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?

```c
int is_abs(int x, int y) {
    if (x >= 0) {
        if (x == y)
            return 1;
    }
    else {
        if (x == -y)
            return -1;
    }
    return 0;
}
```
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

Dijkstra, “Notes on Structured Programming” (1972)
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

```matlab
float where_are_errors(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)
    {
        x = x / (x - y);
    }
    return x;
}
```

- How many run-time errors are possible?
  1. Divide by zero
  2. Overflow
  3. Uninitialized variables
Proving Absence of Critical Run-Time Errors

```c
float where_are_errors(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 ((3.0f * k + 100.0f) >= 71.0f) {
        y++;
        x = x / (x - y);
    }
    return x;
}
```

What Polyspace infers:

- \( k \in [-10,10] \)
- \( y = k + 5 \)
- \( y = k + 5 + (x - 2) \times 3.141592 \)
- \( x = 10 \)
- \( k \geq (71 - 100)/3 \) \( y \geq 20.466 \)
Proving Absence of Critical Run-Time Errors

```plaintext
float where_are_errors(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 ((3.0f * k + 100.0f) >= 71.0f)
    {
        y++;
        x = x / (x - y);
    }

    return x;
}
```

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