

# Reducing risks and surprises with formal code verification

Security and DevOps

**Dr. Martin Becker. The MathWorks, Inc.**

Session 1: SW Security, Safety and Dependability

Room D001

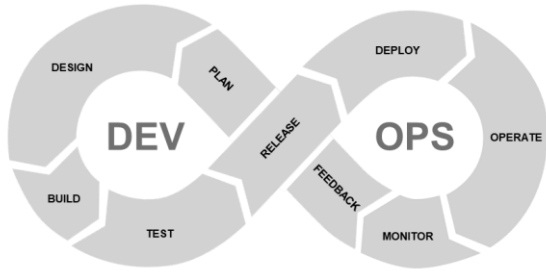
© 2023, The MathWorks, Inc.

Software Product Assurance  
Workshop 2023

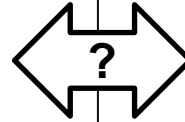
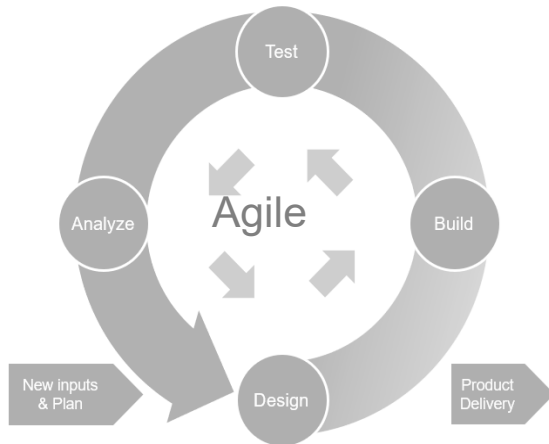
European Space Astronomy Centre, Spain  
25-28 September, 2023



# Risk is growing with “New Space”: Complexity, agility, security, ...



Scrum



Alexander Martin  
April 25th, 2023

News Technology  
Industry

### Hackers to show they can take over a European Space Agency satellite

The Washington Post

TECH

### Cyberattack knocks out satellite communications for Russian military

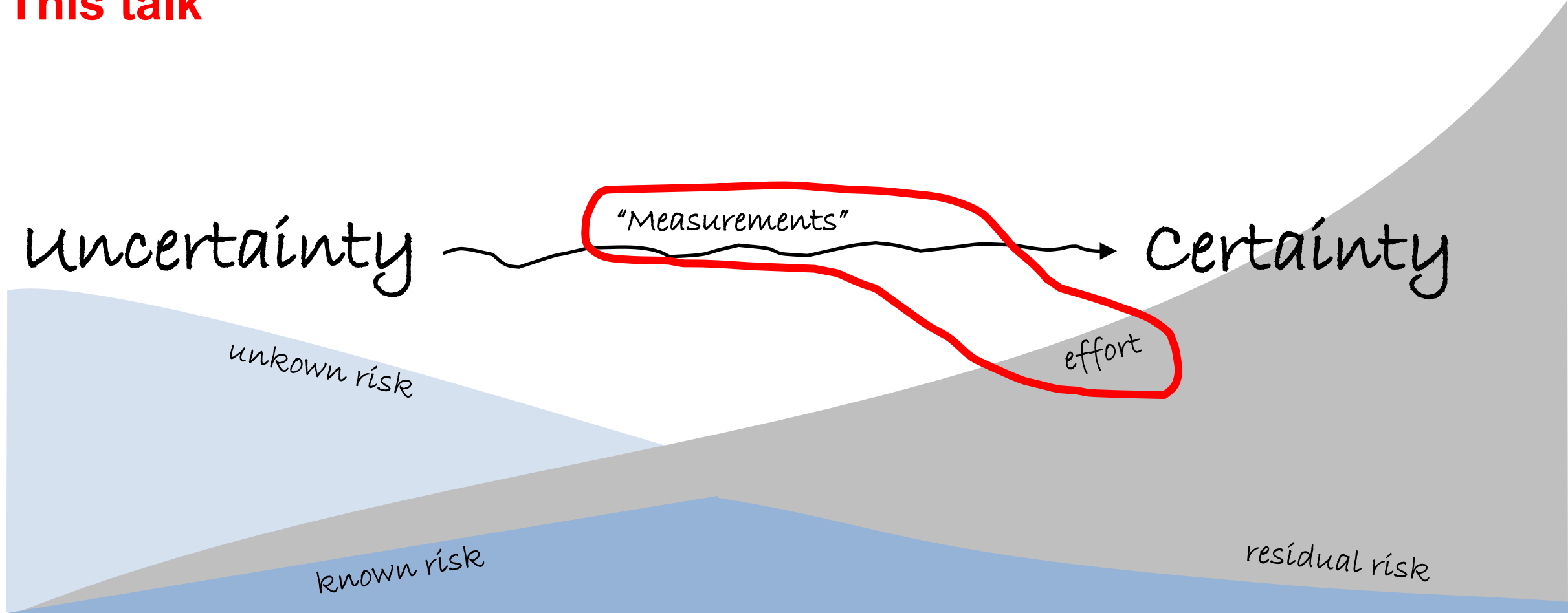
Was it pro-Ukrainian hackers or Wagner rebels?



By [Joseph Menn](#)

June 30, 2023 at 1:00 a.m. EDT

# This talk



## Conclusion: Reducing Risks and Surprises

- Use formal methods reduce uncertainty
- Perfect match for security & safety concerns
- Identify waste to minimize your effort



### Typical Results:

- 60% less defects
- 3x faster development
- Less surprises

**Part I:**  
**A powerful tool  
to reduce uncertainty**

# Does this program fail? (1) – Conventional Tools

```
1 int where_are_errors (int input) {
2     int x, y, k;
3     ...
4     k = input / 100;
5     x = 2;
6     y = k + 5;
7     ...
8     while (x < 10)
9     {
10        x++;
11        y = y + 3;
12    }
13    ...
14    if ((3*k + 100) > 43)
15    {
16        y++;
17        x = x / (x - y);
18    }
19    ...
20    return x;
21 }
```

**Cppcheck:**  
All okay.

**Cpp-lint:**  
Whitespace warnings.

...

← **What about DIV/0?**

~~Write many test cases...~~

# Does this program fail? (2) – Formal Code Analysis

```

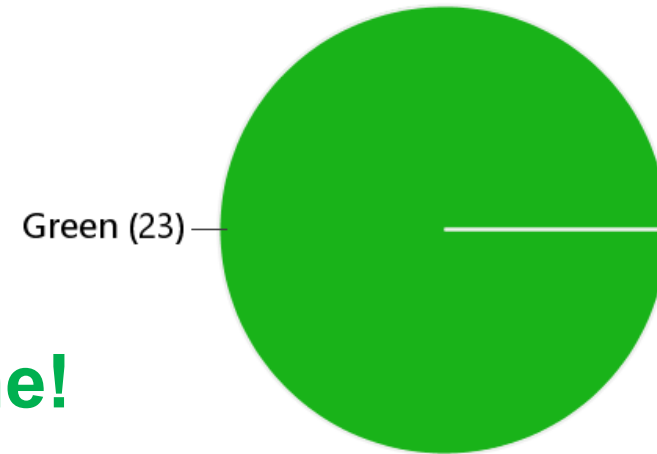
1  int where are errors (int input) {
2      int x, y, k;
3
4      k = input / 100;
5      x = 2;
6      y = k + 5;
7
8      while (x < 10)
9      {
10         x++;
11         y = y + 3;
12     }
13
14     if ((3*k + 100) >= 43)
15     {
16         y++;
17         x = x / (x - y);
18     }
19
20     return x;
21 }

```

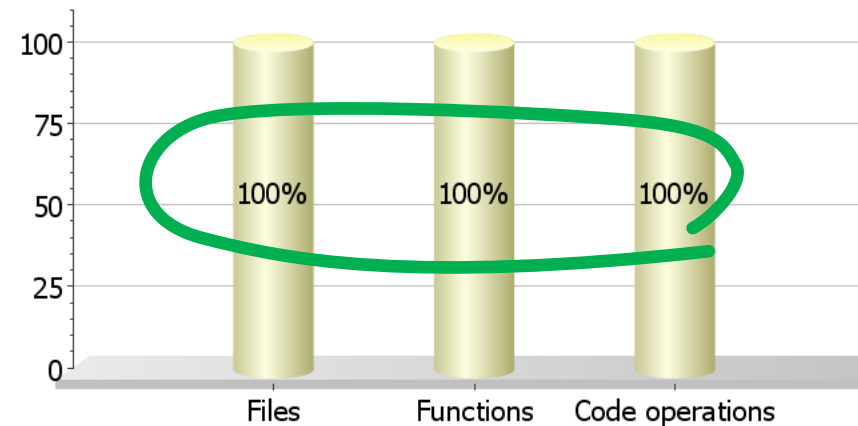
operator / on type *int 16*  
left: 10  
right: [-347 .. -2]  
result: [-5 .. 0]

**There are none!  
(guaranteed)**

**Check distribution**  
Proven: 100%



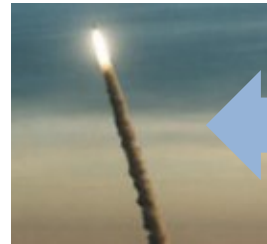
**Code covered by verification**



# Formal Code Analysis: Verification with Mathematical Certainty

1977 Theory of sound approximation of computer programs

1996 1<sup>st</sup> Ariane 5



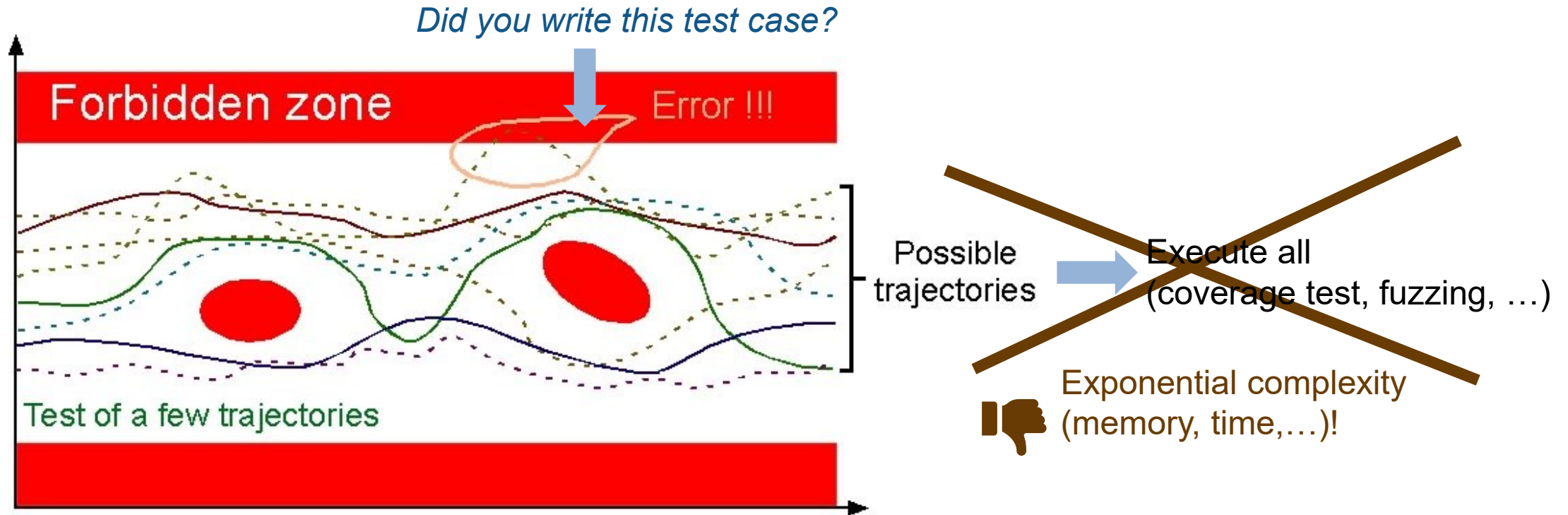
today



## How does it work?



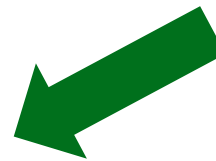
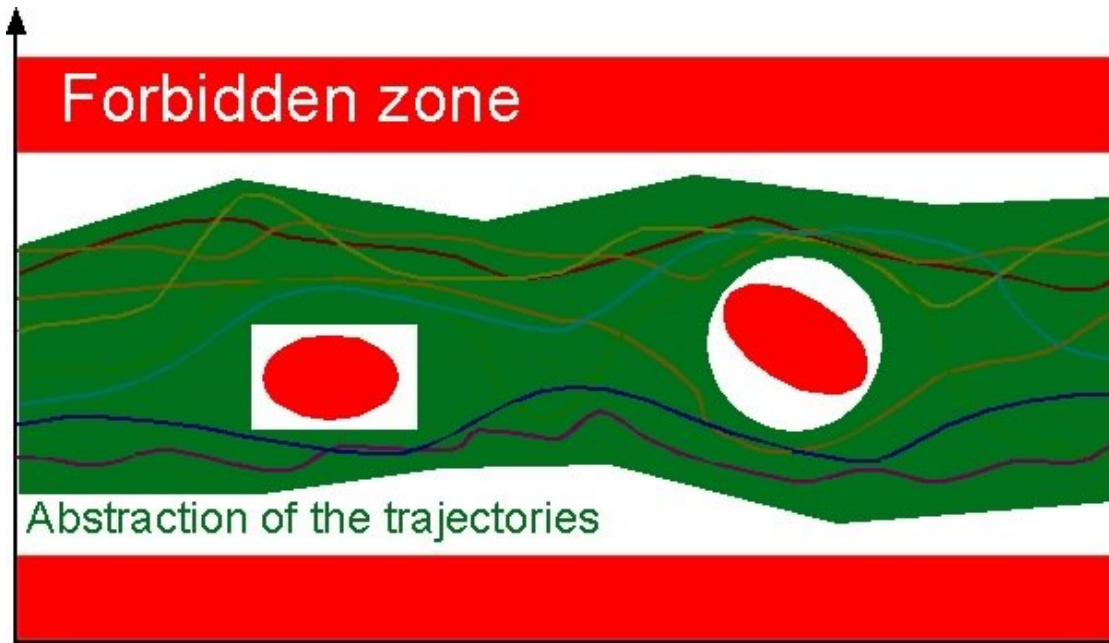
# Testing and pattern matching cannot provide certainty



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

- Edsger Dijkstra, Computer Science Pioneer

# Formal Code Analysis: Sound Math for Certainty & “Zero” Risk



E.g., simple\* abstraction based on *data ranges*:

```

34 int myloop(int x) {
35     while (1) {
36         if (x < 0)
37             break;
38         if (x % 2 == 0)
39             break;
40         x--;
41     }
42     return x;
43 }

```

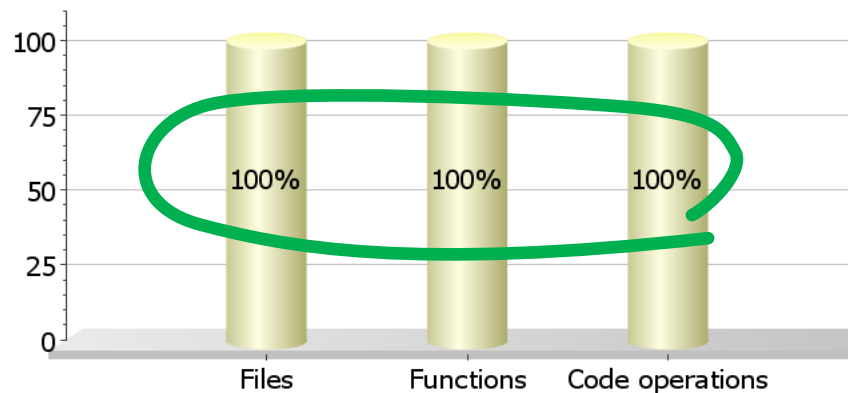
Parameter x (int 32): [1 .. 11]

Maximum number of iterations: [1 .. 2]

operator % on type int 32  
left: [0 .. 9] or 11  
right: 2  
result: [0 .. 1]

Parameter 'x' (int 32): 0 or [2 .. 8] or 10

Code covered by verification ?



**Exponentially faster than testing!**



No False Negatives ⇔ misses no bugs



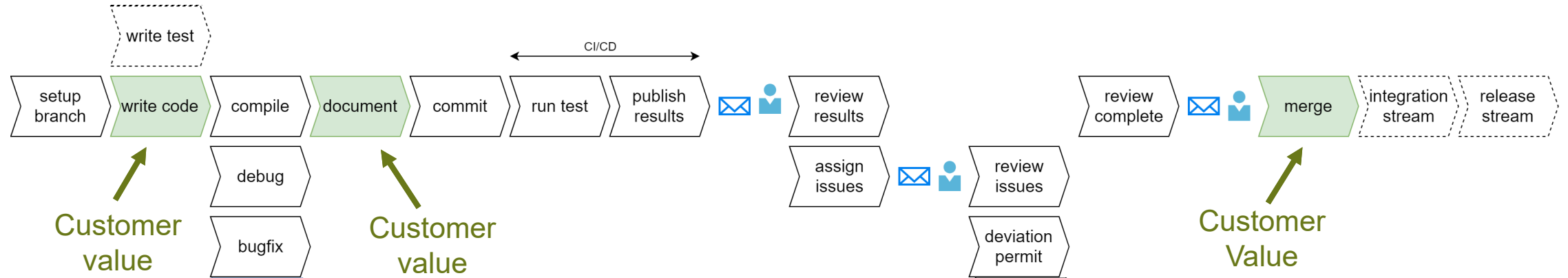
Can prove *absence* of defects



Works for large programs

# Part II: Efficient Use

# Minimize Waste (🗑️) in Your Process\*



- W1) Overproduction
- W2) Waiting Times
- W3) Handover/Transport
- W4) Overprocessing
- W5) Large Inventory/WiP
- W6) Workplace friction
- W7) Defects**

*=zero?*



★★★ Higher Quality

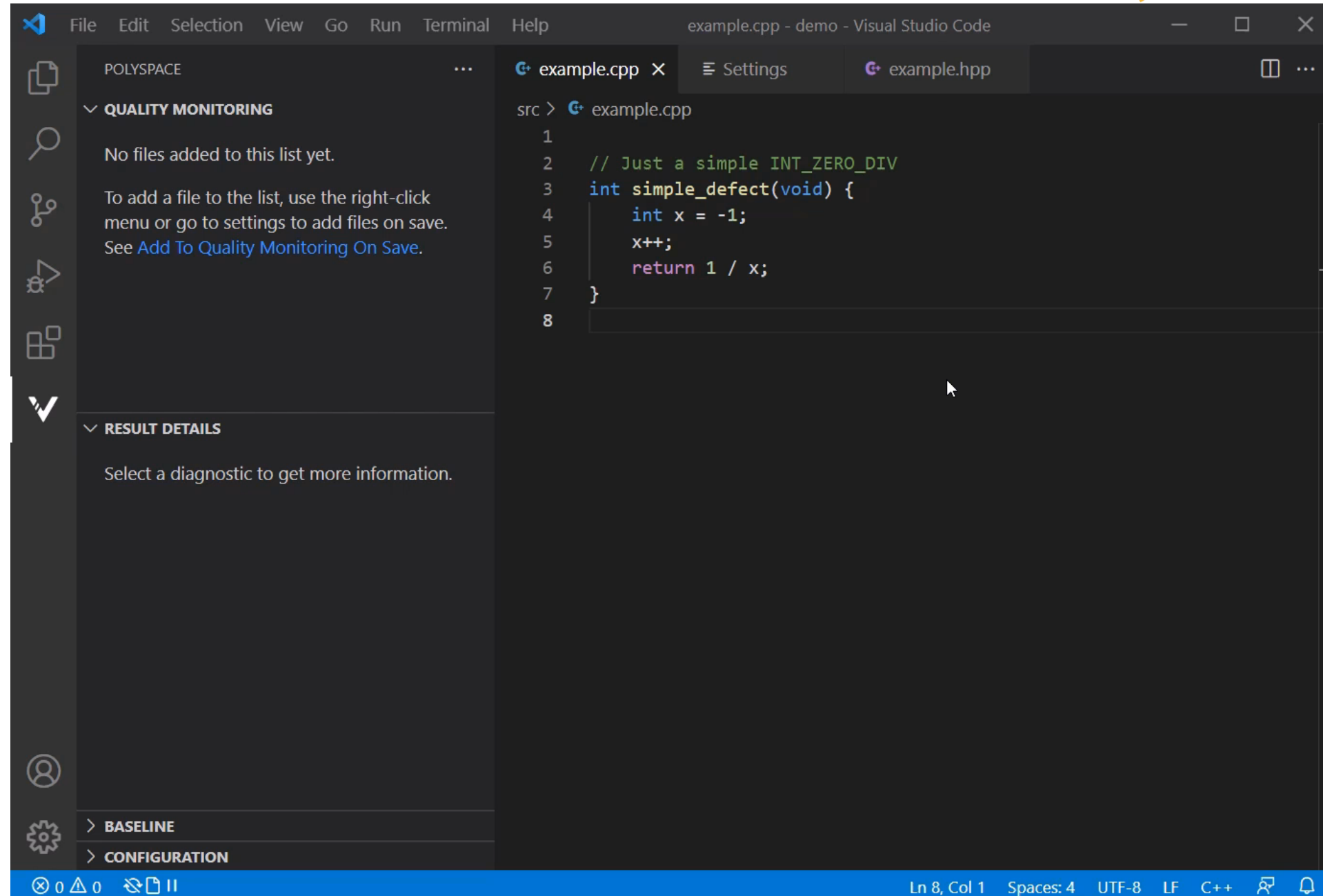
Faster Development

Lower Cost

Avoid defects at  
“the source”

Train developers

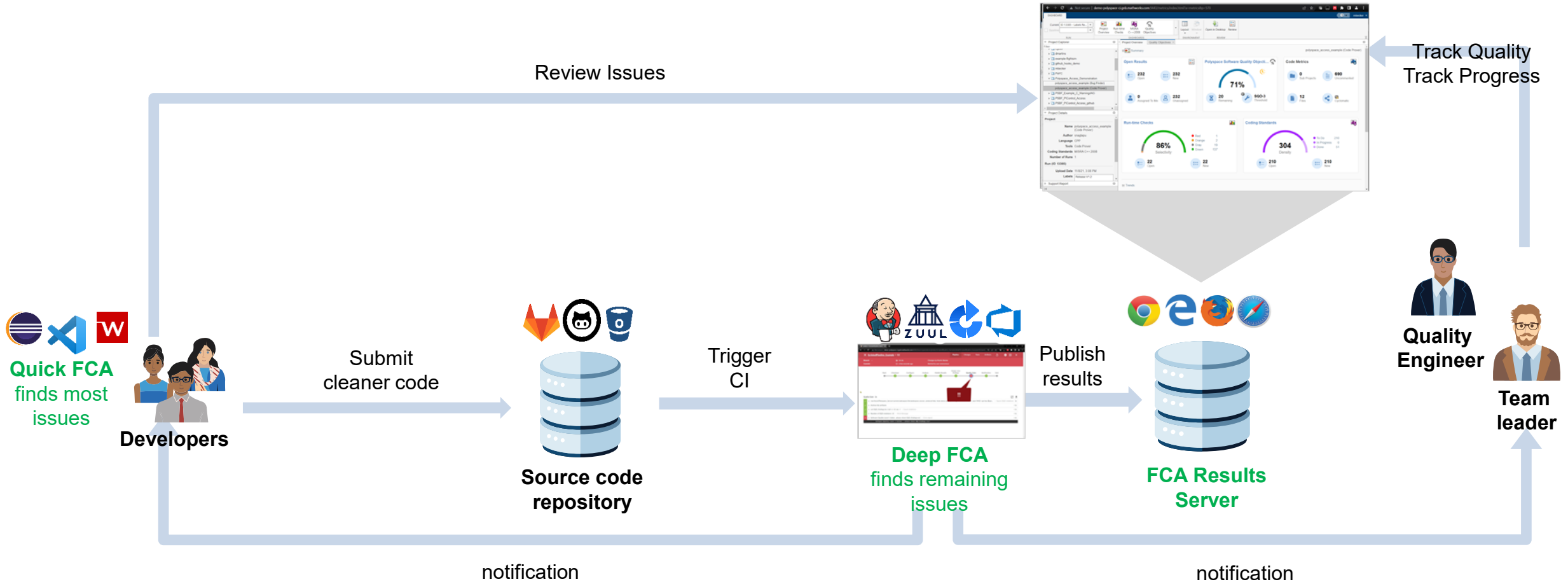
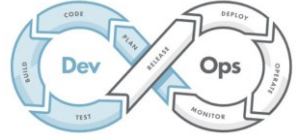
Less design  
iterations



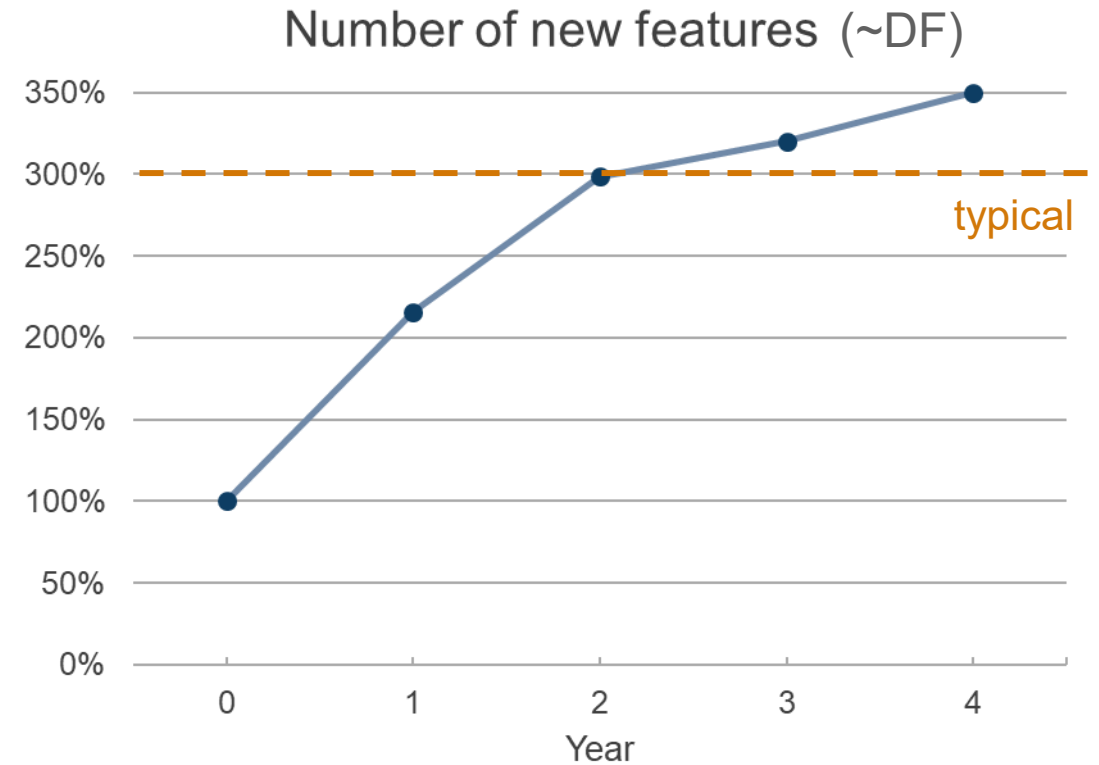
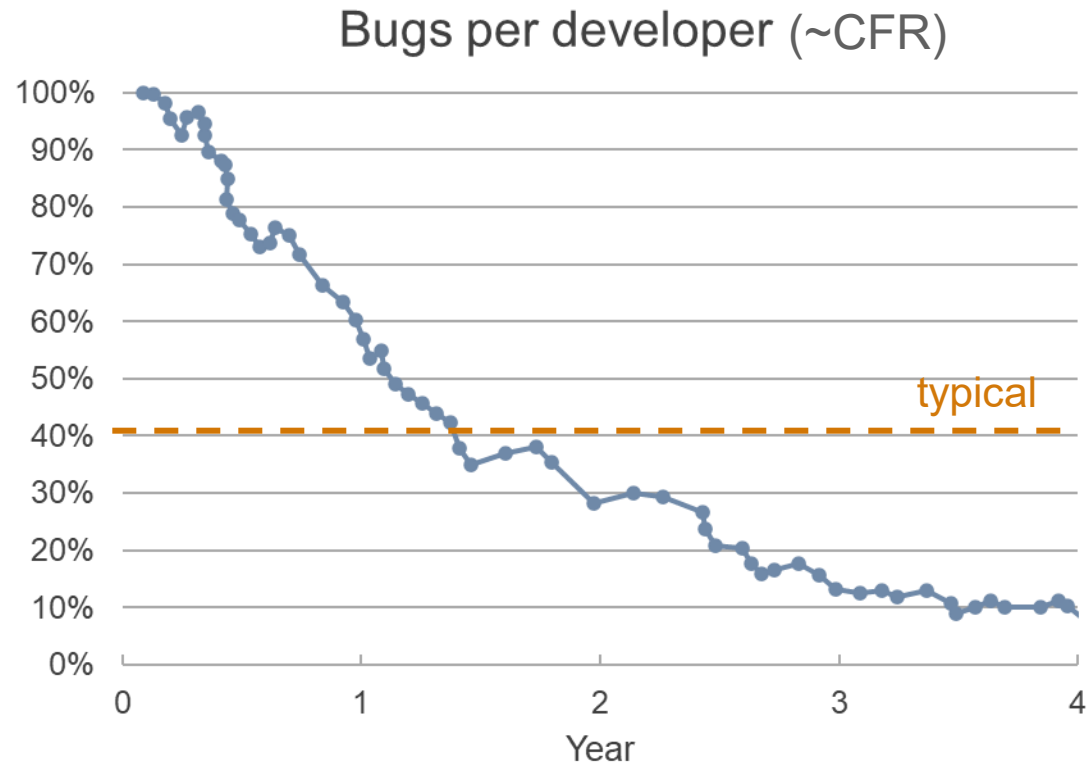
The screenshot shows the Visual Studio Code interface with a C++ file named `example.cpp` open. The code contains a function `simple_defect` that returns `1 / x`, where `x` is initialized to `-1`. The Quality Monitoring sidebar on the left is expanded, showing the `QUALITY MONITORING` section with the message: "No files added to this list yet. To add a file to the list, use the right-click menu or go to settings to add files on save. See [Add To Quality Monitoring On Save](#)." Below this is the `RESULT DETAILS` section with the message: "Select a diagnostic to get more information." The status bar at the bottom indicates the current position is `Ln 8, Col 1` with `Spaces: 4`, `UTF-8` encoding, `LF` line endings, and `C++` language.

```
File Edit Selection View Go Run Terminal Help example.cpp - demo - Visual Studio Code
POLYSPACE
QUALITY MONITORING
No files added to this list yet.
To add a file to the list, use the right-click menu or go to settings to add files on save.
See Add To Quality Monitoring On Save.
RESULT DETAILS
Select a diagnostic to get more information.
BASELINE
CONFIGURATION
src > example.cpp
1
2 // Just a simple INT_ZERO_DIV
3 int simple_defect(void) {
4     int x = -1;
5     x++;
6     return 1 / x;
7 }
8
Ln 8, Col 1 Spaces: 4 UTF-8 LF C++
```

# FCA on the CI system: Your Safety Net



# It works! One example ...



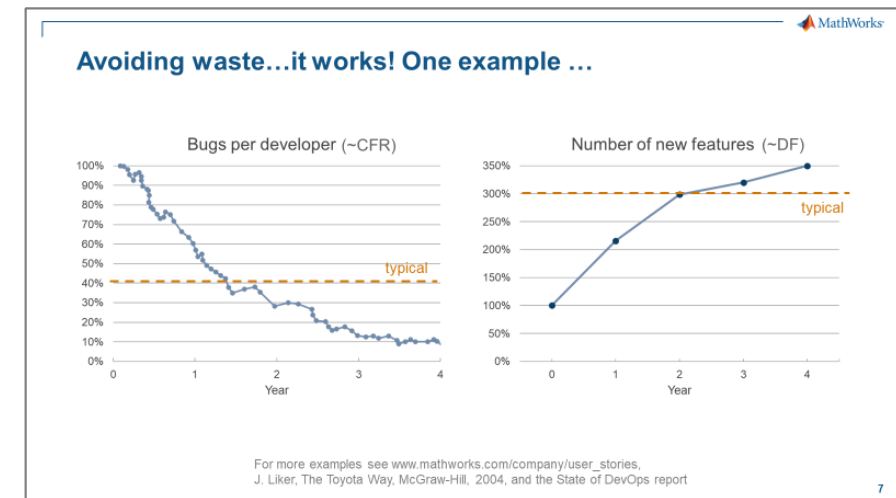
For more examples see [www.mathworks.com/company/user\\_stories](http://www.mathworks.com/company/user_stories),  
 J. Liker, The Toyota Way, McGraw-Hill, 2004, and the State of DevOps report

**Conclusion (again)**



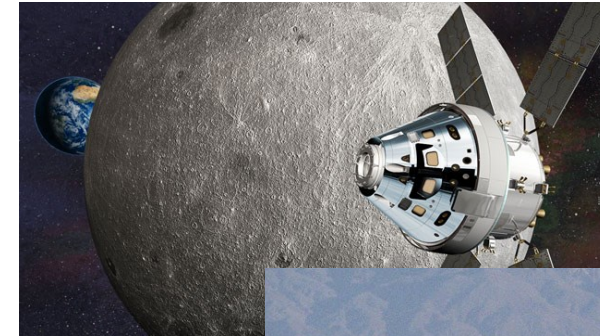
# Exceptional teams grow with their tools

- *Hansei* and *Kaizen* – reflect and improve the process continually
- Decide slowly, with consensus
- Secure and evolve team skills



# Conclusion: Reducing Risks and Surprises

- Reduces uncertainty and risk
- Perfect match for security & safety concerns
- Identify waste to minimize your effort



## Formal Code Analysis

Avoid  
waste



### Typical Results:

- 60% less defects
- 3x faster development
- Less surprises

