Introducing Polyspace into the Software Development Process

Eileen Davidson
Ford Motor Company

12-May-2015

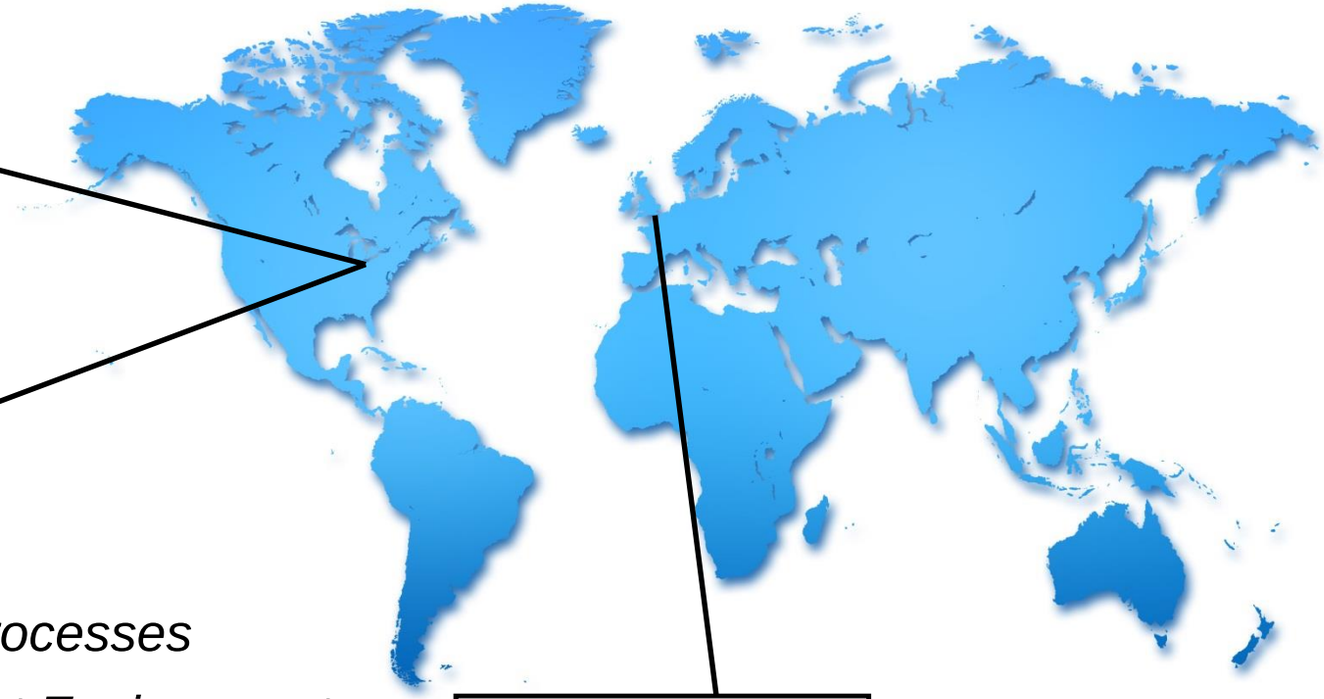
Overview

- Why Ford Powertrain introduced Polyspace into the software development process
- Advantages of Polyspace over other tools
- Quick examples of useful Polyspace Features
- Best Practices for introducing a new tool into your process

Ford Powertrain Controls. Calibration and NVH (PCCN)

Responsible for developing Software for Gasoline/Engine/Driveline Controls Worldwide

200+ Software Developers Worldwide



Gasoline Engine Controls
Dearborn, MI

Transmission and Driveline Controls
Livonia, MI

Diesel Engine Controls
Basildon, Essex, UK

Hand Code + Model Based Processes

Windows + Linux Development Environments

Previous State of Ford Powertrain Software Development Process

- Static Analysis was run automatically on both the individual feature level and the application level as a part of the “make” process
  - ***Tool was not Polyspace, but another static analysis tool***
- Although checks were run, the output of the tool was suboptimal
  - Report was a flat text file
  - Errors are not grouped, sorted by type of error, so difficult for a human to parse
  - Reports only referenced file name and line number
  - For model based features, no direct link back to model

Previous State of Ford Powertrain Software Development Process (continued)

The tool provided no interface to justify inconsequential errors. A separate reporting mechanism was required

Example:

```
#define MODEA 1U
extern unit8 cold_mode_global;

void determine_cold_mode(void)
{
        auto uint8 cold_mode_tmp = 0U;

        cold_mode_temp = cold_mode_global;
        if ( cold_mode_tmp = MODEA )
        {
                .....
```

← *written to by an external process*

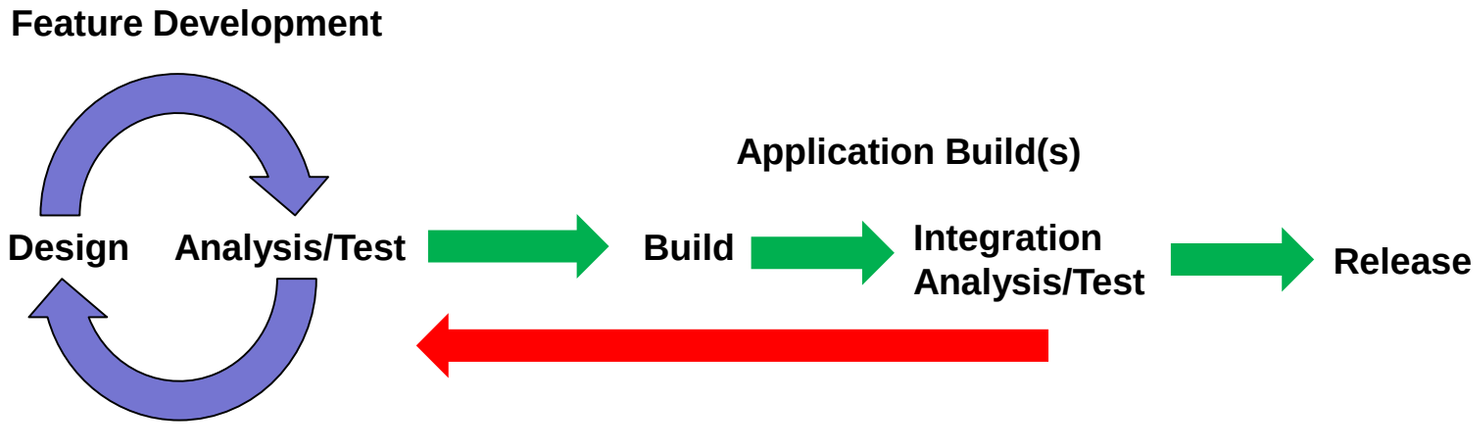
The tool assumes the signal cold_mode_global is initialized to zero and doesn't change

If the range of the input could be specified, the warning would be suppressed from the final report

This codes flags the warning -→ 'IF' always evaluates to False

As a result, many real errors were slipping through the process at the feature level

- A number of static analysis errors that could have been fixed at the feature level made their way to the application level
- Application engineers now have to “chase” multiple feature engineers to get the errors corrected
- Ideally, most errors should be found as early as possible (prior to final application build)

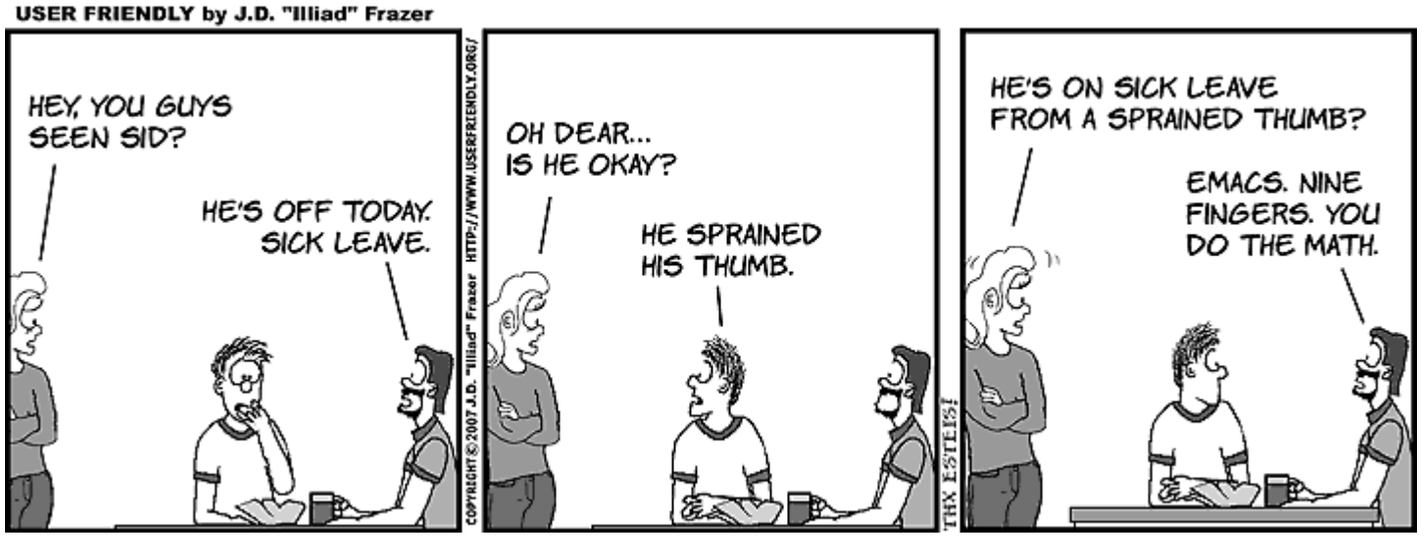


Four Key Criteria for a New Tool

1) Must have a “great” user interface

Graphical user interface that ties errors to actual code

Errors should be able to be organized in such a way it is easy for a human to understand



Four Key Criteria for a New Tool (continued)

2) Automatic Report Generation

- Report must be able to be annotated by the user to provide extra information, such as justifications for errors that are not fixed

3) Easy Integration into the existing Software Development Process

- Must accommodate both MBD and hand code developers
- Must accommodate multiple development environments (LINUX/Windows)

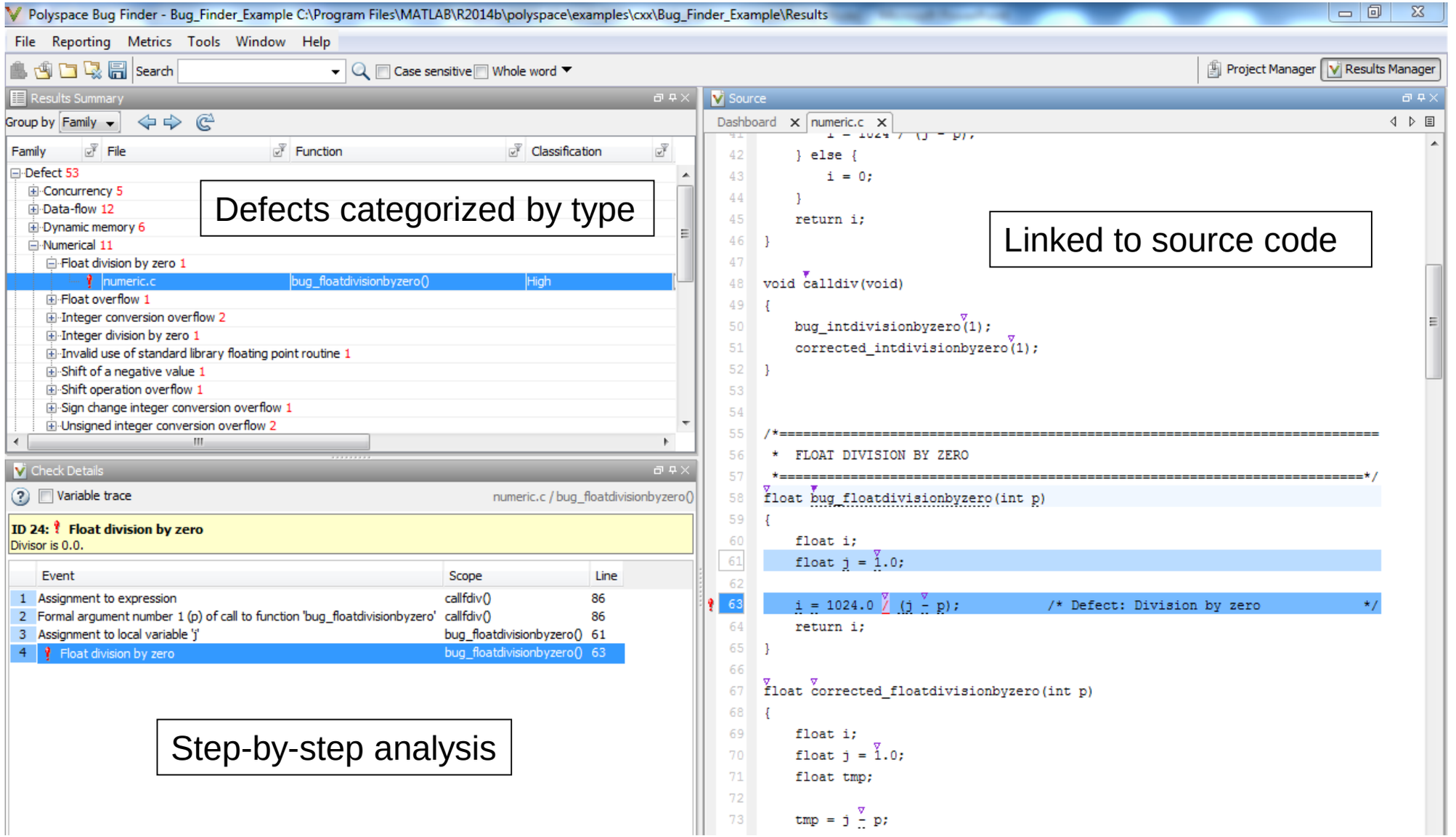
4) Must be “fast”

- Feature engineers iterate their code many times over the course of a normal development process. The tool must not take a long time to startup and run (< 15 minutes)
- And…tools must be run “iteratively” as engineers find/fix errors.

How did Polyspace stack up against our criteria ?

1) Excellent User Interface

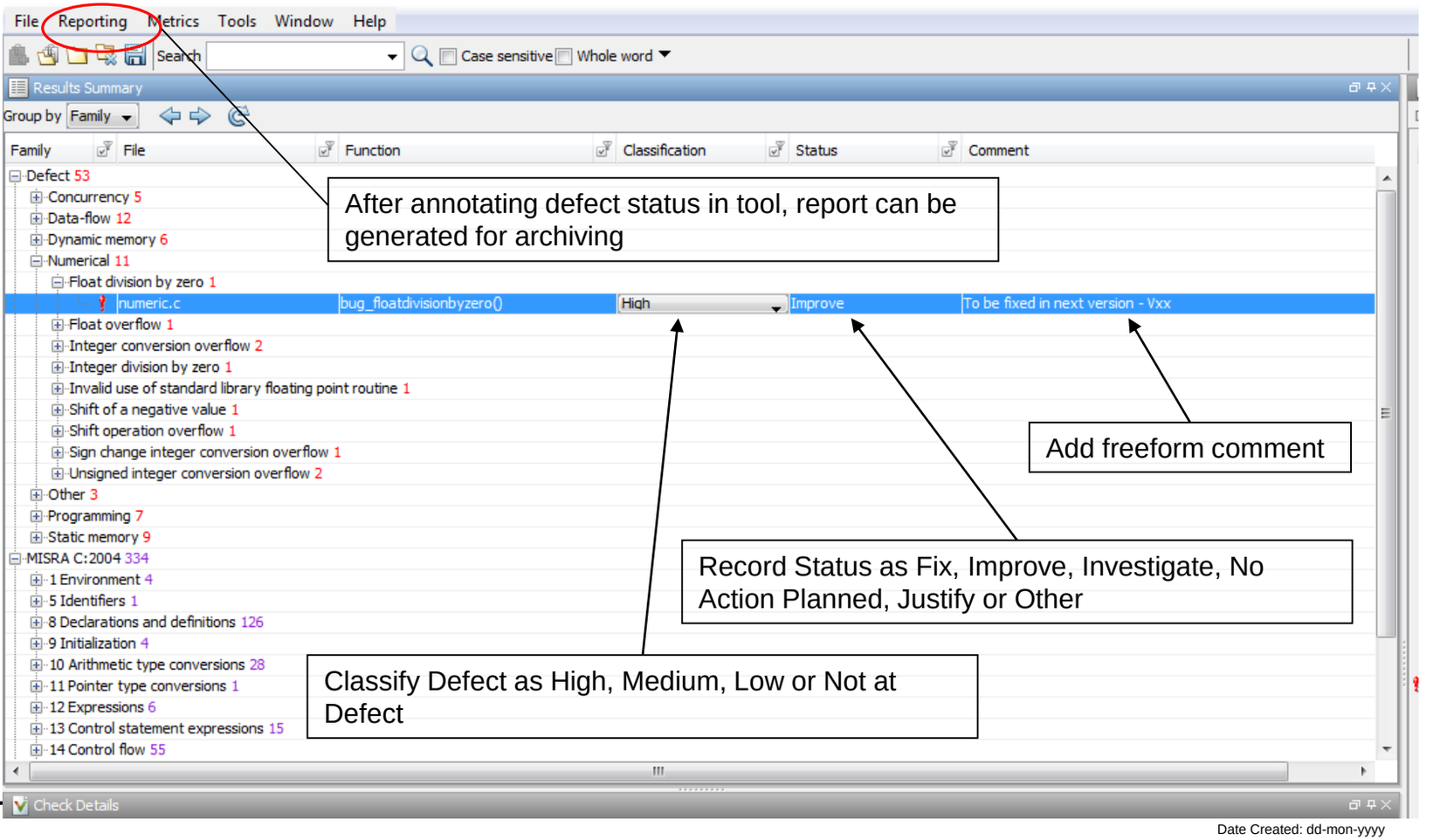
- Polyspace has a graphical display
- Sorts Errors by frequency and type
- Links errors to source code in the display – can see exactly where the error was detected
- Links errors back to model for MBD feature
- Provides step-by-step analysis of how the analysis was performed to get to the error state
- Errors should be able to be organized in such a way it is easy for a human to understand
- Able to configure interfaces with DRS to eliminate erroneous errors caused by lack of visibility into entire application


Polyspace Bug Finder - Bug_Finder_Example C:\Program Files\MATLAB\R2014b\polyspace\examples\cxx\Bug_Finder_Example\Results
File Reporting Metrics Tools Window Help
Search
Case sensitive
Whole word
Project Manager
Results Manager
Results Summary
Group by Family
Family
File
Function
Classification
Defect 53
Concurrency 5
Data-flow 12
Dynamic memory 6
Numerical 11
Float division by zero 1
numeric.c
bug_floatdivisionbyzero()
High
Float overflow 1
Integer conversion overflow 2
Integer division by zero 1
Invalid use of standard library floating point routine 1
Shift of a negative value 1
Shift operation overflow 1
Sign change integer conversion overflow 1
Unsigned integer conversion overflow 2
Defects categorized by type
Check Details
Variable trace
numeric.c / bug_floatdivisionbyzero()
ID 24: Float division by zero
Divisor is 0.0.
Event
Scope
Line
1 Assignment to expression callfdiv() 86
2 Formal argument number 1 (p) of call to function 'bug_floatdivisionbyzero' callfdiv() 86
3 Assignment to local variable 'j' bug_floatdivisionbyzero() 61
4 Float division by zero bug_floatdivisionbyzero() 63
Step-by-step analysis
Source
Dashboard
numeric.c
42 } else {
43 i = 0;
44 }
45 return i;
46 }
47
48 void calldiv(void)
49 {
50 bug_intdivisionbyzero(1);
51 corrected_intdivisionbyzero(1);
52 }
53
54
55 /*==========
56 * FLOAT DIVISION BY ZERO
57 *==========*/
58 float bug_floatdivisionbyzero(int p)
59 {
60 float i;
61 float j = 1.0;
62
63 i = 1024.0 / (j - p); /* Defect: Division by zero */
64 return i;
65 }
66
67 float corrected_floatdivisionbyzero(int p)
68 {
69 float i;
70 float j = 1.0;
71 float tmp;
72
73 tmp = j - p;
Linked to source code

How did Polyspace stack up against our criteria ?

2) Automatic Report Generation

- Provides space to annotate individual errors in final report
- Space to show compliance plan if those aren't planned to be fixed in this immediate release



How did Polyspace stack up against our criteria ?

3) Integration into the existing Software Development Process

- This was more difficult that it sounds !
- Most other tools require that you provide your own build environment
  - Need to support multiple build environments worldwide
    - Model Based and Hand code
    - Windows and Linux
  - Other tools were eliminated from our study because we could not meet their requirements to compile the software in all of our build environments
- Polyspace provides its own software build environment
  - No need to tell is how to compile the software
  - Configurable to include compiler specific extensions if required (i.e. gnu)

How did Polyspace stack up against our criteria ?

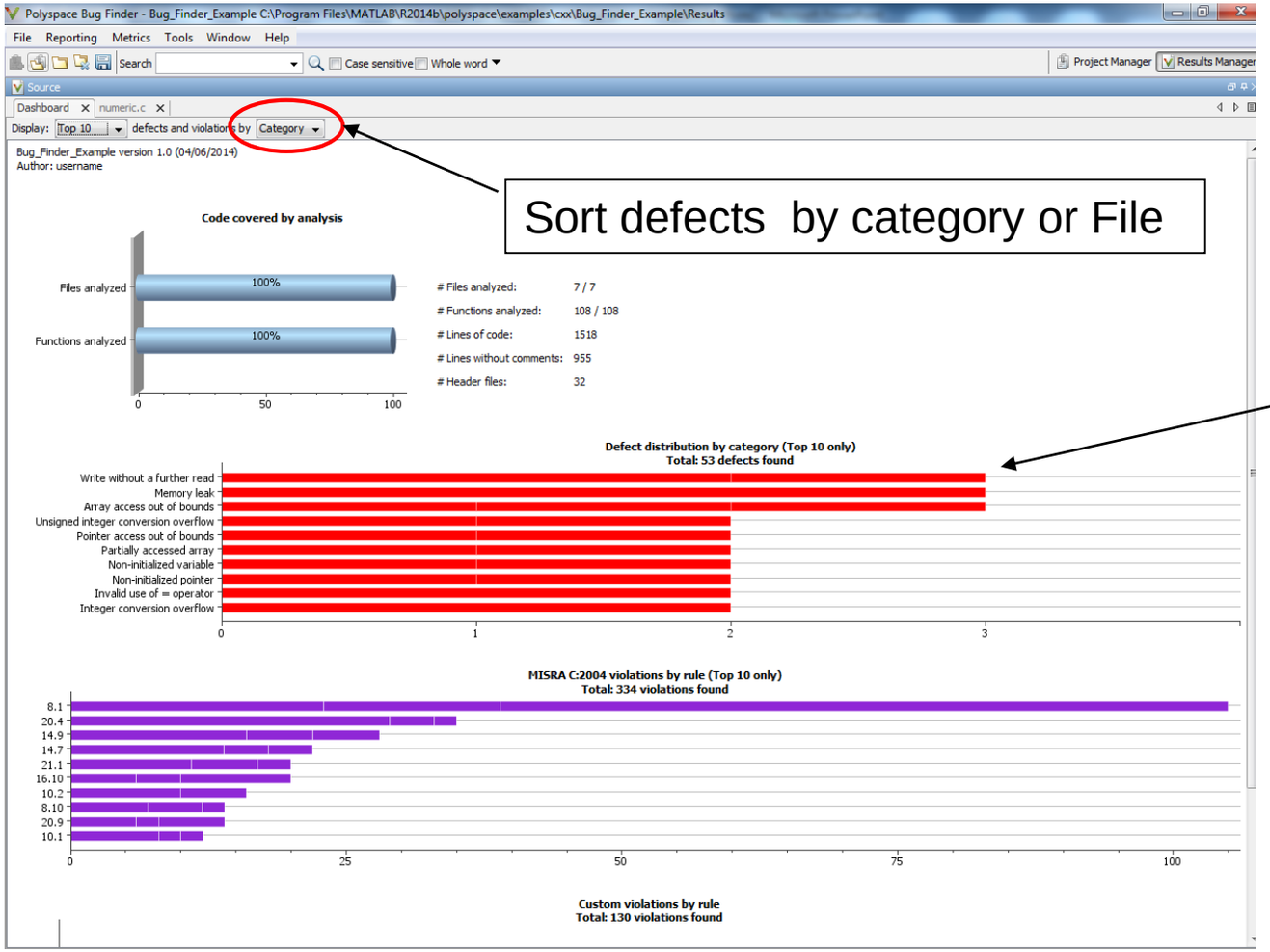
4) Must be “fast”

- We have found the typical runtime for Bugfinder is between 2 and 12 minutes, depending on the complexity of the feature

- Critical for iterative nature of static analysis

Useful Features we have found in Polyspace

Dashboard Display



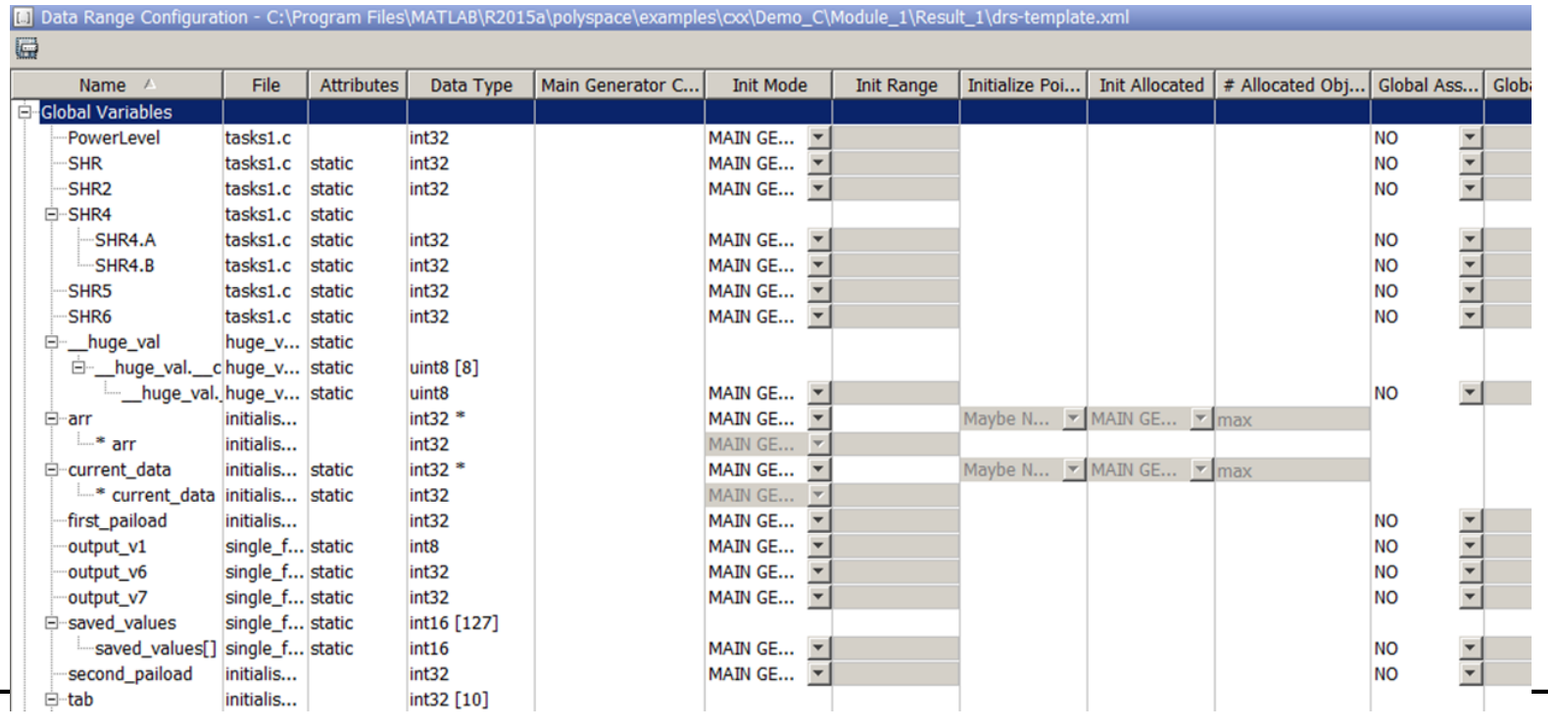
Sort defects by category or File

Top 10 Defects vs. number of occurrences

Useful Features we have found in Polyspace

Ability to configure Range of Values on Input

- Many static analysis tools cannot “mask” inconsequential errors due to incorrect assumptions with regard to input signal ranges
- Helps to clean up final reports – fewer justifications recorded

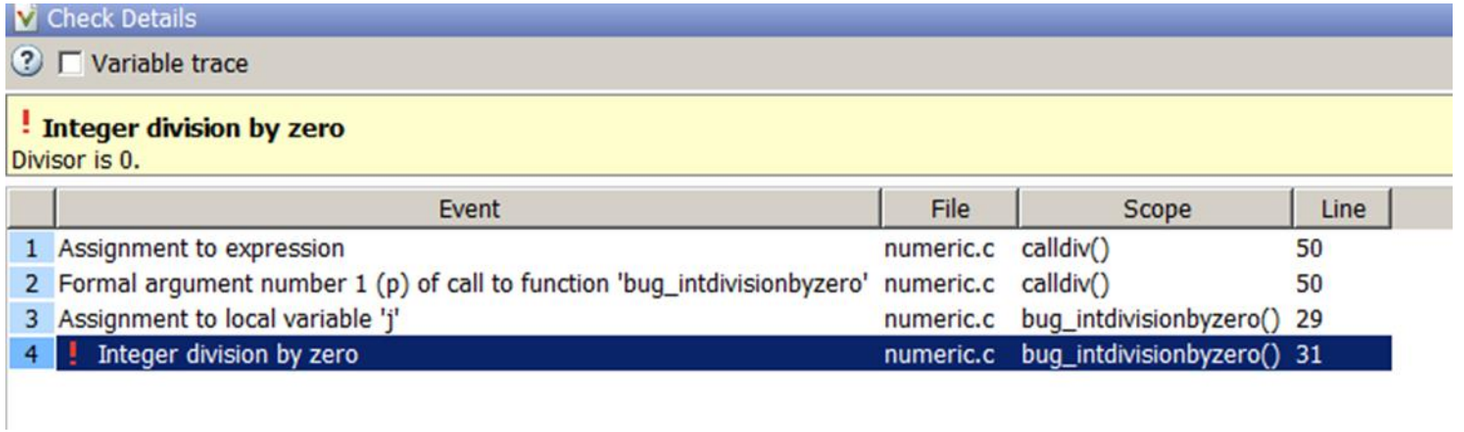


Name	File	Attributes	Data Type	Main Generator C...	Init Mode	Init Range	Initialize Poi...	Init Allocated	# Allocated Obj...	Global Ass...	Glob
Global Variables											
PowerLevel	tasks1.c		int32		MAIN GE...					NO	
SHR	tasks1.c	static	int32		MAIN GE...					NO	
SHR2	tasks1.c	static	int32		MAIN GE...					NO	
SHR4	tasks1.c	static									
SHR4.A	tasks1.c	static	int32		MAIN GE...					NO	
SHR4.B	tasks1.c	static	int32		MAIN GE...					NO	
SHR5	tasks1.c	static	int32		MAIN GE...					NO	
SHR6	tasks1.c	static	int32		MAIN GE...					NO	
__huge_val	huge_v...	static									
__huge_val.__c	huge_v...	static	uint8 [8]								
__huge_val.	huge_v...	static	uint8		MAIN GE...					NO	
arr	initialis...		int32 *		MAIN GE...		Maybe N...	MAIN GE...	max		
* arr	initialis...		int32		MAIN GE...						
current_data	initialis...	static	int32 *		MAIN GE...		Maybe N...	MAIN GE...	max		
* current_data	initialis...	static	int32		MAIN GE...						
first_paiload	initialis...		int32		MAIN GE...					NO	
output_v1	single_f...	static	int8		MAIN GE...					NO	
output_v6	single_f...	static	int32		MAIN GE...					NO	
output_v7	single_f...	static	int32		MAIN GE...					NO	
saved_values	single_f...	static	int16 [127]								
saved_values[]	single_f...	static	int16		MAIN GE...					NO	
second_paiload	initialis...		int32		MAIN GE...					NO	
tab	initialis...		int32 [10]								

Useful Features we have found in Polyspace

Divide by Zero Detection

- Many static analysis tools cannot find divide by zero, so these are often detected late in the development cycle



Best Practices for Introducing a New Tool into your Process

Training

- In general, the tool is easy to use, but as many of our users still do hand code, and have no model based experience, formal training was necessary to get them up and running
- Local training sessions were added as well to explain how the tool fits into the development process

Best Practices for Introducing a New Tool into your Process

Plan a Gradual Roll Out

- Our process requires Feature Engineers to fix or “justify” 100% of the errors found
- First wave – BugFinder only – limited set of checks turned on (September 1, 2014)
- After 6 months, next level of checks were turned on
- Plan to have 100% of what static analysis can cover from our coding standards by year end

What about CodeProver ?

Planning on rolling in CodeProver late 2015/ 2016

- Lots of value in CodeProver – larger time commitment to run analysis
- Target for CodeProver
  - New Feature Development
  - Large Feature Changes
  - Late Program Changes

- BugFinder will continue to be primary tool during normal feature development cycle.

Thank You !