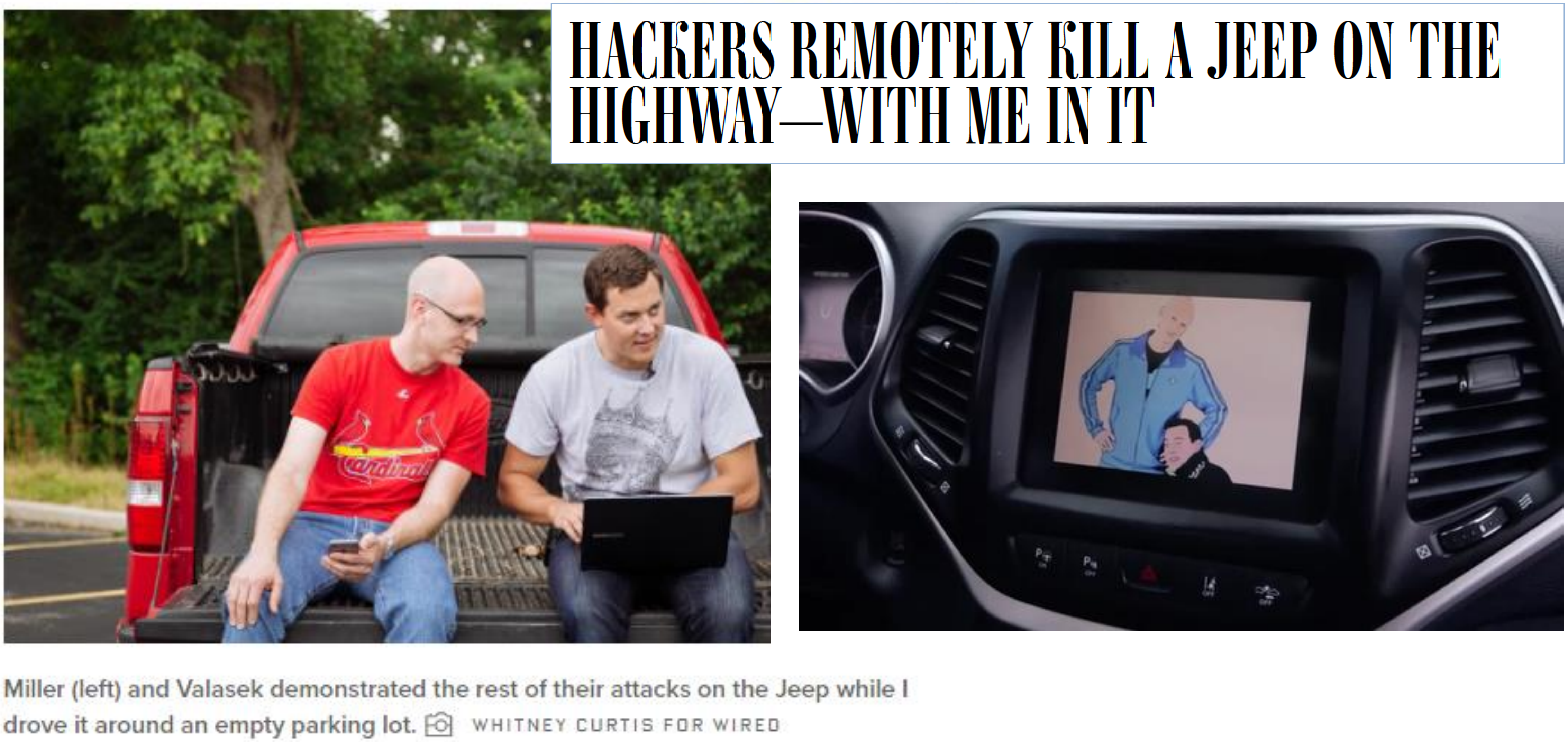
…of customers would never buy from an OEM if they had been hacked



…of automakers admit their organization had been hacked in the past 2 years

according to 2016 KPMG Consumer Loss Barometer study

In the News.... Embedded Software Security - New Challenge



HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

Miller (left) and Valasek demonstrated the rest of their attacks on the Jeep while I drove it around an empty parking lot. WHITNEY CURTIS FOR WIRED

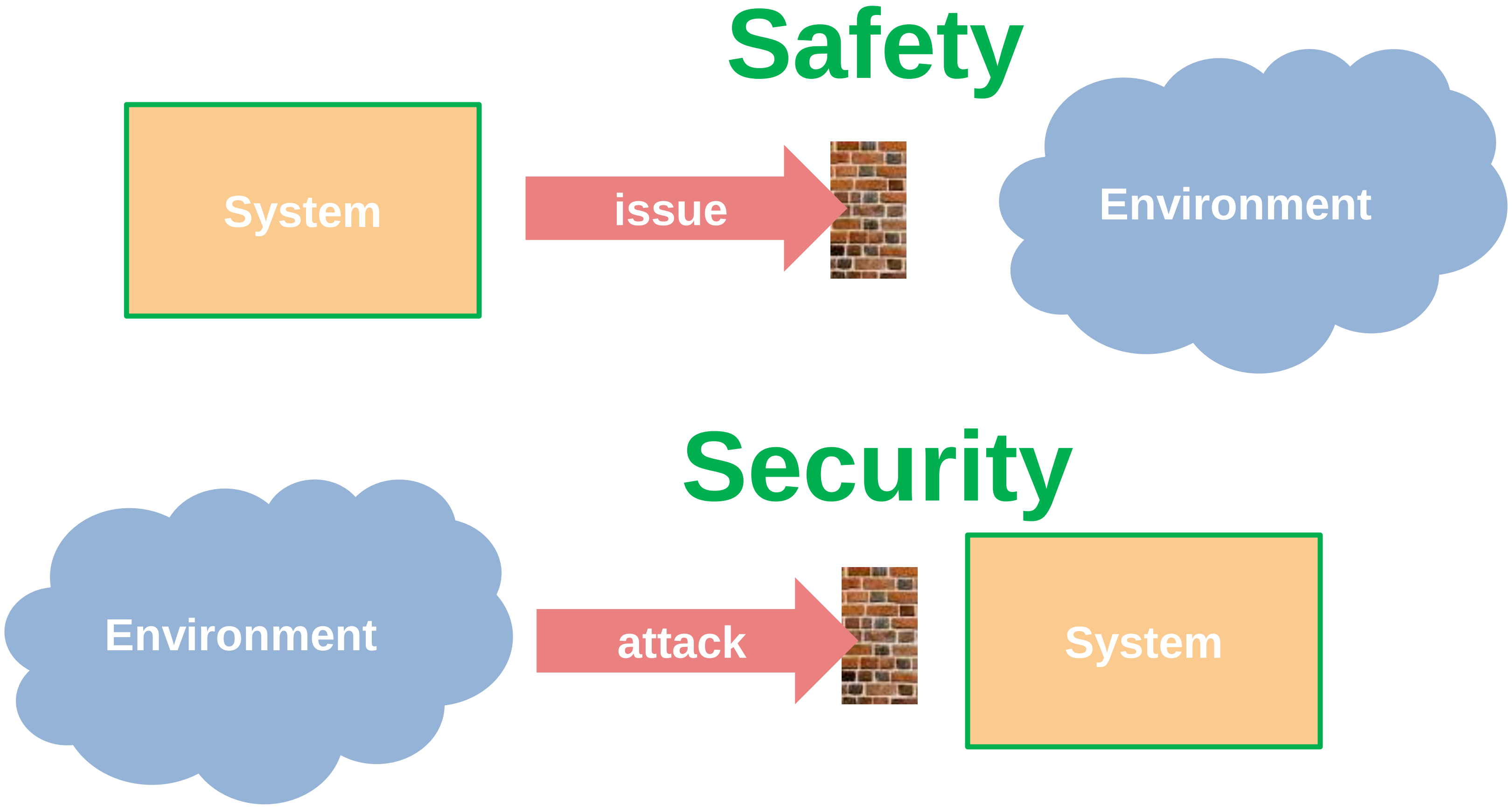
Source: https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/

In the News.... Embedded Software Security - New Challenge

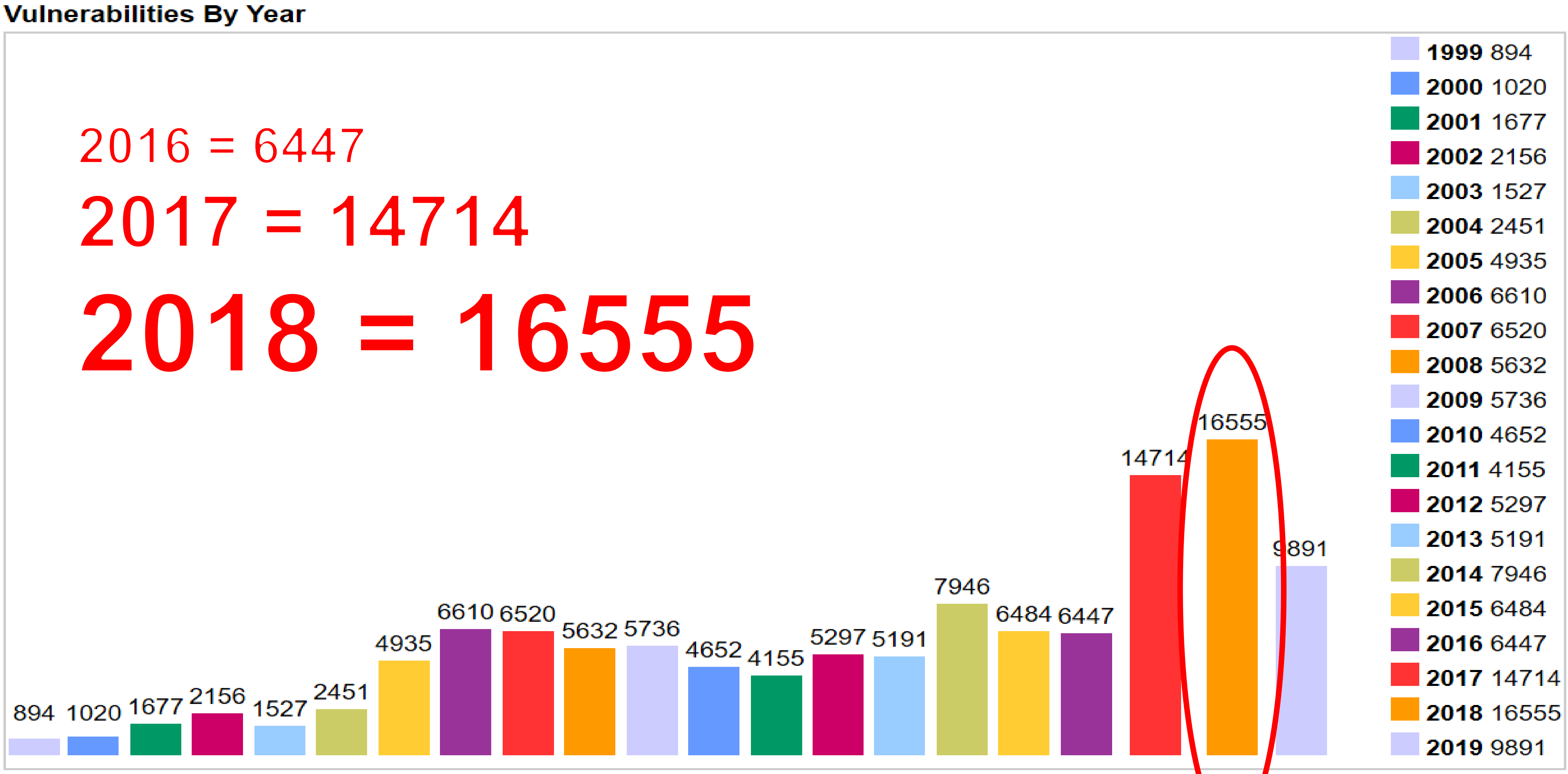
MA1

Source: https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/

Safety & Security Goals



Note: Security issues may cause safety issues source: https://www.cvedetails.com (CVE ... Common Vulnerabilities and Exposures)



2016 = 6447

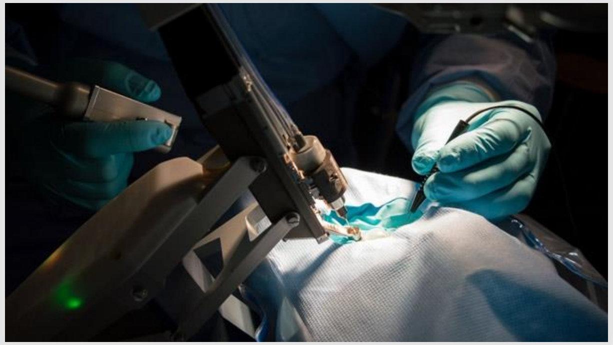
2017 = 14714

2018 = 16555

When Software Safety and Security Matter

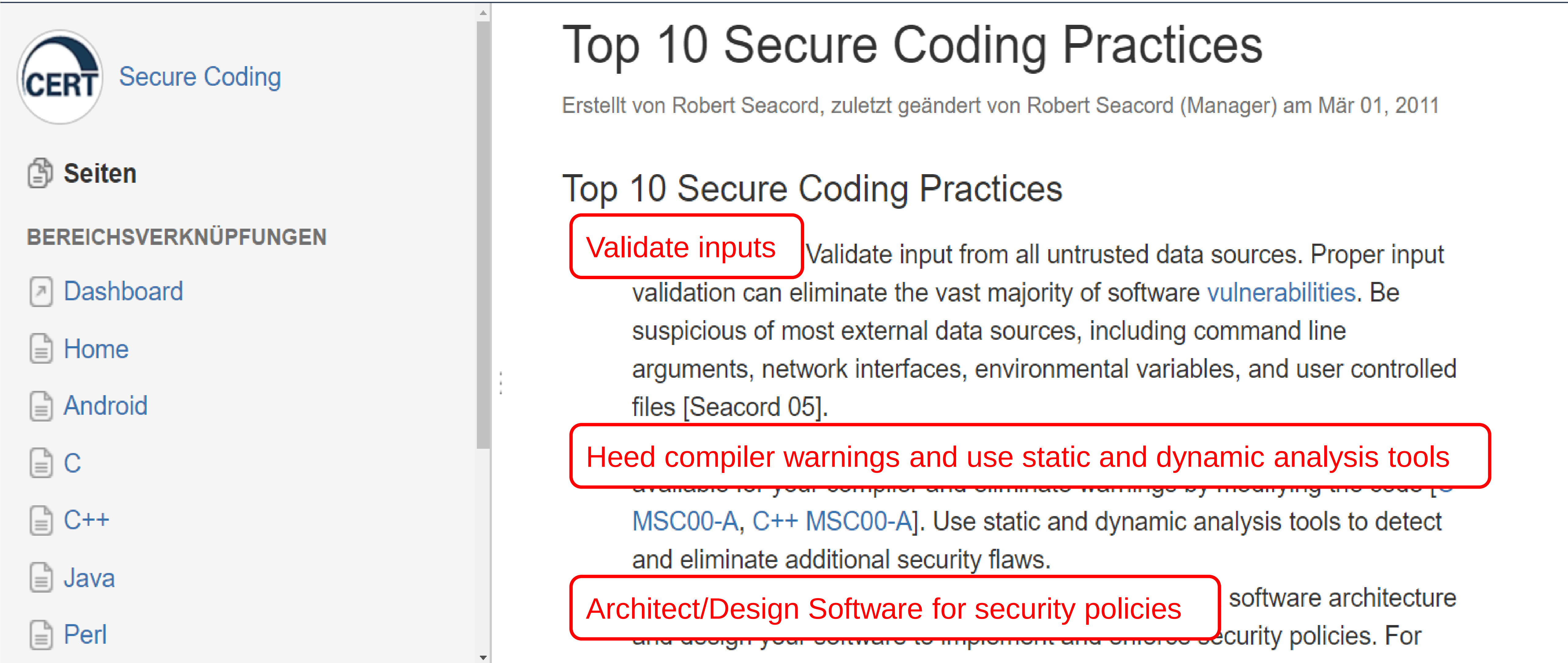
- Industries where safety and security matter
  - Automotive, Aerospace, Medical Device, Industrial Machinery

- Governed by functional safety and other standards
  - ISO 26262, DO-178, IEC 62304, IEC 61508
  - ISO/SAE 21434, RTCA DO-326
  - MISRA, CERT, AUTOSAR



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

source: https://www.securecoding.cert.org



CERT Secure Coding

Seiten

BEREICHSVERKNÜPFUNGEN

- Dashboard
- Home
- Android
- C
- C++
- Java
- Perl

Top 10 Secure Coding Practices

Erstellt von Robert Seacord, zuletzt geändert von Robert Seacord (Manager) am Mär 01, 2011

Top 10 Secure Coding Practices

Validate inputs

Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files [Seacord 05].

Heed compiler warnings and use static and dynamic analysis tools

available for your compiler and eliminate warnings by modifying the code [C MSC00-A, C++ MSC00-A]. Use static and dynamic analysis tools to detect and eliminate additional security flaws.

Architect/Design Software for security policies

software architecture and design your software to implement and enforce security policies. For

“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, IBM

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

2. Polyspace Static Analysis

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

Proving Absence of Critical Run-Time Errors

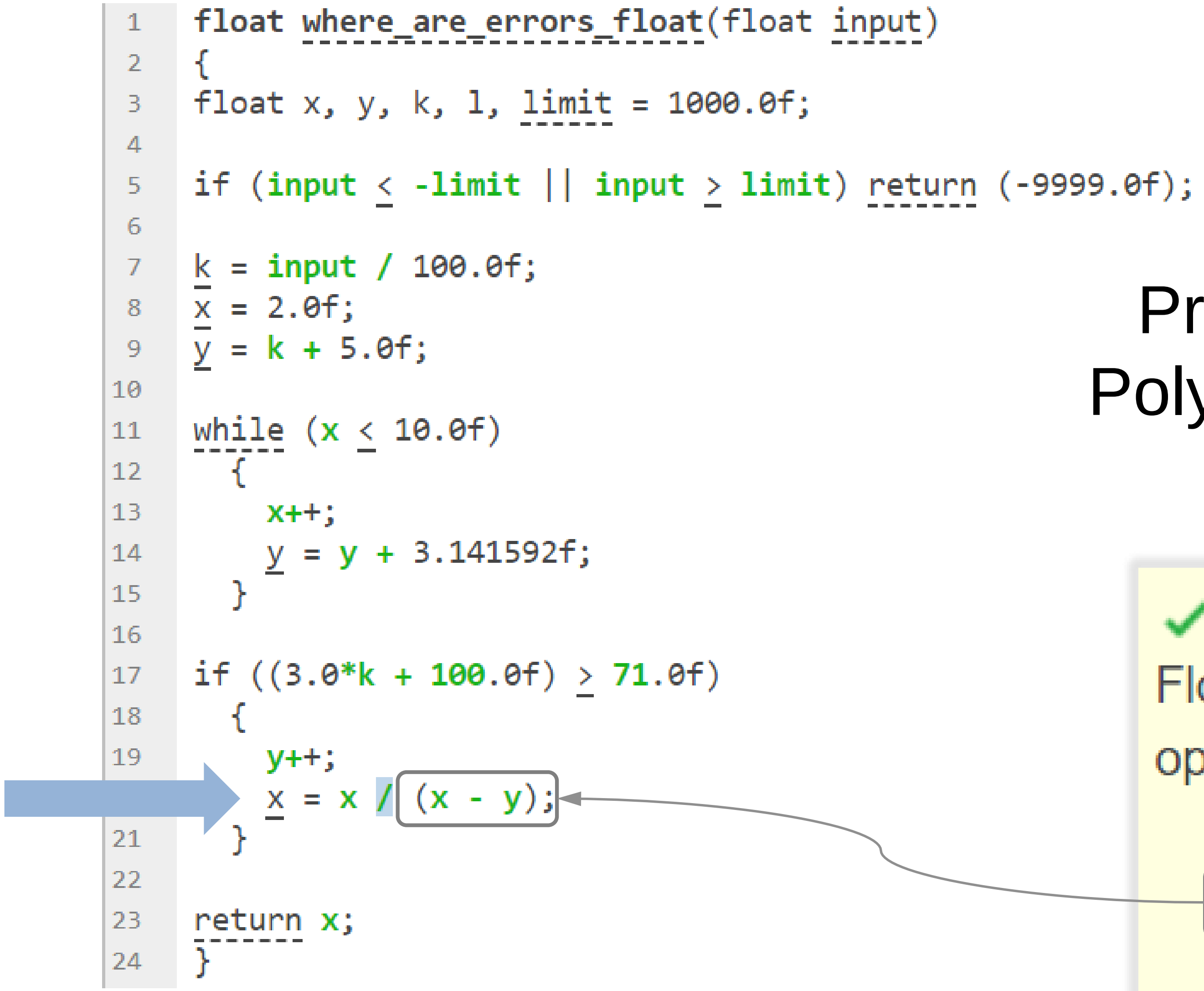
```
float x, y;

...

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?

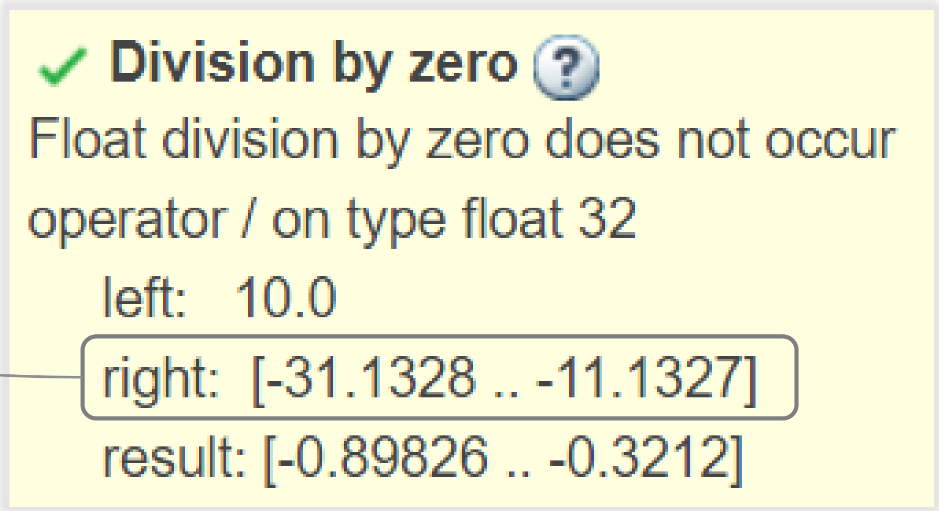
- How do you prove that this code will not fail?

Proving Absence of Critical Run-Time Errors



```
float where_are_errors_float(float input)
{
float x, y, k, l, limit = 1000.0f;

if (input < -limit || input > limit) return (-9999.0f);

k = input / 100.0f;
x = 2.0f;
y = k + 5.0f;

while (x < 10.0f)
  {
    x++;
    y = y + 3.141592f;
  }

if ((3.0*k + 100.0f) > 71.0f)
  {
    y++;
    x = x / (x - y);
  }

return x;
}
```

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]

Experiences from the field...

Using Polyspace code verifiers...

- Identified and fixed potential run-time errors and unsafe code
- Reliably analyzed C codebase early, without test cases and compilation!

An idea born in Switzerland
SOLARIMPULSE
AROUND THE WORLD IN A SOLAR AIRPLANE

“Independent, systematic code reviews, compliance to MISRA-C”

*“Bug Finder and Code Prover provided **1-2 Man-Year savings** and automated capability **in parallel to development which were not available otherwise**“*

(Source: Ralph Paul, Head of Flight Test & Dynamics, Solar Impulse)

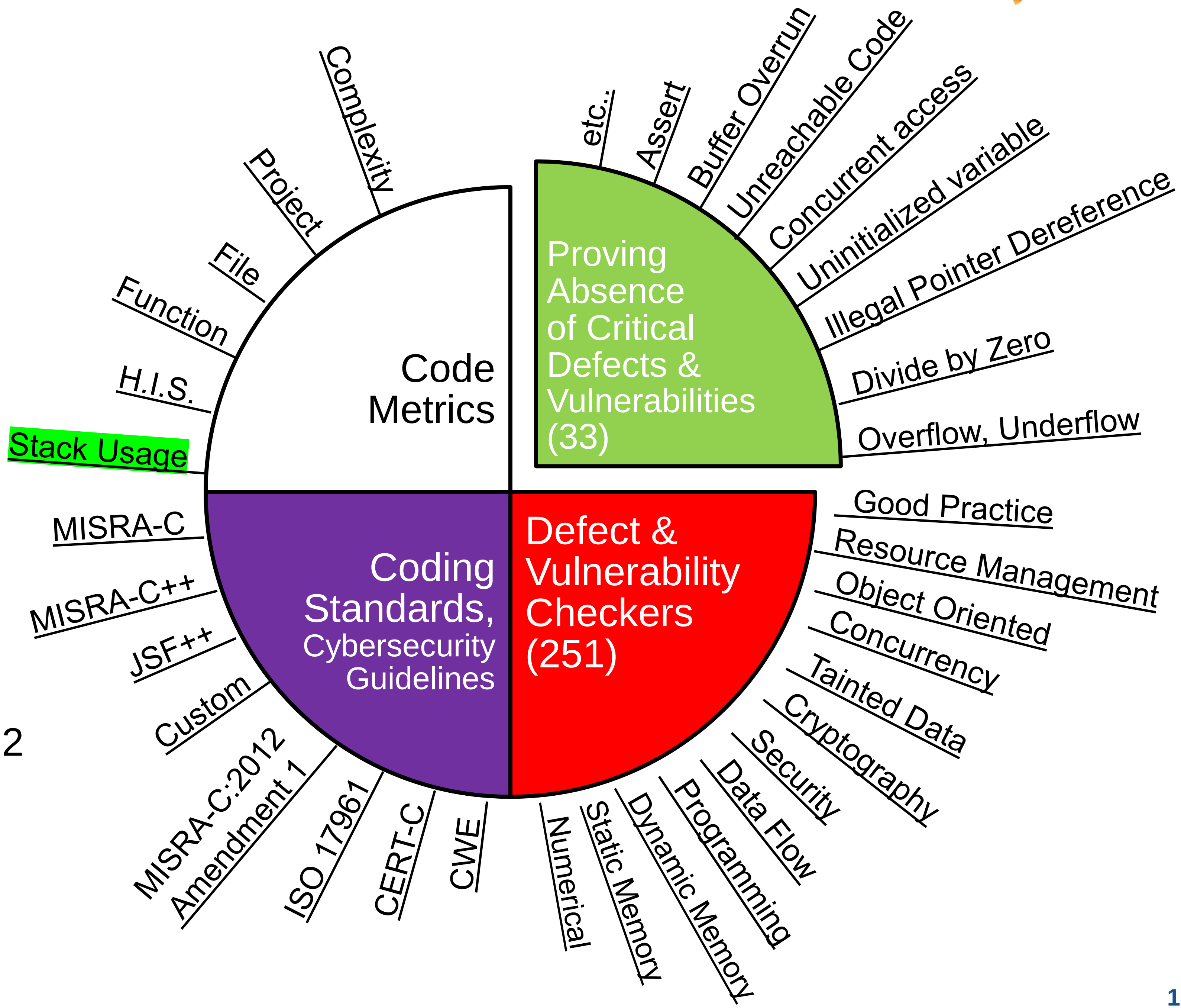
Polyspace Tools

Bug Finder

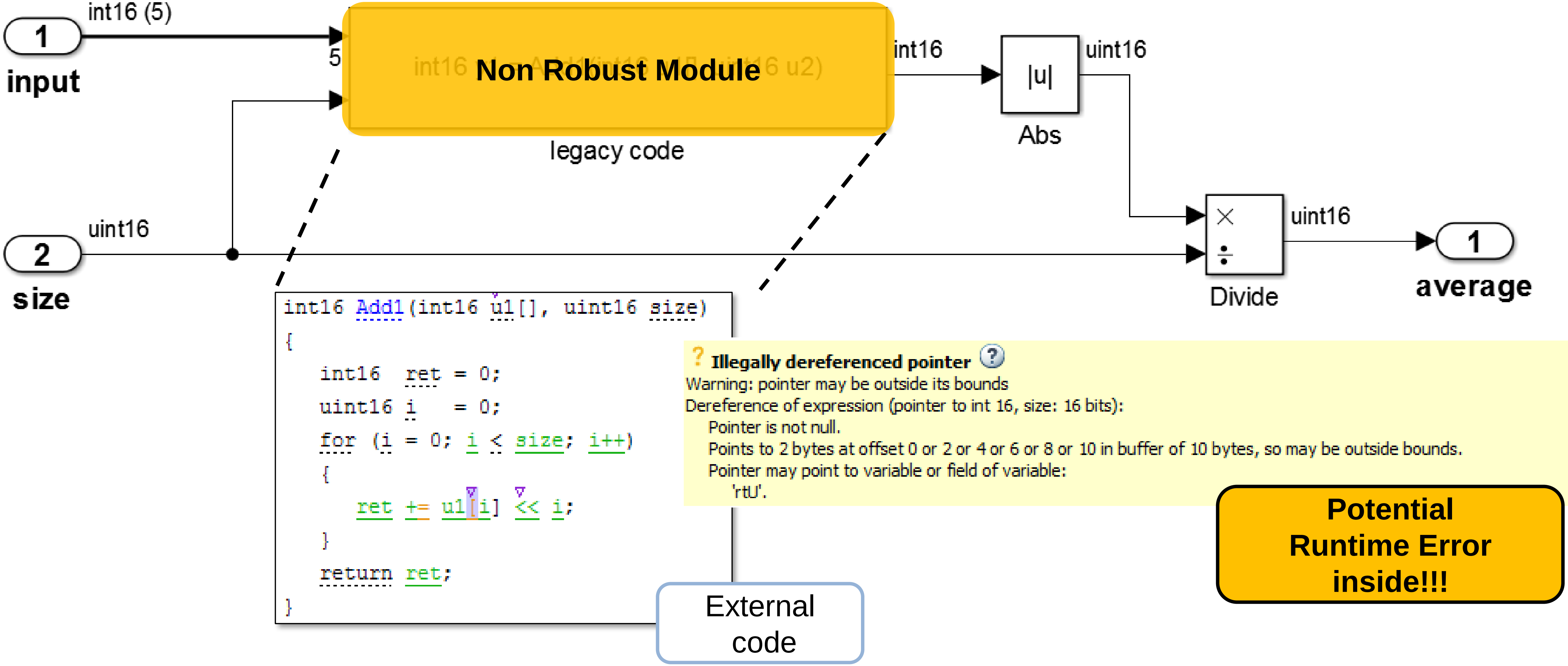
- Produce code metrics
- Check coding standards
- Find defects and vulnerabilities

Code Prover

- Proves code Safe and Secure
- 33 most critical run-time checks
- Supports DO-178 and ISO 26262



Example: Optimize design and architecture



```
int16 Add1(int16 u1[], uint16 size)
{
   int16  ret = 0;
   uint16 i   = 0;
   for (i = 0; i < size; i++)
   {
      ret += u1[i] << i;
   }
   return ret;
}
```

Illegally dereferenced pointer

Warning: pointer may be outside its bounds

Dereference of expression (pointer to int 16, size: 16 bits):

Pointer is not null.

Points to 2 bytes at offset 0 or 2 or 4 or 6 or 8 or 10 in buffer of 10 bytes, so may be outside bounds.

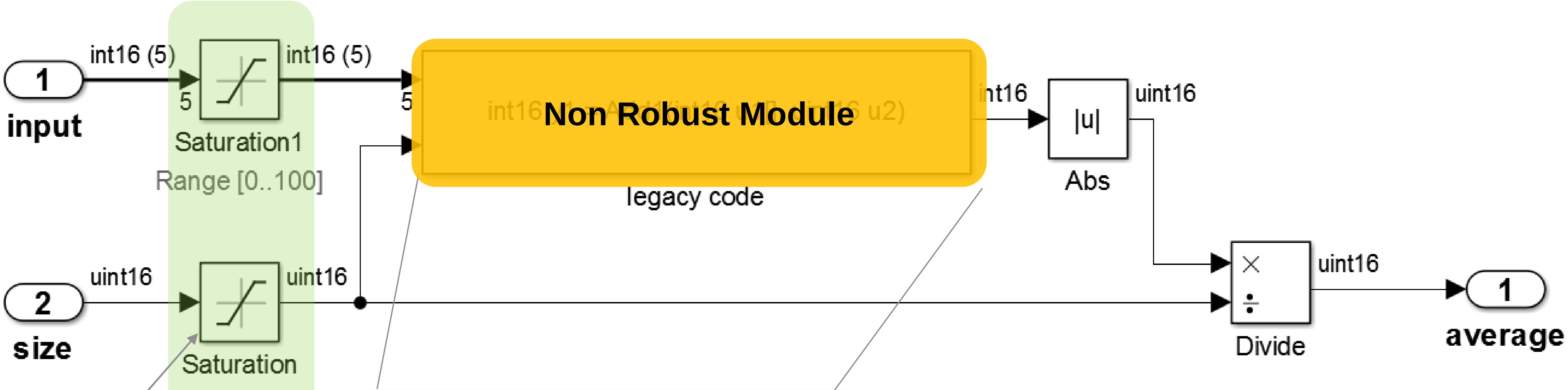
Pointer may point to variable or field of variable:

'rtU'.

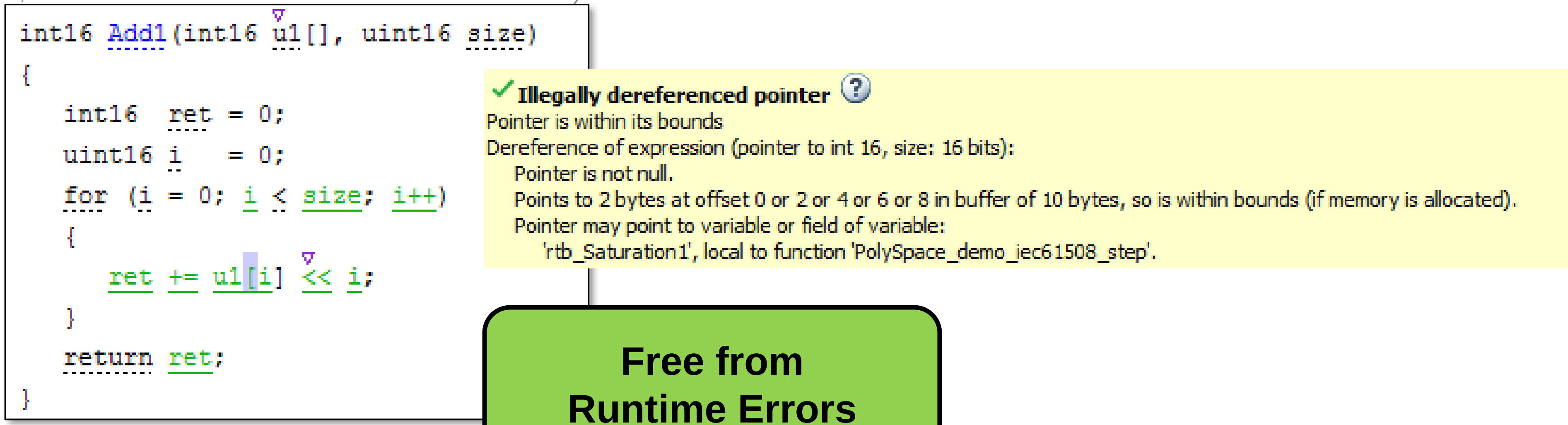
External code

Potential Runtime Error inside!!!

Example: Optimize design and architecture



Additional Range-Limiting Code



```
int16 Add1(int16 u1[], uint16 size)
{
    int16  ret = 0;
    uint16 i   = 0;
    for (i = 0; i < size; i++)
    {
        ret += u1[i] << i;
    }
    return ret;
}
```

✓ **Illegally dereferenced pointer**

Pointer is within its bounds

Dereference of expression (pointer to int 16, size: 16 bits):

Pointer is not null.

Points to 2 bytes at offset 0 or 2 or 4 or 6 or 8 in buffer of 10 bytes, so is within bounds (if memory is allocated).

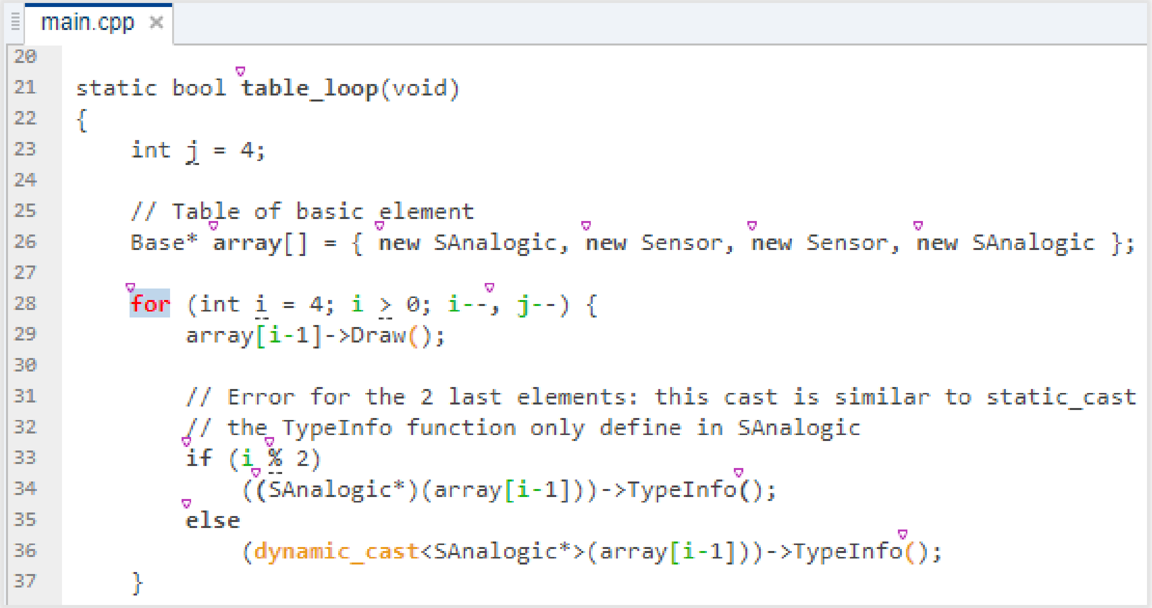
Pointer may point to variable or field of variable:

'rtb_Saturation1', local to function 'PolySpace_demo_iec61508_step'.

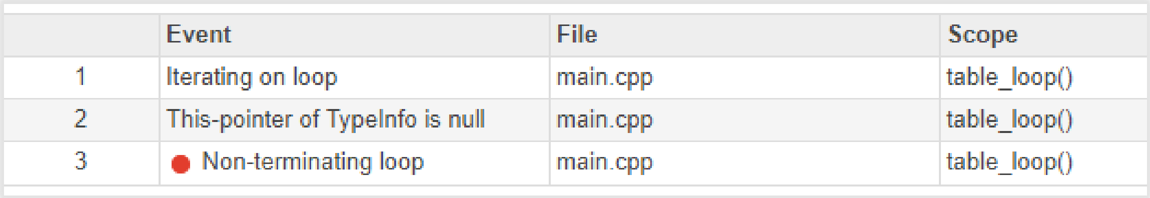
Free from Runtime Errors

Using Static Analysis to Make Software Safe and Secure

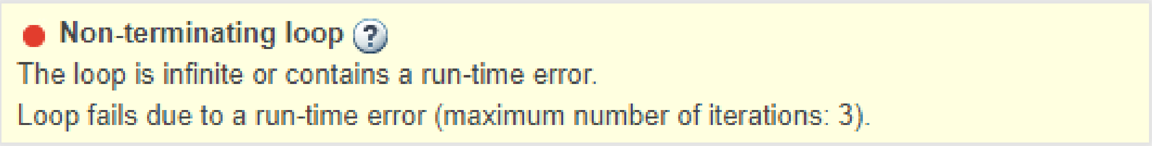
- Find bugs without code execution
  - Code analyzed without running tests
  - Identify bugs and coding rule violations for MISRA, AUTOSAR, CERT
- Prove absence of critical run-time errors
  - Identify code that will never experience errors regardless of run-time conditions
- Complements dynamic testing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



```
static bool table_loop(void)
{
    int j = 4;

    // Table of basic element
    Base* array[] = { new SAnalogic, new Sensor, new Sensor, new SAnalogic };

    for (int i = 4; i > 0; i--, j--) {
        array[i-1]->Draw();

        // Error for the 2 last elements: this cast is similar to static_cast
        // the TypeInfo function only define in SAnalogic
        if (i % 2)
            ((SAnalogic*)(array[i-1]))->TypeInfo();
        else
            (dynamic_cast<SAnalogic*>(array[i-1]))->TypeInfo();
    }
```



	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).

Polyspace Customer References

Electronic Steering Lock

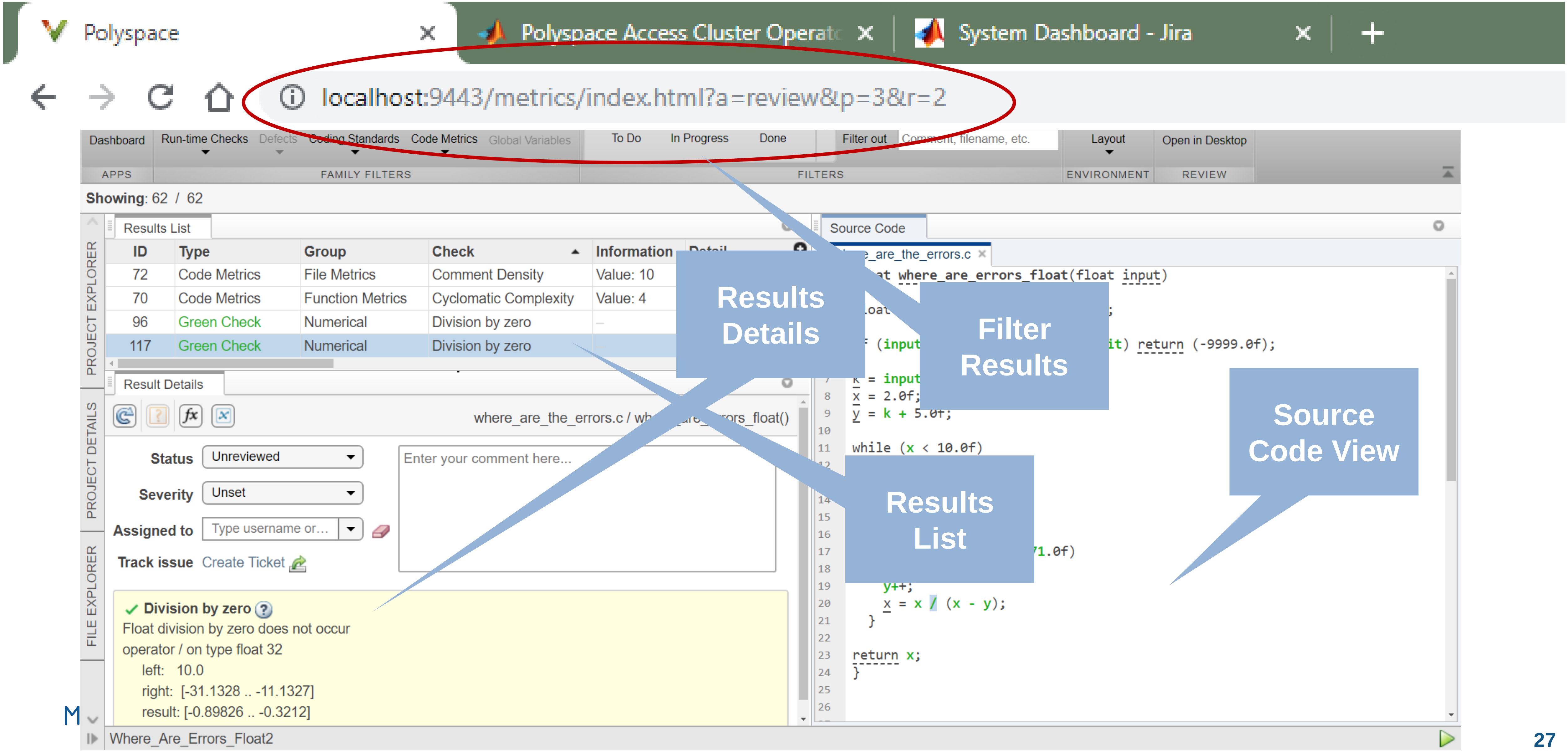
KOSTAL Asia R&D Center Receives ISO 26262 ASIL D Certification for Automotive Software

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

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



Showing: 62 / 62

ID	Type	Group	Check	Information	Detail
72	Code Metrics	File Metrics	Comment Density	Value: 10	
70	Code Metrics	Function Metrics	Cyclomatic Complexity	Value: 4	
96	Green Check	Numerical	Division by zero	–	
117	Green Check	Numerical	Division by zero		

where_are_the_errors.c / where_are_errors_float()

Status Unreviewed

Severity Unset

Assigned to Type username or…

Track issue Create Ticket

Enter your comment here...

✓ **Division by zero**

Float division by zero does not occur

operator / on type float 32

left: 10.0

right: [-31.1328 .. -11.1327]

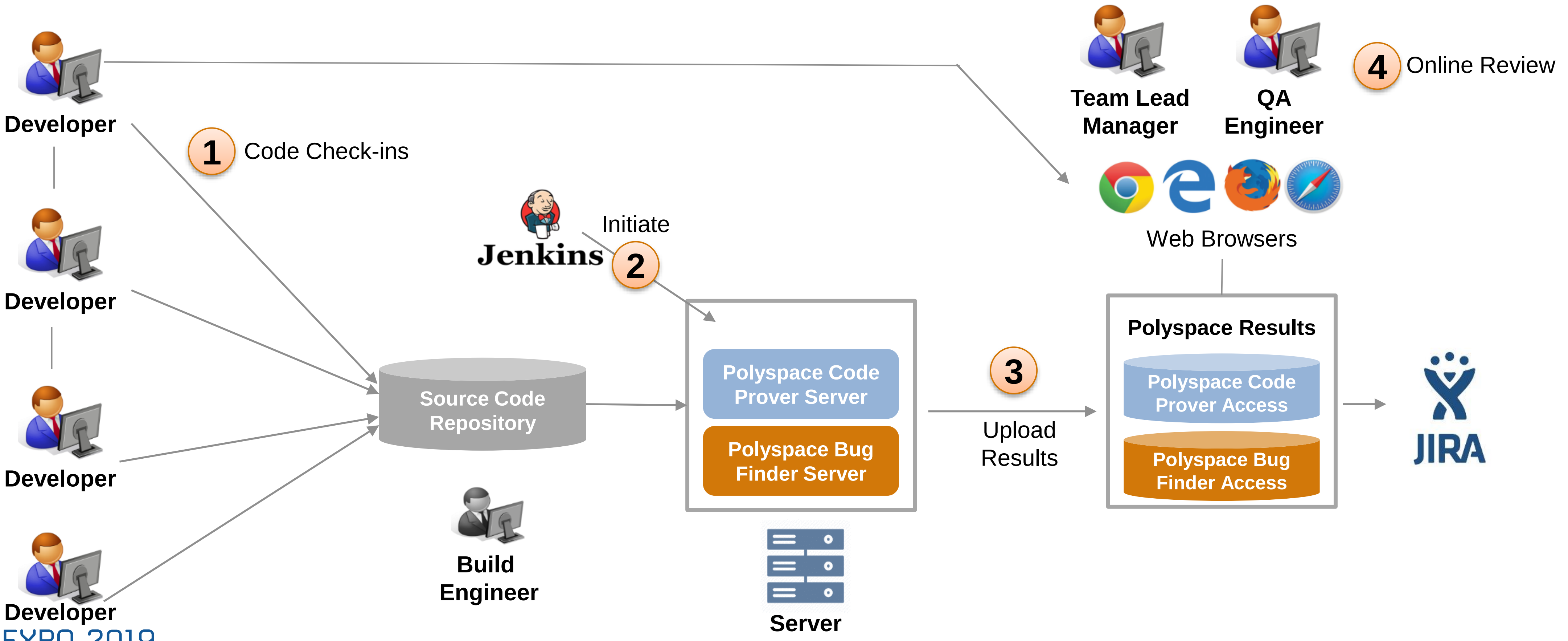
result: [-0.89826 .. -0.3212]

```
at where_are_errors_float(float input)

oat                                      ;
  (input                              it) return (-9999.0f);

k = input
x = 2.0f;
y = k + 5.0f;

while (x < 10.0f)

                              1.0f)
      y++;
      x = x / (x - y);
   }

return x;
}
```

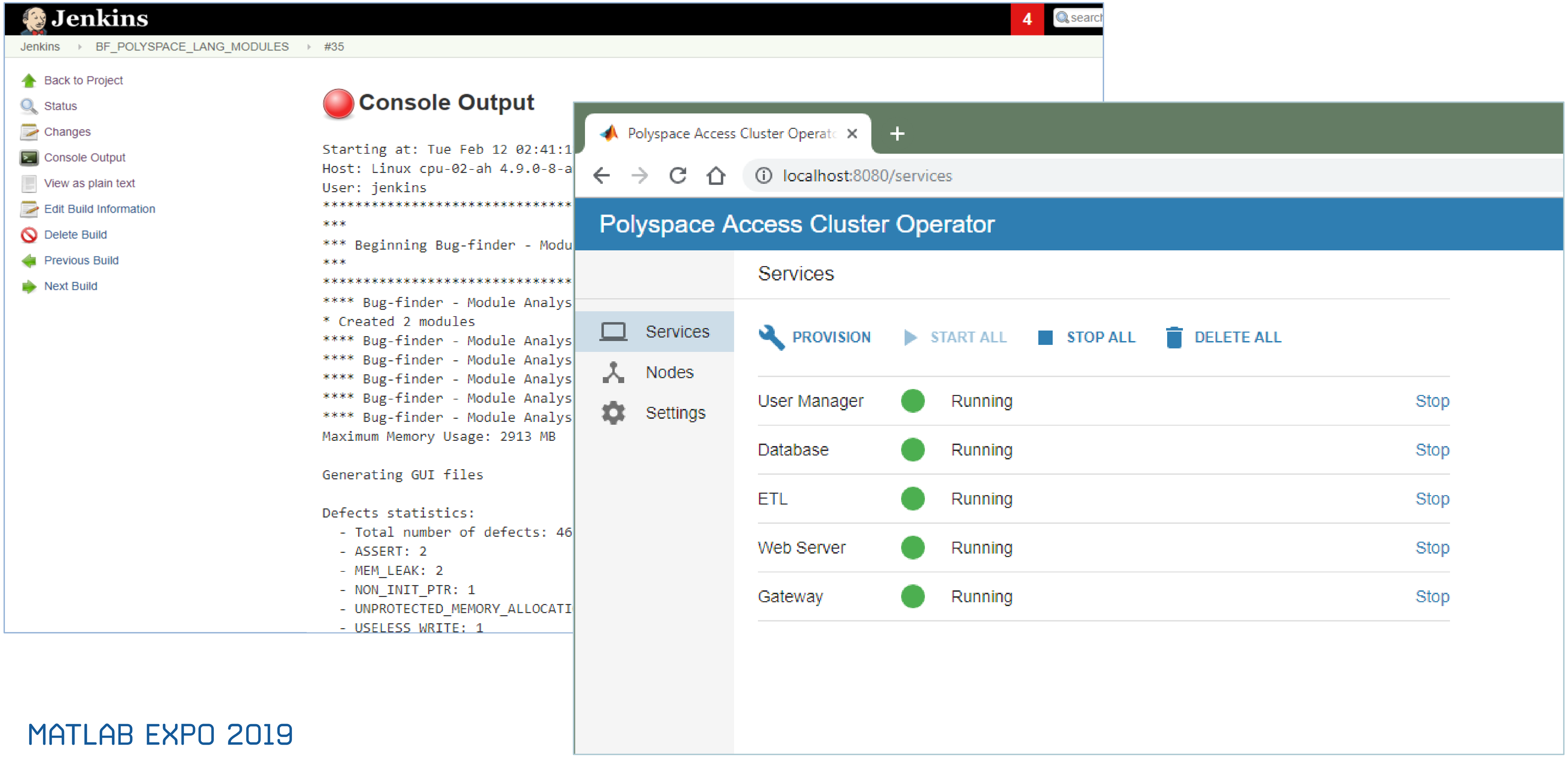
Where_Are_Errors_Float2

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



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



Console Output

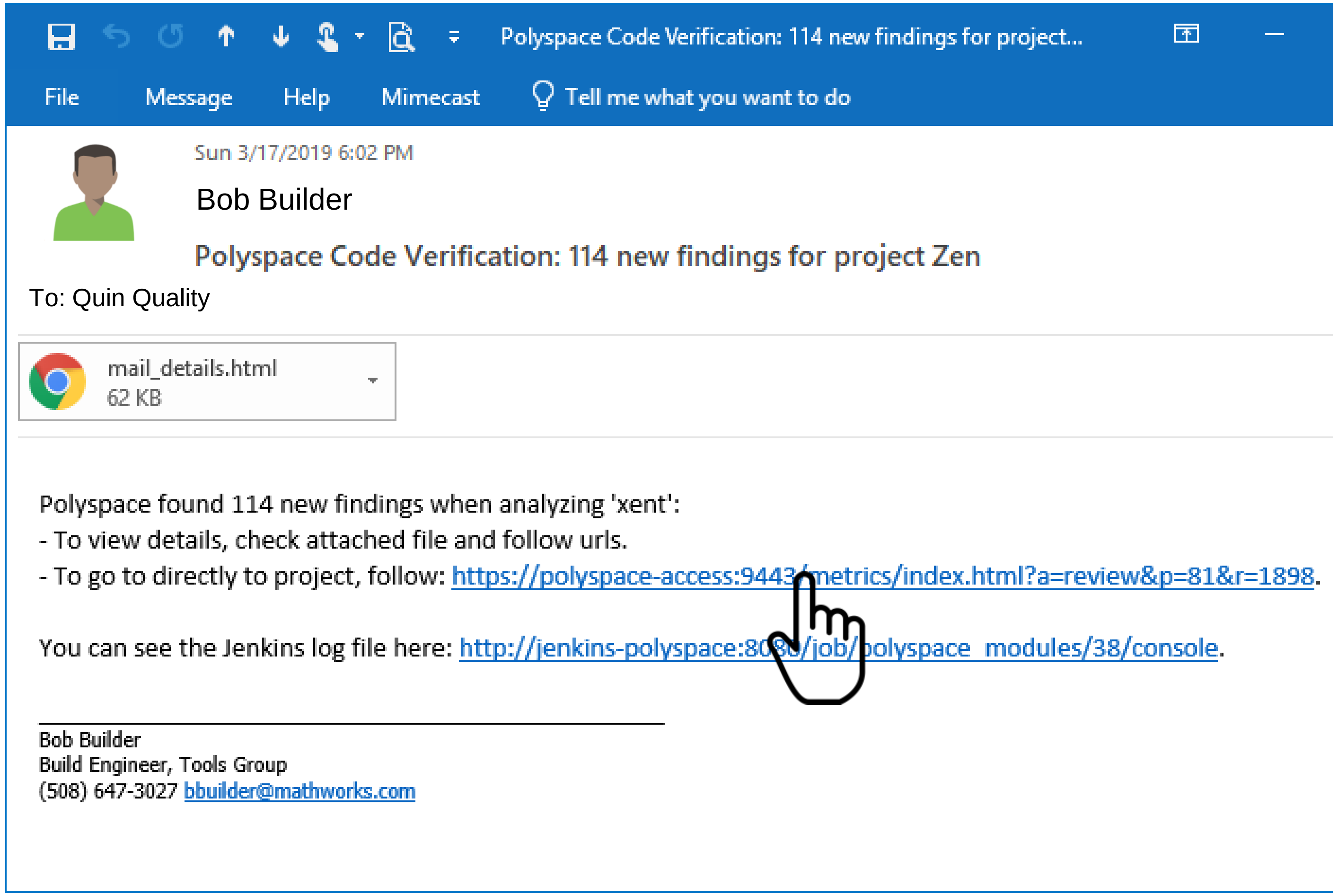
```
Starting at: Tue Feb 12 02:41:1
Host: Linux cpu-02-ah 4.9.0-8-a
User: jenkins
*******************************
***
*** Beginning Bug-finder - Modu
***
*******************************
**** Bug-finder - Module Analys
* Created 2 modules
**** Bug-finder - Module Analys
**** Bug-finder - Module Analys
**** Bug-finder - Module Analys
**** Bug-finder - Module Analys
**** Bug-finder - Module Analys
Maximum Memory Usage: 2913 MB

Generating GUI files

Defects statistics:
  - Total number of defects: 46
  - ASSERT: 2
  - MEM_LEAK: 2
  - NON_INIT_PTR: 1
  - UNPROTECTED_MEMORY_ALLOCATI
  - USELESS_WRITE: 1
```

Services

User Manager	●	Running	Stop
Database	●	Running	Stop
ETL	●	Running	Stop
Web Server	●	Running	Stop
Gateway	●	Running	Stop

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



Polyspace Code Verification: 114 new findings for project...
File
Message
Help
Mimecast
Tell me what you want to do

Sun 3/17/2019 6:02 PM

Bob Builder

Polyspace Code Verification: 114 new findings for project Zen

To: Quin Quality

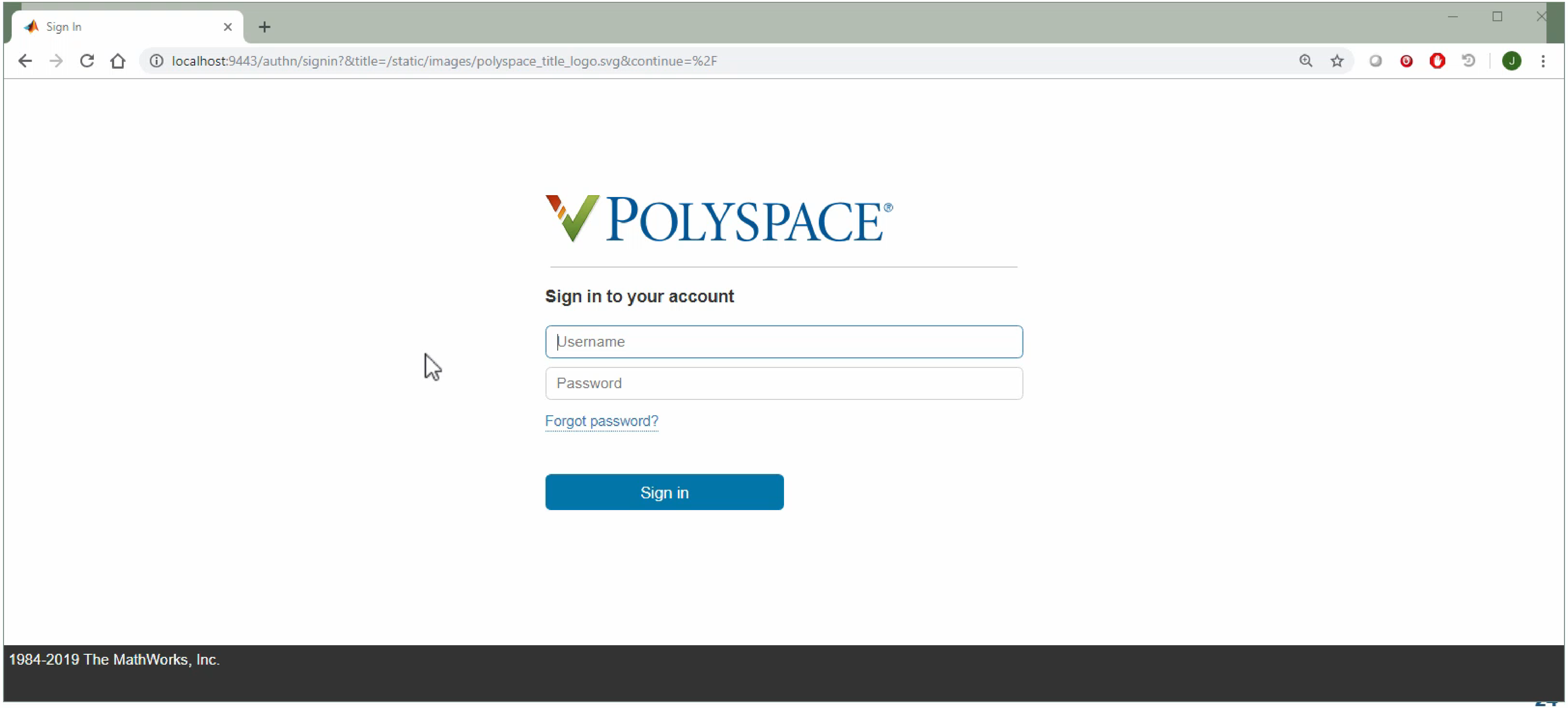
mail_details.html
62 KB

Polyspace found 114 new findings when analyzing 'xent':
- To view details, check attached file and follow urls.
- To go to directly to project, follow: https://polyspace-access:9443/metrics/index.html?a=review&p=81&r=1898.

You can see the Jenkins log file here: http://jenkins-polyspace:8080/job/polyspace_modules/38/console.

Bob Builder
Build Engineer, Tools Group
(508) 647-3027 bbuilder@mathworks.com

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



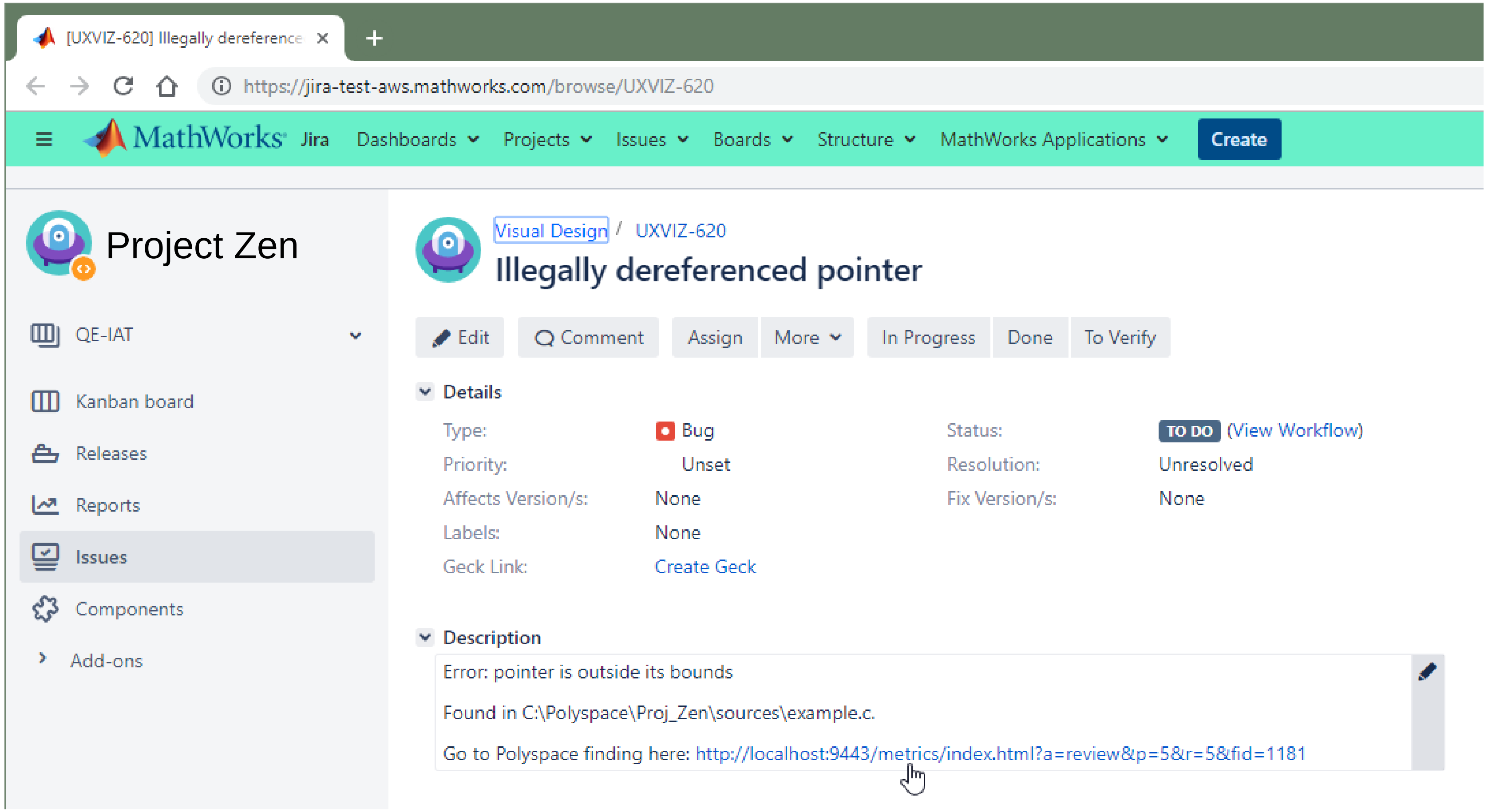
localhost:9443/authn/signin?&title=/static/images/polyspace_title_logo.svg&continue=%2F

Sign in to your account

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



Visual Design / UXVIZ-620

Illegally dereferenced pointer

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		

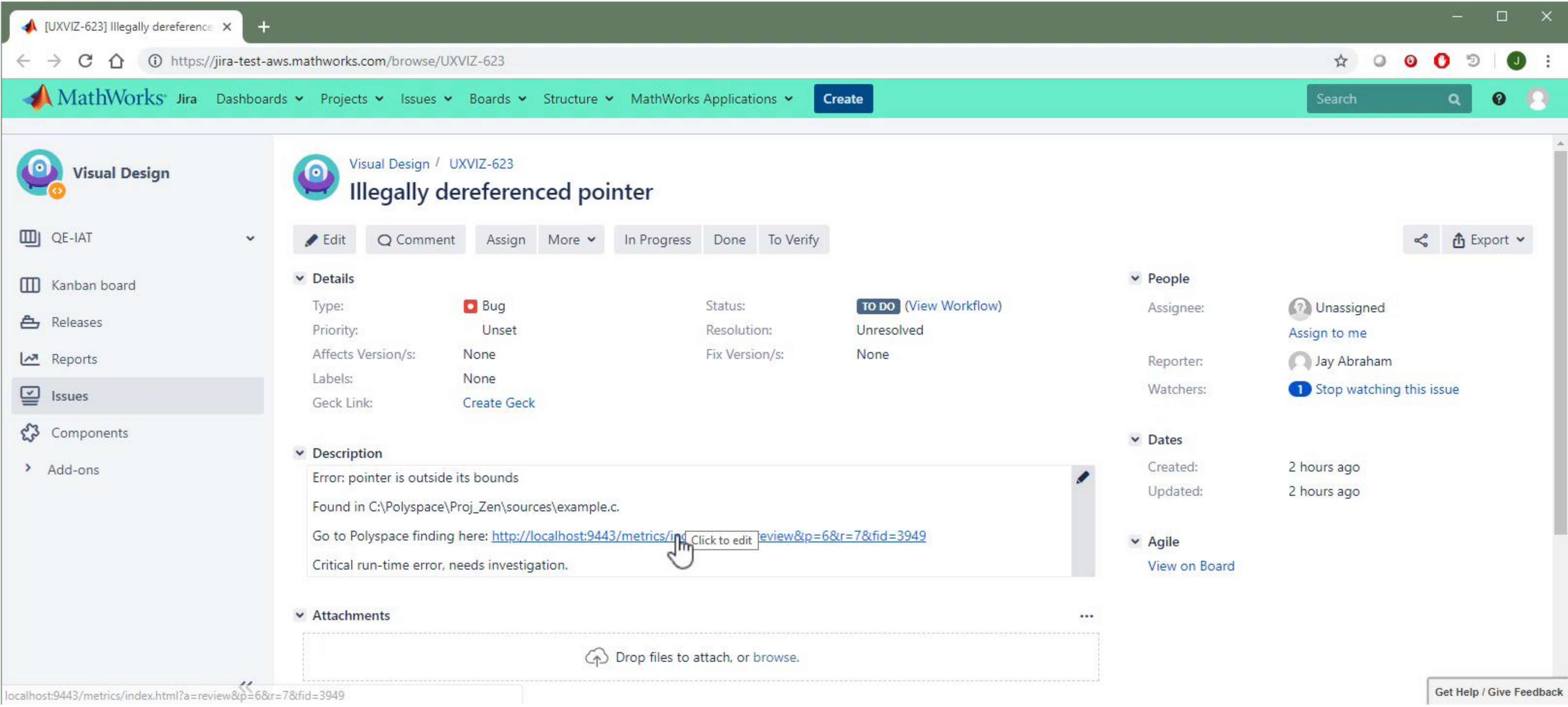
Description

Error: pointer is outside its bounds

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

Go to Polyspace finding here: http://localhost:9443/metrics/index.html?a=review&p=5&r=5&fid=1181

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



Visual Design / UXVIZ-623

Illegally dereferenced pointer

Details

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

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[...]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

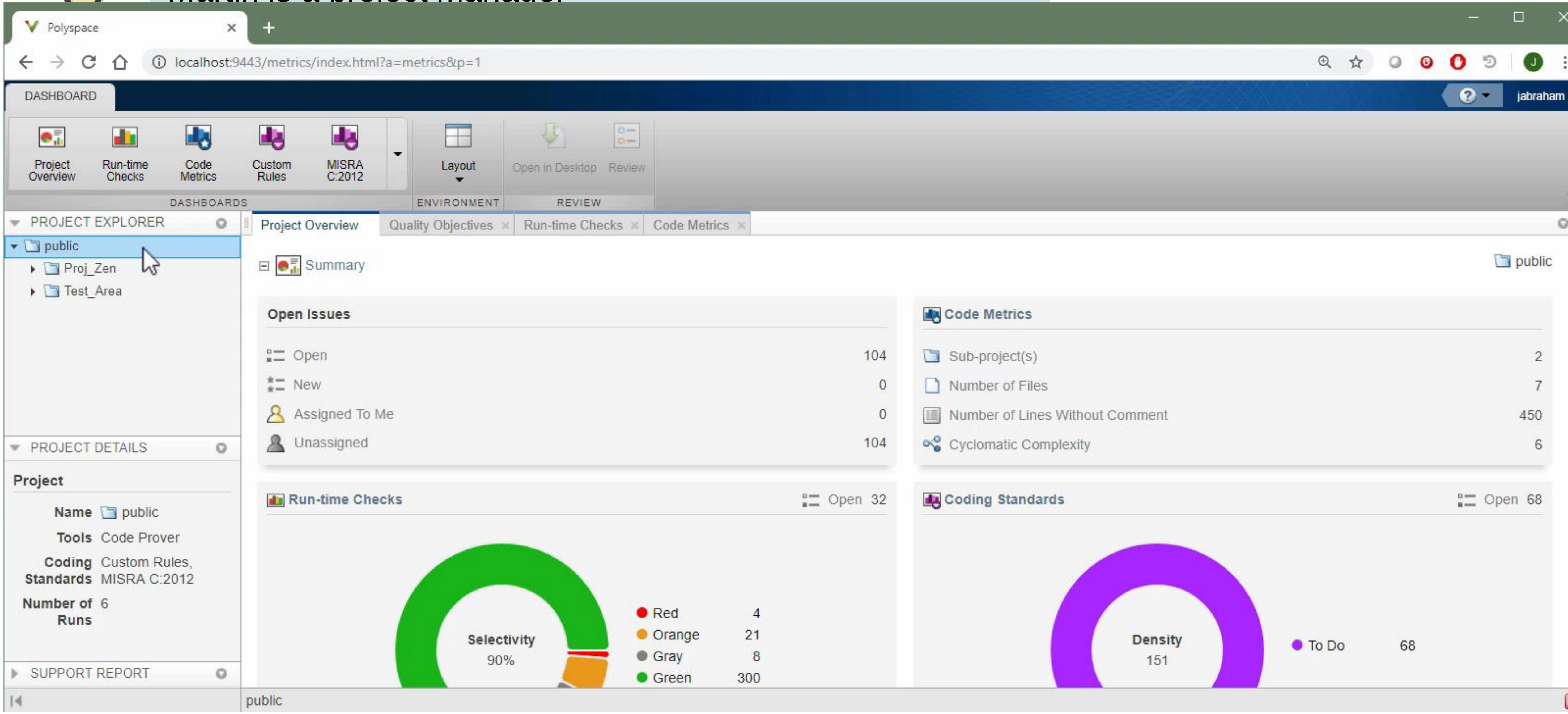
Created: 2 hours ago
Updated: 2 hours ago

Agile

View on Board

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

Martin is a project manager



Project

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

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

Selectivity 90%

● Red	4
● Orange	21
● Gray	8
● Green	300

Coding Standards Open 68

Density 151

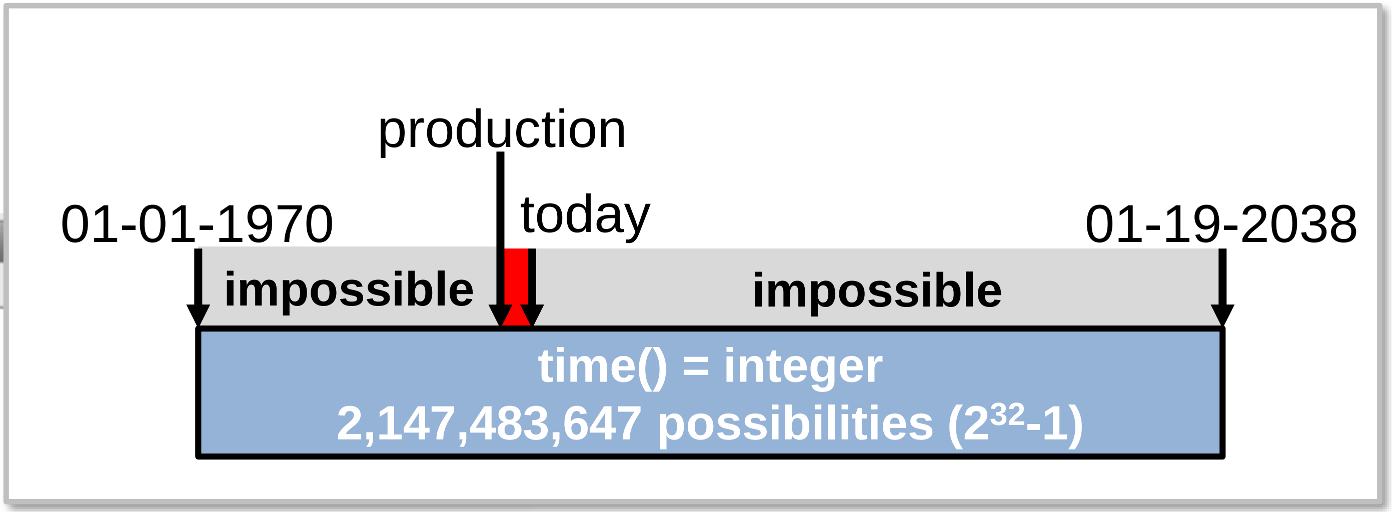
● To Do	68

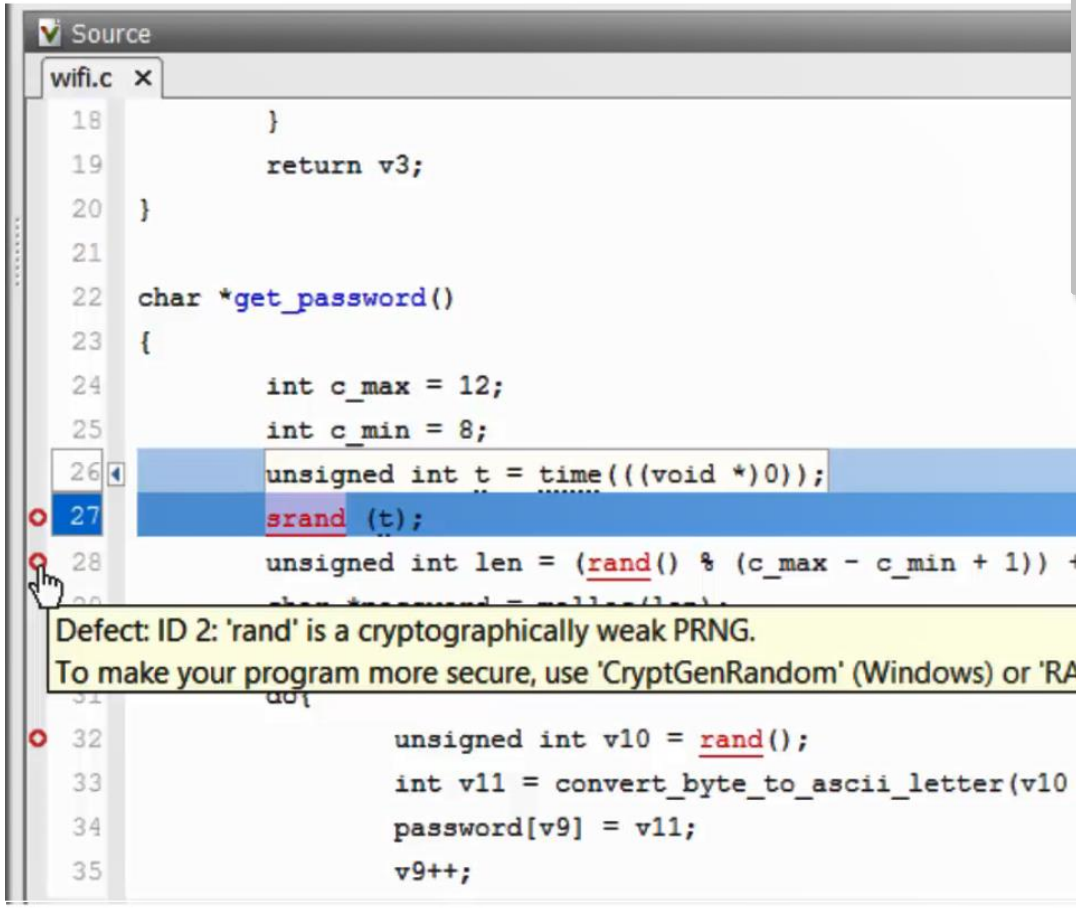
public

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.

Finally.... Jeep Hack: Deterministic Random Number Generator





```
        }
        return v3;
}

char *get_password()
{
        int c_max = 12;
        int c_min = 8;
        unsigned int t = time(((void *)0));
        srand (t);
        unsigned int len = (rand() % (c_max - c_min + 1)) + c_min;
        char *password = malloc(len);
        do{
                unsigned int v10 = rand();
                int v11 = convert_byte_to_ascii_letter(v10 % 62);
                password[v9] = v11;
                v9++;
```

Defect: ID 2: 'rand' is a cryptographically weak PRNG.
To make your program more secure, use 'CryptGenRandom' (Windows) or 'RAND_bytes' (OpenSSL) instead.

Miller (left) and Valasek demonstrated the rest of their attacks on the Jeep while I drove it around an empty parking lot. WHITNEY CURTIS FOR WIRED

End

Backup

New Polyspace Products in R2019a

1. Products for web browser results access
   - ***Polyspace Bug Finder Access*** and ***Polyspace Code Prover Access***
   - Web-browser based review of static code analysis results
   - Integration with Jira

2. Products for servers
   - ***Polyspace Bug Finder Server*** and ***Polyspace Code Prover Server***
   - Support for Continuous Integration systems such as Jenkins

3. Products for desktop use
   - ***Polyspace Bug Finder*** and ***Polyspace Code Prover***
   - Find bugs and run time errors before submitting code to repository

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

Safety

Standards:

- DO-178 (aero)
- ISO 26262 (auto)
- IEC 61508 (industrial)
- IEC 62304 (medical)
- EN 50128 (rail)
- MISRA
- AUTOSAR

Security

Standards:

- CERT-C
- CWE
- ISO 17961
- MISRA-C:2012 Appendix 1
- Tainted data tracking

Reliability and **Robustness**

Code Proving

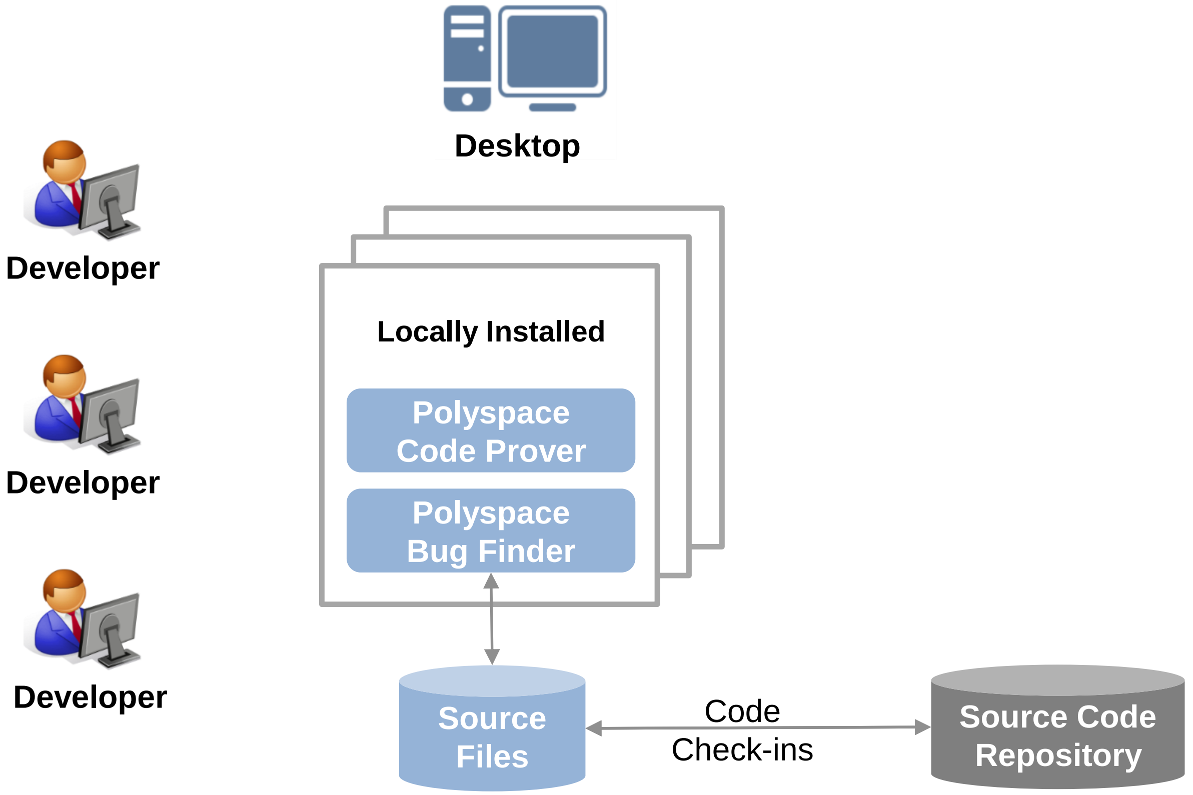
- Prove absence of critical runtime errors (or find even the slightest vulnerability)
- Exhaustive: all possible inputs, control flows, data flows (no instrumentation, execution, test cases)
- Sound: no false negatives

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

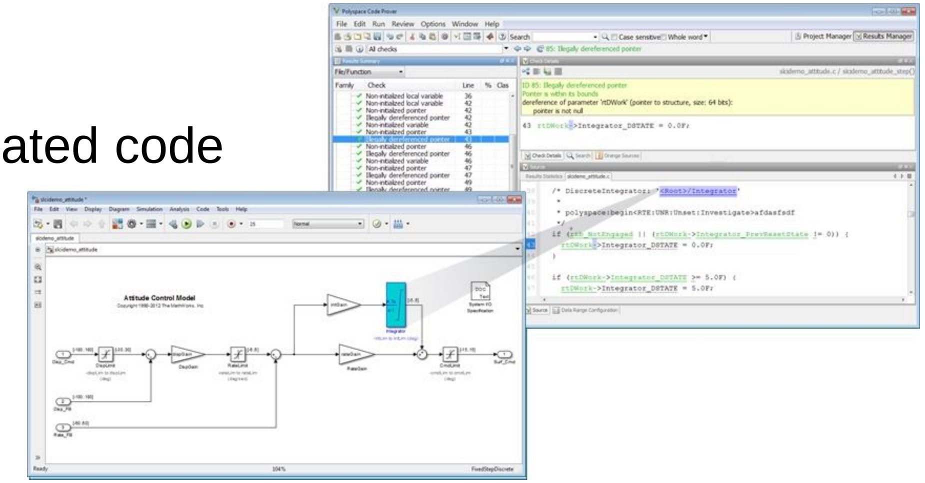
Optional Workflow: Analyze and Verify Code Prior to Check-In

- Run Polyspace Bug Finder and Polyspace Code Prover interactively
- Analyze code before it gets checked into the source code repository



When To Use Polyspace

- Checking generated code
  - Integrated code may consist of handwritten code + generated code
  - For certification, check coding standards (MISRA, CERT)
  - For AUTOSAR, prove interface requirements are met

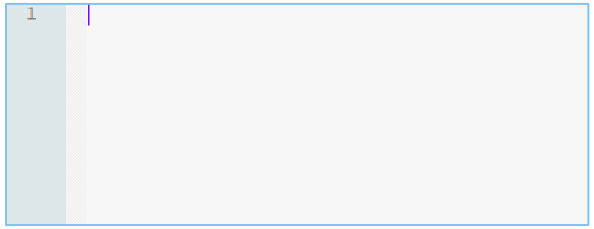


ID 85: Illegally dereferenced pointer
Pointer is within its bounds
dereference of parameter 'rtDWork' (pointer to structure, size: 64 bits):
pointer is not null

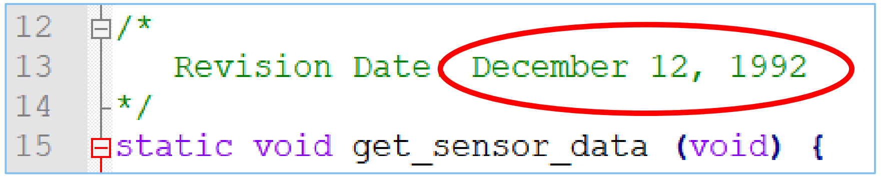
```
43 rtDWork->Integrator_DSTATE = 0.0F;
```

```
/* DiscreteIntegrator: '<Root>/Integrator'
 *
 * polyspace:begin<RTE:UNR:Unset:Investigate>afdasfsdf
 */
if (rtb_NotEngaged || (rtDWork->Integrator_PrevResetState != 0)) {
  rtDWork->Integrator_DSTATE = 0.0F;
}

if (rtDWork->Integrator_DSTATE >= 5.0F) {
  rtDWork->Integrator_DSTATE = 5.0F;
```

- Check new code as soon as it written
  - Find issues early, when it is easier and cheaper to fix



- For heritage or legacy code
  - Fix issues when modifications to code are made
  - Create a baseline, only review new findings
  - Justify findings you don’t wish to fix or review again



```
/*
   Revision Date  December 12, 1992
*/
static void get_sensor_data (void) {
```

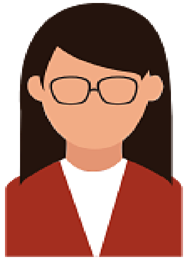
Abstract

Do you need evidence that your code will not cause safety hazards or security issues? Polyspace products allow you to achieve the highest levels of software quality with reduced testing effort. Using formal methods based static code analysis, it can prove that your code is free from certain critical run-time errors. The analysis can be done interactively by software developers during code development to quickly find coding defects and violations of safety and security standards like MISRA, CERT-C/C++. When used with Continuous Integration tools such as Jenkins, Polyspace helps improve software quality, safety, and security across your projects. Results are published for web-browser based code review with tracing information to identify the root cause of defects. Polyspace supports modern team collaboration dashboards to show quality metrics for project and safety managers. Integration with defect tracking tools such as Jira help manage issues across your development enterprise.

Outline

- Static Analysis Concepts
  - Why is it important, what is it
  - Relevance to Auto, Aero, Med, IAM industries
- Polyspace Static Analysis
  - Proving absence of run-time errors
  - Polyspace products
  - Customer references (values and benefits)
- Team Collaboration with Polyspace
  - Workflow overview with new products
  - Build automation – runs Polyspace on server, sends email notifications
  - Quality Engineer, Team Lead – reviews results, triages and assigns defects
  - Developer – uses PS Access to debug defects, fixes code, does pre-submit checks
  - Project, Quality Manager – monitors trends
  - Pre-submit workflow
- Summary

Workflow for Quality Engineers



- Quin is a Quality Engineer

- She has received an email notification indicating XX new defects have been found in various projects that were analyzed last night

- She clicks on the links in the email to view results of the analysis
- She looks at the Project Overview Dashboard to identify projects and issues to focus on
- She can triage issues and opens Jira tickets from the PS Access web-browser

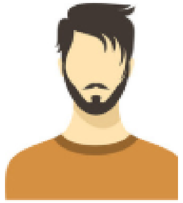
- She notices that code belonging to Dara the developer has dead code in a case statement
- She opens a Jira ticket from within Polyspace Access and assigns defect in Jira to Dara

- Show video of these tasks in Polyspace Access

Software Developer Responding to Issues

- Dara looks at defects assigned to her in Jira
- She clicks on the link the Jira ticket to debug issue via web-browser with Polyspace Access
- She notices that priorities can be set, annotations can be provided to report on status, all from within the web-browser interface of Polyspace Access
- She uses the information provided by the tool (result details and contextual help) to formulate a fix for the defect
- Dara fixes the code to address the unreachable case statement and checks it in
- Show video of developer performing these tasks with Polyspace Access and in the code editor to fix the defect

Workflow for Project Manager



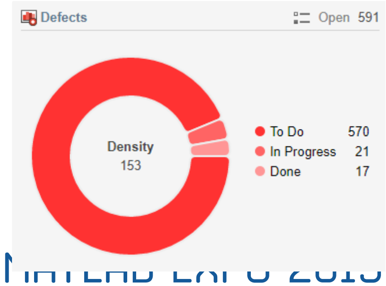
- Doug is a project manager with responsibility for software quality

- He monitors overall project status via web-browser dashboard
- He checks SQO levels and compliance to standards (MISRA, CERT)
- He also can see that the defect that Dara fixed has been confirmed to be fixed in the last analysis run that was initiated by Jenkins

- Show screenshots or short video of these tasks

Workflow for Developers

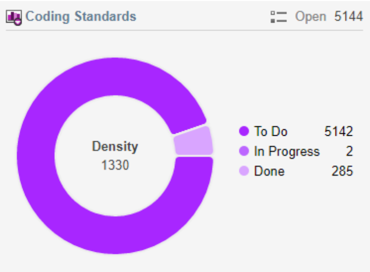
- Dara is a software developer
- She is tasked with adding a new feature which requires changing the behavior of a function that has a case statement
- She makes the code change, then runs her unit tests, which all pass, then checks the code into the source code repository
- Show short video of code edits and command line execution of unit tests



Defects Open 591

Density 153

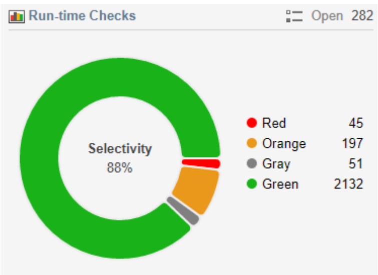
● To Do	570
● In Progress	21
● Done	17



Coding Standards Open 5144

Density 1330

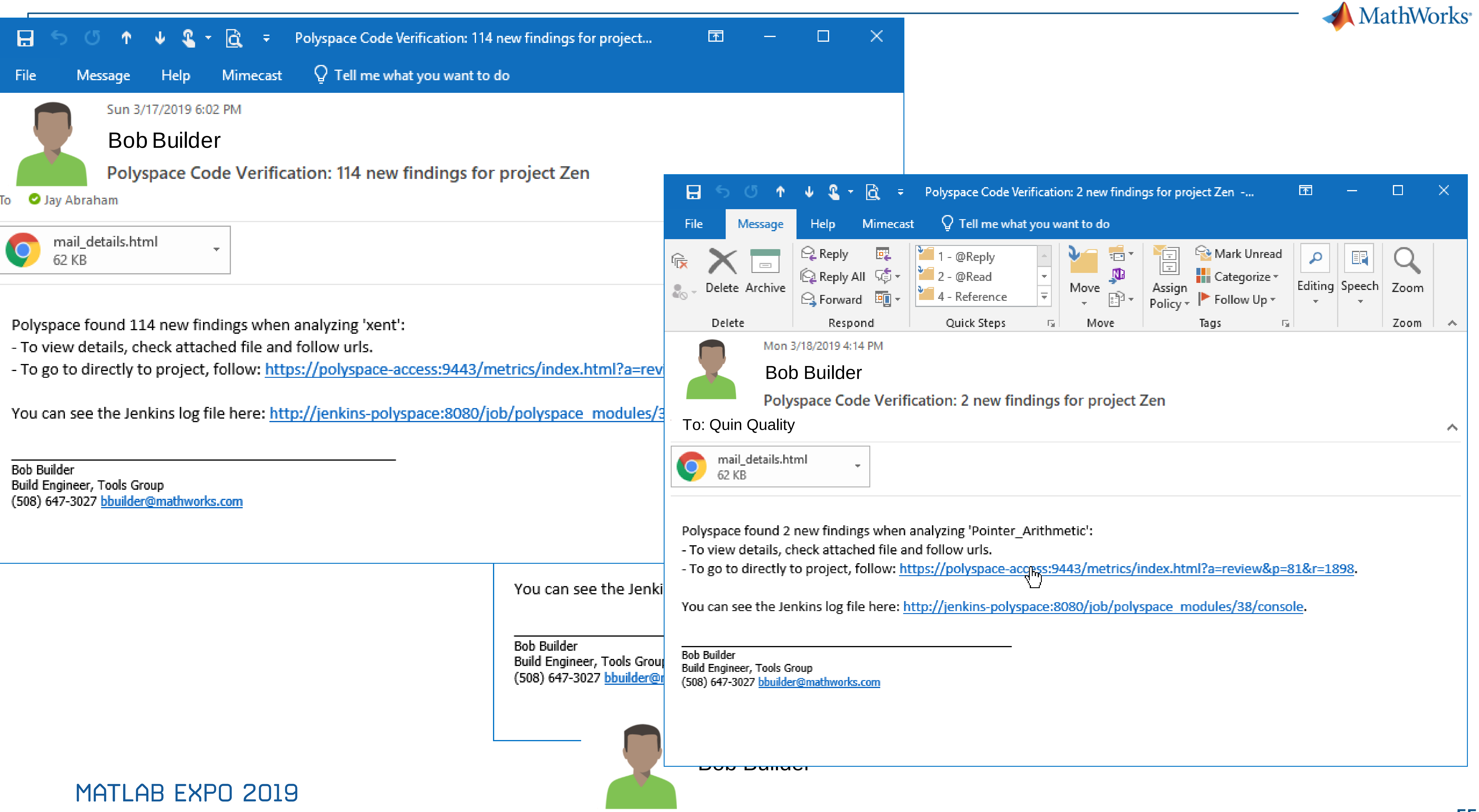
● To Do	5142
● In Progress	2
● Done	285



Run-time Checks Open 282

Selectivity 88%

● Red	45
● Orange	197
● Gray	51
● Green	2132


Polyspace Code Verification: 114 new findings for project...
File
Message
Help
Mimecast
Tell me what you want to do

Sun 3/17/2019 6:02 PM

Bob Builder

Polyspace Code Verification: 114 new findings for project Zen

To Jay Abraham

mail_details.html
62 KB

Polyspace found 114 new findings when analyzing 'xent':
- To view details, check attached file and follow urls.
- To go to directly to project, follow: https://polyspace-access:9443/metrics/index.html?a=rev

You can see the Jenkins log file here: http://jenkins-polyspace:8080/job/polyspace_modules/3

Bob Builder
Build Engineer, Tools Group
(508) 647-3027 bbuilder@mathworks.com

You can see the Jenki

Bob Builder
Build Engineer, Tools Grou
(508) 647-3027 bbuilder@r

Mon 3/18/2019 4:14 PM

Bob Builder

Polyspace Code Verification: 2 new findings for project Zen

To: Quin Quality

mail_details.html
62 KB

Polyspace found 2 new findings when analyzing 'Pointer_Arithmetic':
- To view details, check attached file and follow urls.
- To go to directly to project, follow: https://polyspace-access:9443/metrics/index.html?a=review&p=81&r=1898.

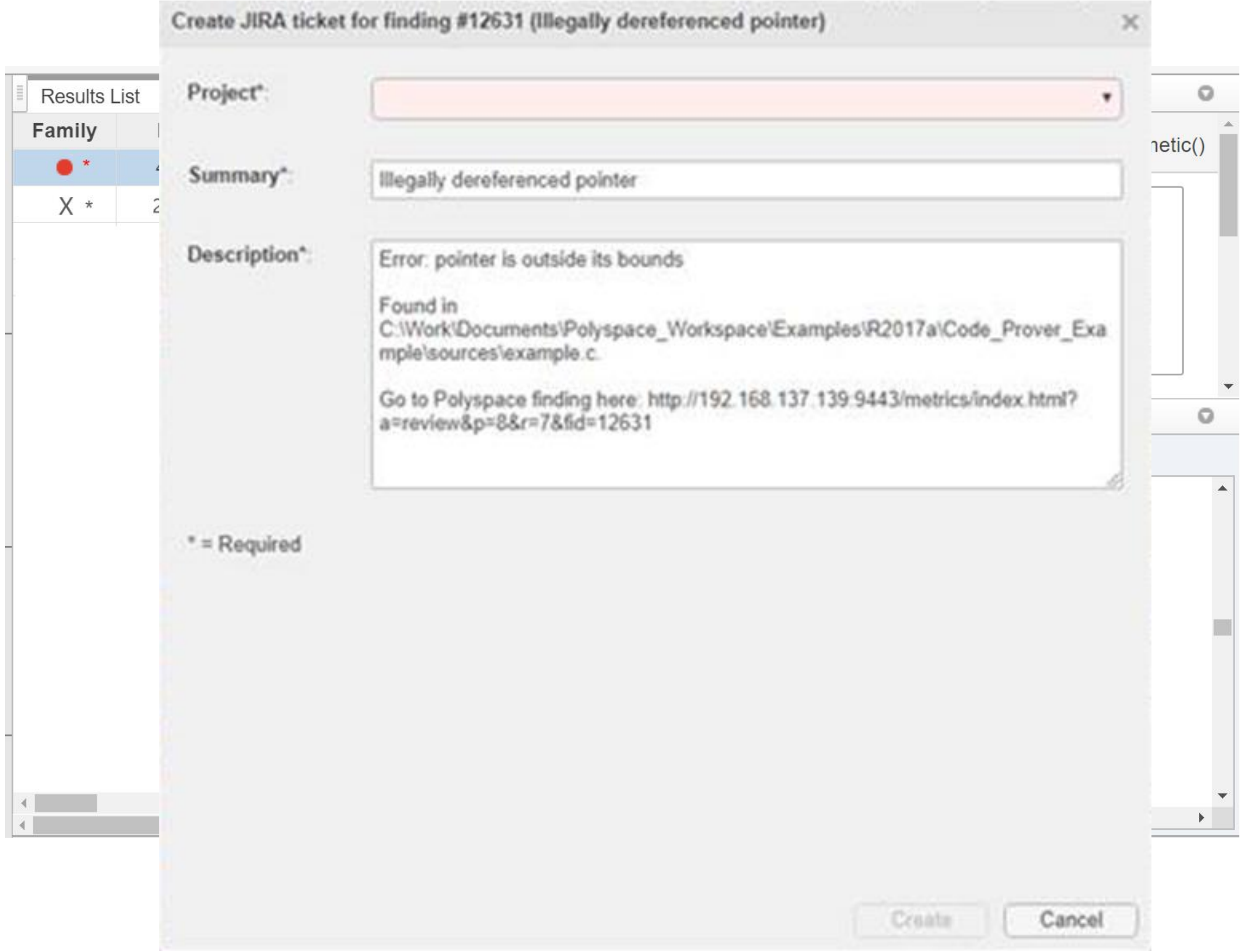
You can see the Jenkins log file here: http://jenkins-polyspace:8080/job/polyspace_modules/38/console.

Bob Builder
Build Engineer, Tools Group
(508) 647-3027 bbuilder@mathworks.com

Bob Builder

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

- She has received an email notification from indicating 2 new findings were found in her project
- She click on the link in the email to view new findings in Polyspace Web UI
- The results list shows 2 findings that are in Dara’s code
- She opens two Jira tickets and assigns them to Dara



Create JIRA ticket for finding #12631 (Illegally dereferenced pointer)

Project*:

Summary*: Illegally dereferenced pointer

Description*:

Error: pointer is outside its bounds

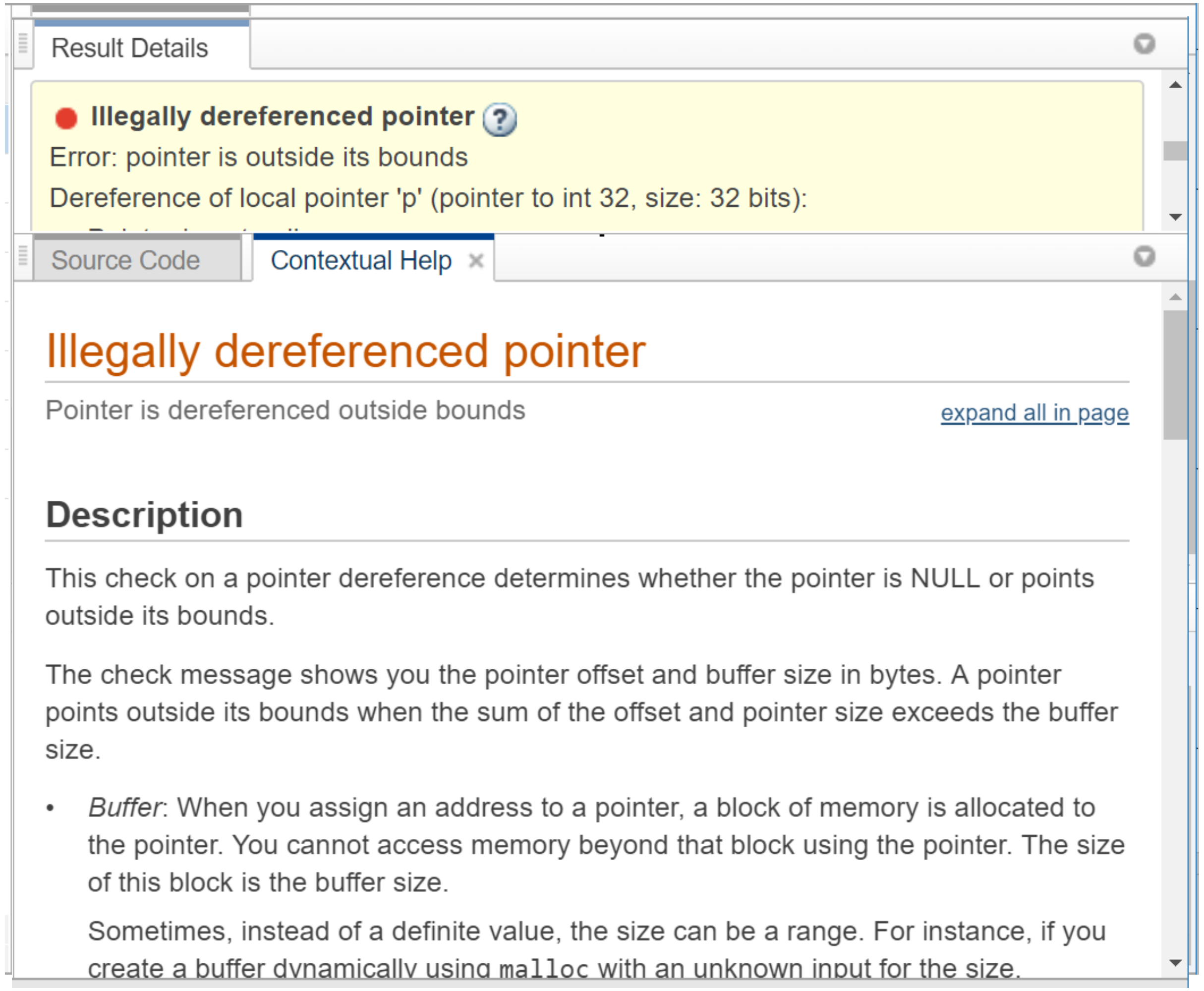
Found in C:\Work\Documents\Polyspace_Workspace\Examples\R2017a\Code_Prover_Example\sources\example.c

Go to Polyspace finding here: http://192.168.137.139:9443/metrics/index.html?a=review&p=8&r=7&fid=12631

* = Required

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

- She opens the first JIRA ticket and clicks the Polyspace Access link
- She uses the information provided by the tool (result details and contextual help) to formulate a fix for the defect
- She fixes the defect in her IDE and check-in the changes



● **Illegally dereferenced pointer** ?

Error: pointer is outside its bounds

Dereference of local pointer 'p' (pointer to int 32, size: 32 bits):

Illegally dereferenced pointer

Pointer is dereferenced outside bounds

Description

This check on a pointer dereference determines whether the pointer is NULL or points outside its bounds.

The check message shows you the pointer offset and buffer size in bytes. A pointer points outside its bounds when the sum of the offset and pointer size exceeds the buffer size.

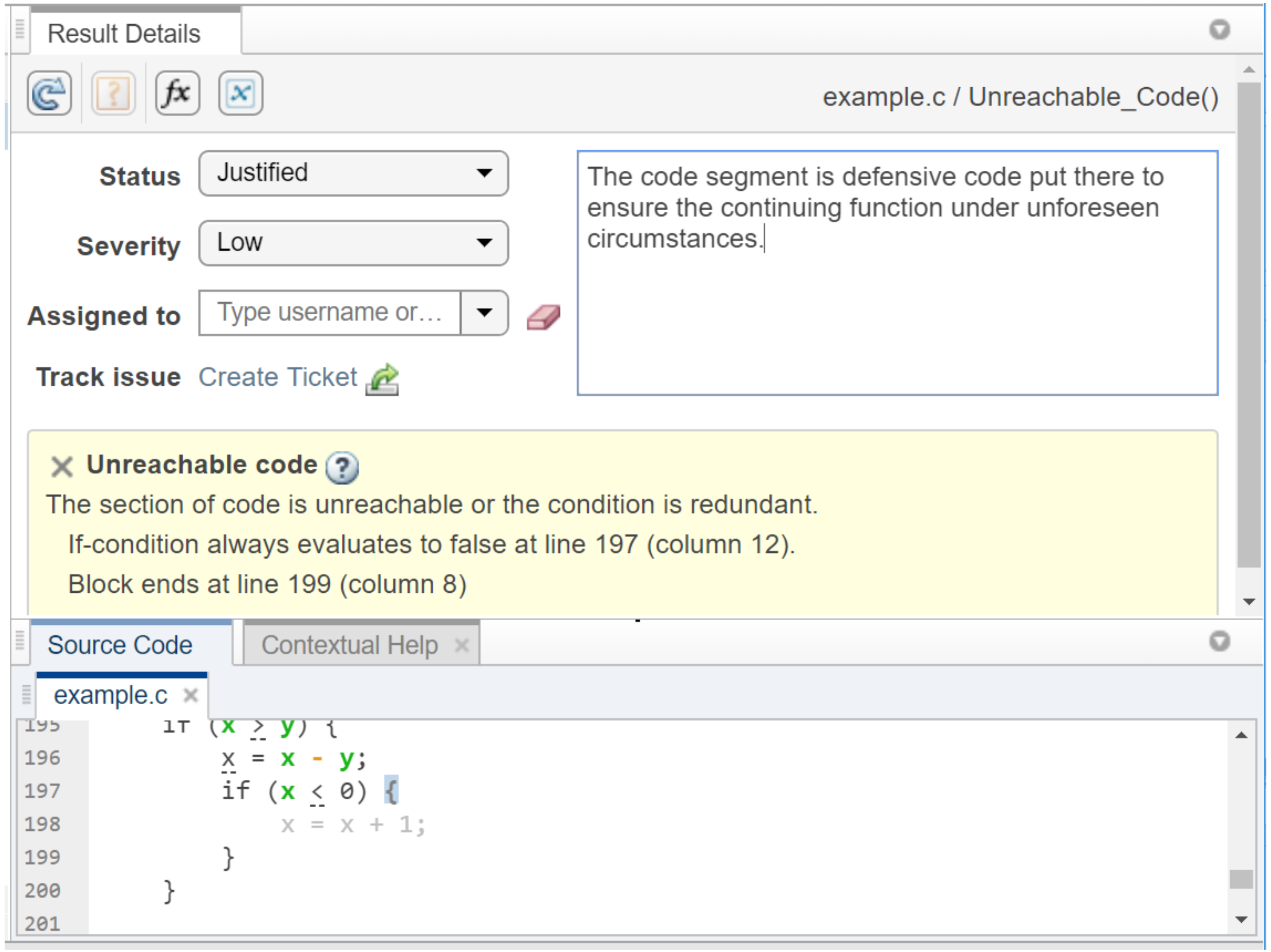
- *Buffer*: When you assign an address to a pointer, a block of memory is allocated to the pointer. You cannot access memory beyond that block using the pointer. The size of this block is the buffer size.

  Sometimes, instead of a definite value, the size can be a range. For instance, if you create a buffer dynamically using `malloc` with an unknown input for the size.

Dara is a software developer

She is responsible for writing code and fixing defects

- She opens the second JIRA ticket and clicks the Polyspace Access link
- She determines that no code changes are required
- She changes the status to justified
- She writes a comment to explain her reasoning



Status: Justified

Severity: Low

Assigned to: Type username or…

Track issue: Create Ticket

The code segment is defensive code put there to ensure the continuing function under unforeseen circumstances.

Unreachable code

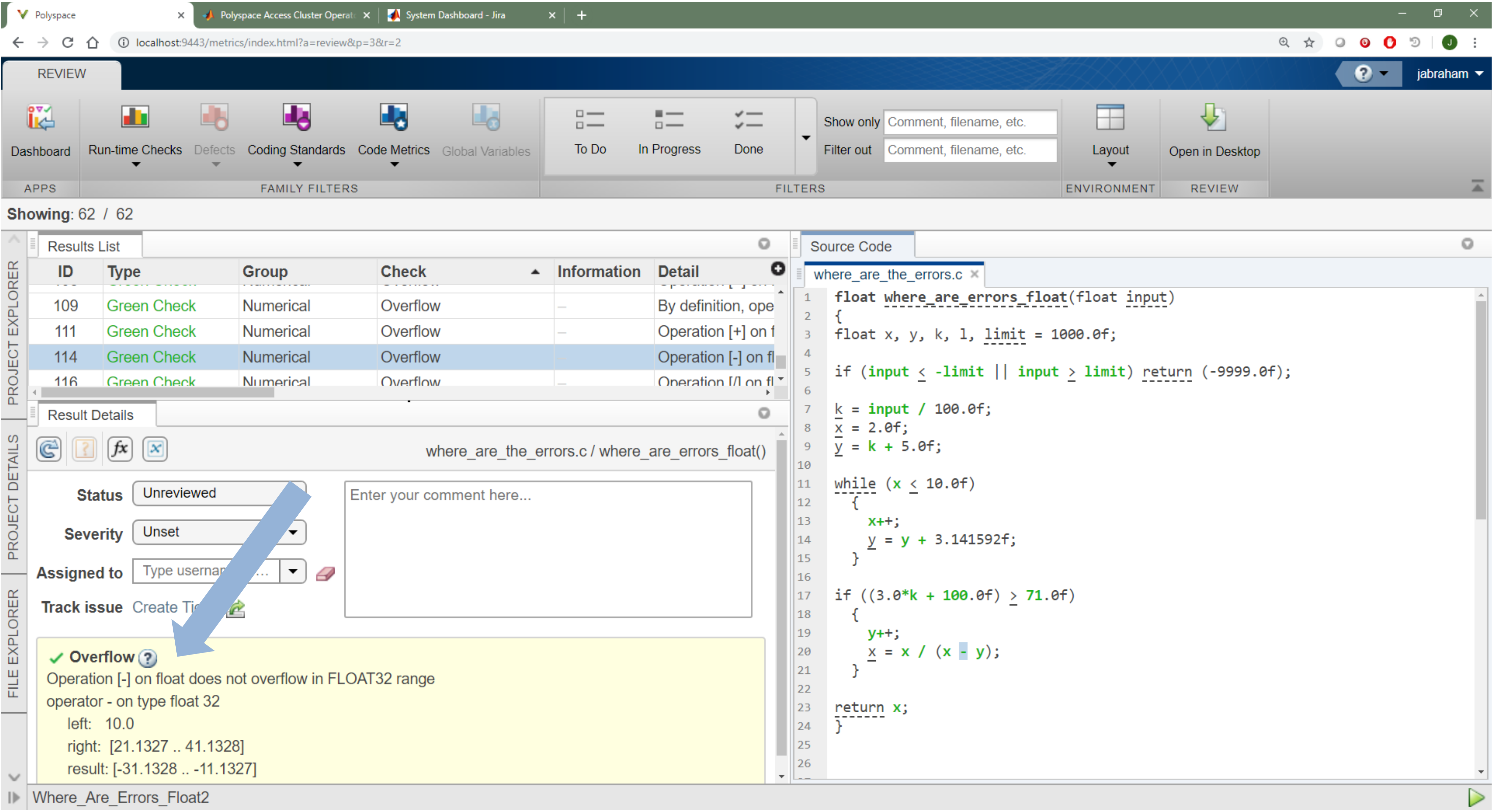
The section of code is unreachable or the condition is redundant.

If-condition always evaluates to false at line 197 (column 12).

Block ends at line 199 (column 8)

```
195         if (x > y) {
196             x = x - y;
197             if (x < 0) {
198                 x = x + 1;
199             }
200         }
201
```

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



Showing: 62 / 62

Results List

ID	Type	Group	Check	Information	Detail
109	Green Check	Numerical	Overflow	–	By definition, ope
111	Green Check	Numerical	Overflow	–	Operation [+] on f
114	Green Check	Numerical	Overflow	–	Operation [-] on fl
116	Green Check	Numerical	Overflow	–	Operation [/] on fl

Result Details

where_are_the_errors.c / where_are_errors_float()

Status: Unreviewed

Severity: Unset

Assigned to: Type usernam…

Track issue: Create Ti…

Enter your comment here...

✓ Overflow ?

Operation [-] on float does not overflow in FLOAT32 range

operator - on type float 32

left: 10.0

right: [21.1327 .. 41.1328]

result: [-31.1328 .. -11.1327]

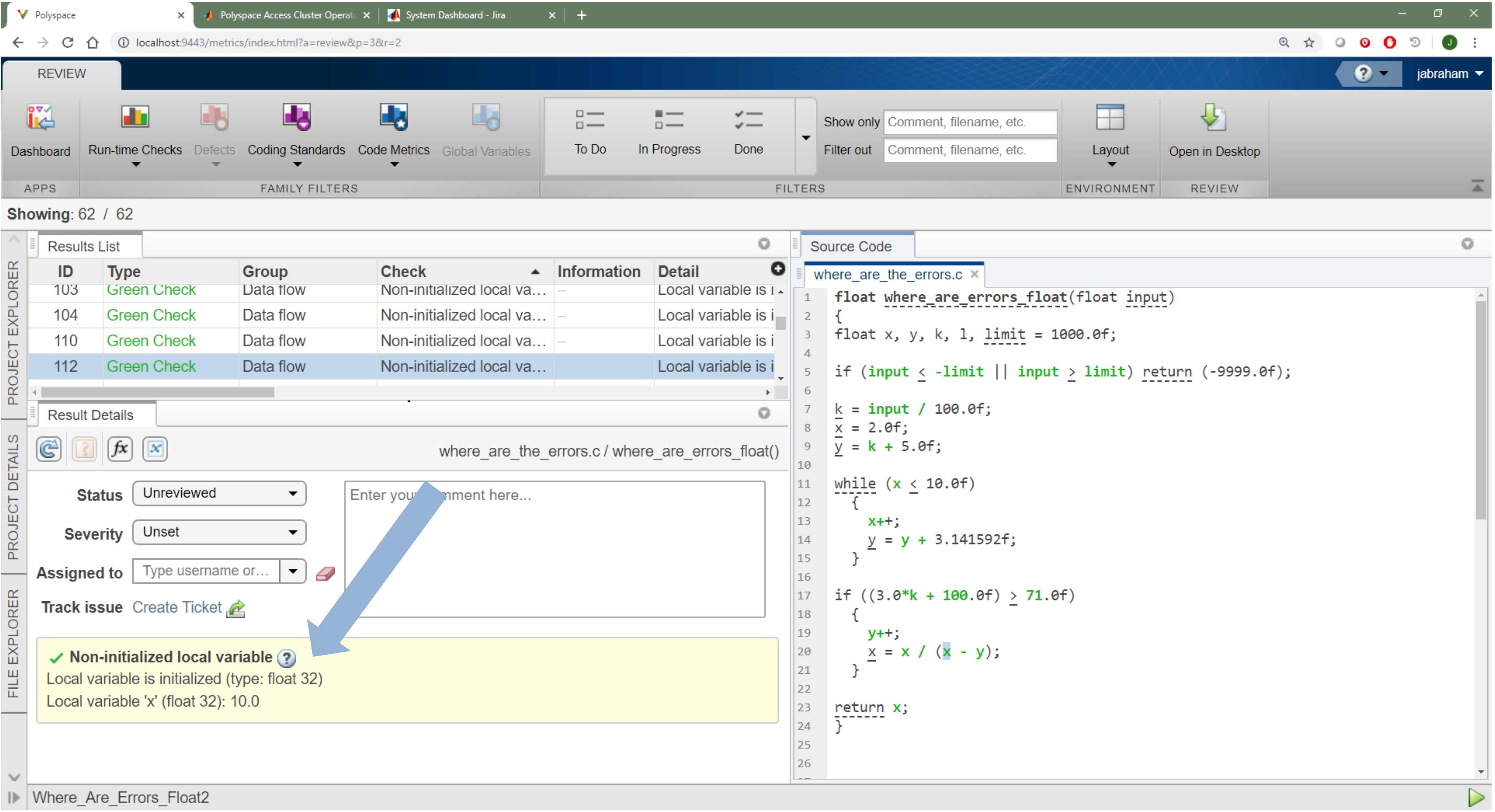
Source Code

where_are_the_errors.c

```
float where_are_errors_float(float input)
{
float x, y, k, l, limit = 1000.0f;

if (input < -limit || input > limit) return (-9999.0f);

k = input / 100.0f;
x = 2.0f;
y = k + 5.0f;

while (x < 10.0f)
  {
    x++;
    y = y + 3.141592f;
  }

if ((3.0*k + 100.0f) > 71.0f)
  {
    y++;
    x = x / (x - y);
  }

return x;
}
```

Where_Are_Errors_Float2

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



Showing: 62 / 62

ID	Type	Group	Check	Information	Detail
103	Green Check	Data flow	Non-initialized local va...	–	Local variable is i
104	Green Check	Data flow	Non-initialized local va...	–	Local variable is i
110	Green Check	Data flow	Non-initialized local va...	–	Local variable is i
112	Green Check	Data flow	Non-initialized local va...	–	Local variable is i

where_are_the_errors.c / where_are_errors_float()

Status: Unreviewed

Severity: Unset

Assigned to: Type username or...

Track issue: Create Ticket

Enter your comment here...

✓ **Non-initialized local variable** ?

Local variable is initialized (type: float 32)

Local variable 'x' (float 32): 10.0

where_are_the_errors.c

```
float where_are_errors_float(float input)
{
float x, y, k, l, limit = 1000.0f;

if (input < -limit || input > limit) return (-9999.0f);

k = input / 100.0f;
x = 2.0f;
y = k + 5.0f;

while (x < 10.0f)
  {
    x++;
    y = y + 3.141592f;
  }

if ((3.0*k + 100.0f) > 71.0f)
  {
    y++;
    x = x / (x - y);
  }

return x;
}
```

Where_Are_Errors_Float2