Lower Your Software Development Costs through Code Analysis

Shawn A. Prestridge, Senior Field Applications Engineer

IAR SYSTEMS

IAR SYSTEMS – A GLOBAL LEADING VENDOR

168 Employees with HQ in Uppsala, Sweden

Listed in Stockholm/Nasdaq

R&D investment 32% of revenue

32 years in the industry

24 hour technical support in

13 languages

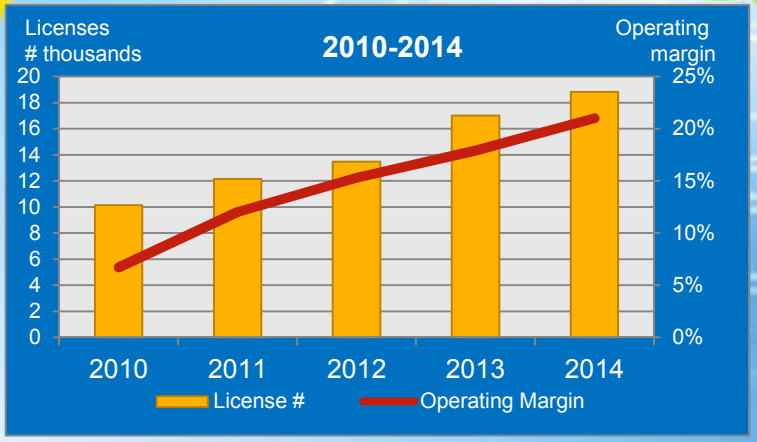
Uppsala
Munich
Sao Paulo
Tokyo
Seoul
Shanghai

London
Paris
San Francisco
Dallas
Boston
Los Angeles

+Distributor representation in

43 countries

Stability and growth

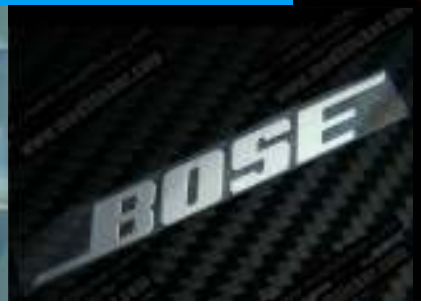


LARGE AND LOYAL CUSTOMER BASE WORLDWIDE

IAR SYSTEMS

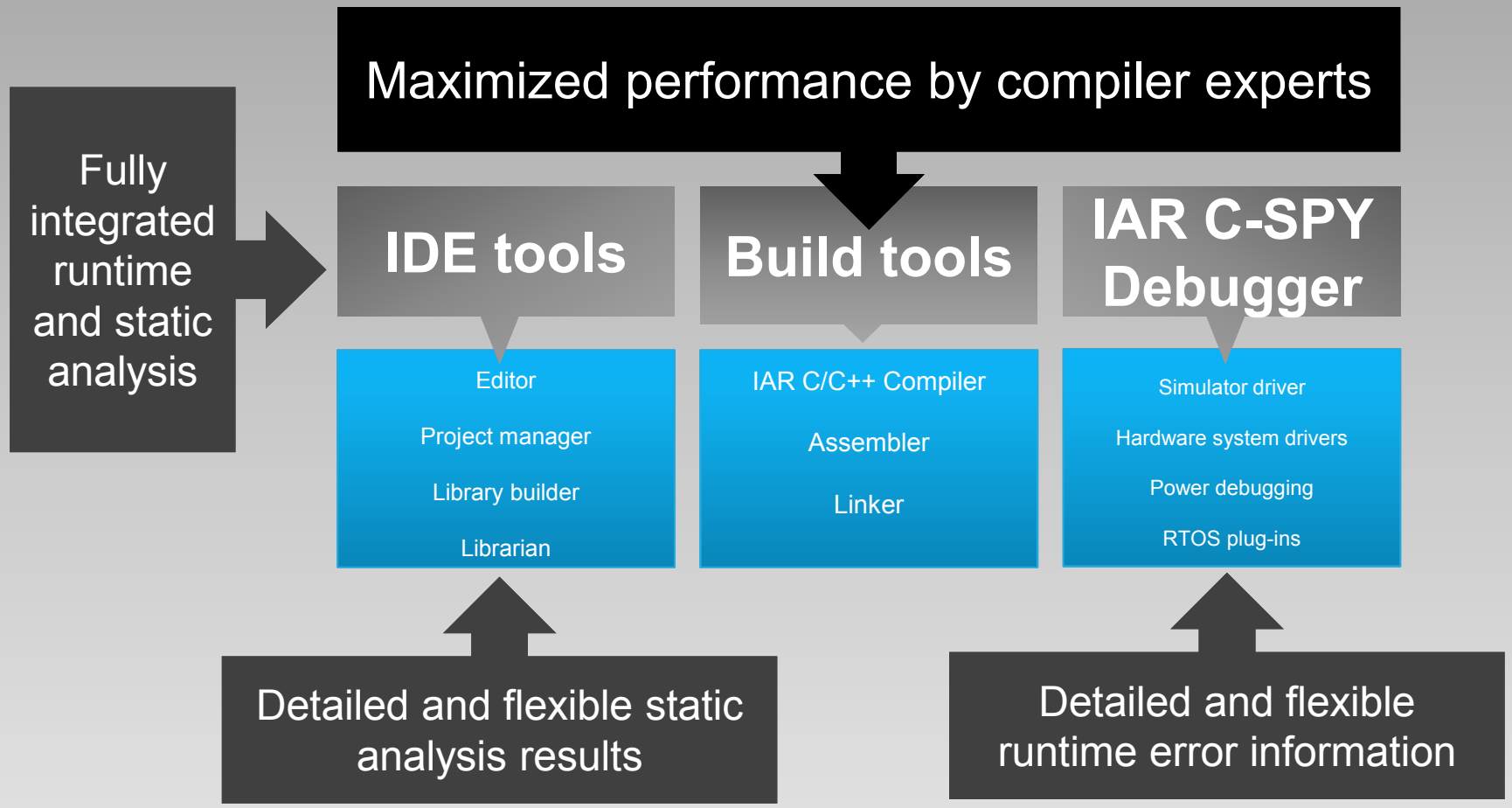
46,000+ customers

95% recurring customers
65% customers with more than one product
22,000+ support agreements
385+ enterprise agreements
200,000+ newsletter subscribers
100,000+ web visitors per month
10,000+ downloads per month
450+ partners in ecosystem



IAR EMBEDDED WORKBENCH WITH INTEGRATED ANALYSIS TOOLS

> We enable developers to **take full control of their development** and gain efficient, adaptable workflows delivering dependable products.

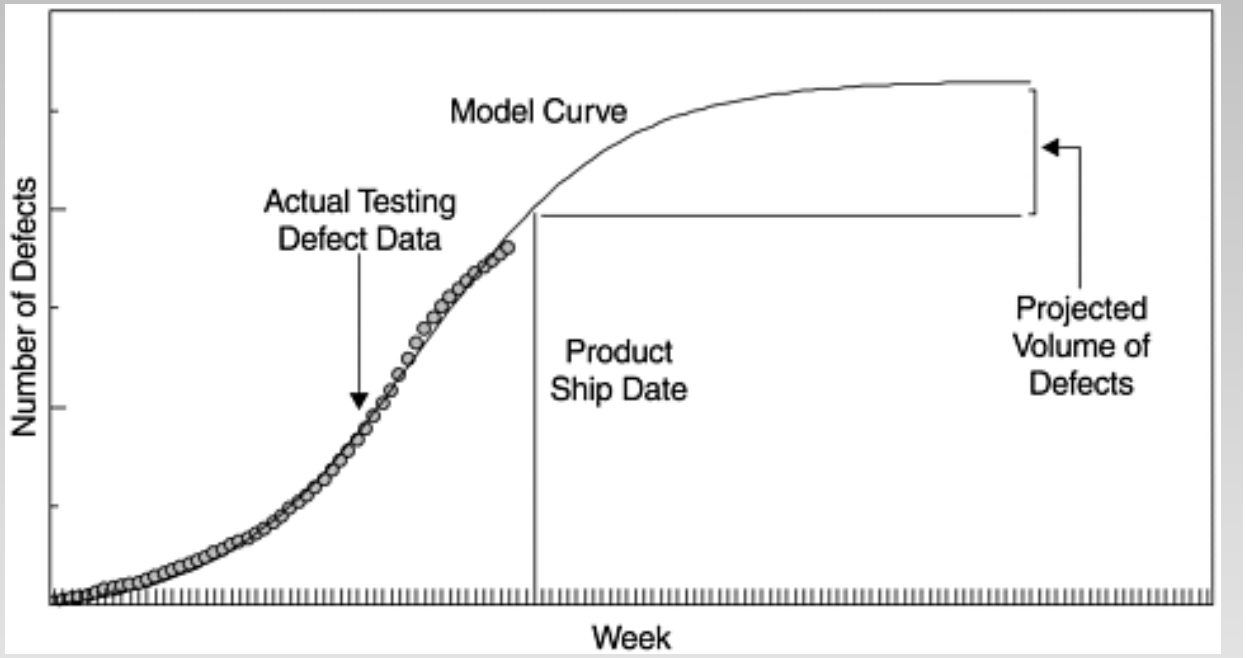


How can Code Analysis lower my costs?

Many successful embedded software companies use “Mean Time to Failure” (MTTF) to determine the quality of their software

- Will not release the product until it meets a minimum MTTF
- Basically says the number of hours you can use the product without encountering a bug
- Depending on your system, you may need an MTTF in the 10-100,000 hour range (or more!)

The way an MTTF curve is calculated, you are really hurt by having many bugs at the start of formal testing



Two types of code analysis

IAR SYSTEMS

Static analysis

- Is done at compile-time
- Looks for common causes of errors in the source code
- Flags potential issues for review by the developer
- Most common type of code analysis
- Available from many sources

Runtime analysis

- Is done at run-time
- Involves instrumenting source code by placing wrappers to check for boundary and error conditions
- Very few false-positives
- Harder to implement due to code size/performance issues
- Available from fewer sources

Static analysis

IAR SYSTEMS

What can static analysis do for me?

Static analysis can find many bug patterns, including:

- **Null pointer dereferencing**
- **Infinite recursion loops**
- **Dead code**
- **Out-of-bounds array references (if index can be determined at compile-time)**
- **Arithmetic errors**
    - **Divide by 0**
    - **Floating point comparisons using == or !=**
    - **Overflow/underflow of numbers for specified type**
- **Dangling else statements**

Continued on next slide…

What can static analysis do for me?

IAR SYSTEMS

- Assignment-as-condition in `if`/`while`/`do` statements
- Negative integers cast to unsigned integers
- Dead code
- Misuse of runtime library:
  - Buffer overruns
  - Heap corruption
- Pointer misuse
- Identifies redundant code
- Finds dereferences to resource pointers
- Code side-effects due to conditional statements

And many other violations!

Examples

```
87      for(Int32U i = (HID_ID_NUMB?1:0); i < HID_ID_NUMB+1; i++)
88      {
```

By examining the boundaries on the for loop, you quickly see that this will never execute! This indicates you may have made a mistake in assigning the boundaries.

Examples

```
642        status = DATA_EEPROM_EraseWord(Address & 0xFFFFFFFC);
643        status = DATA_EEPROM_FastProgramWord((Address & 0xFFFFFFFC), tmp);
```

stm32l1xx_flash.c (4 messages)

Value assigned to variable `status' is n... RED-unused-assign

As you can see, we write to a variable and then immediately overwrite the value with something else. This indicates that we may have some missing code between the statements or that we may have assigned one of the statements' return values to the wrong variable.

Examples

```
276     pllmul = PLLMulTable[(pllmul >> 18)];
277     plldiv = (plldiv >> 22) + 1;
```

system_stm32l1xx.c (1 message)

Array `PLLMulTable' 1st subscript interval [0,15] may be out of bounds [0,8]

In this case, C-STAT knows from the source that your array has nine elements but sees that by only shifting right 18 places, the index may be from 0 to 15. It implies that you probably meant to shift down one more place.

Examples

C-STAT scans your code to make sure that your pointers have been properly initialized and warns you when you are using one that isn't.

What can static analysis do for me?

IAR SYSTEMS

Static analysis can also ensure that your **code conforms to coding standards**

- **Promotes reusability** within your organization (**lowers development costs** on future projects)
- Helps to **constrain effects of code/spec changes** (**lowers development costs** by only testing what has changed)
- Makes **code easier to read and understand** during walkthroughs

Static analysis should be used **at the start of your project!**

Runtime analysis

IAR SYSTEMS

What can runtime analysis do for me?

Runtime analysis can also find many bug patterns, including:

- **Out-of-bounds detection for pointers/arrays**
- **Heap consistency checking**
  - Multiple deallocation calls for same memory block
  - Tracking down memory leaks
- **Arithmetic error checks**
  - Integer over/underflows
  - Integer conversion failures during casting
  - Division by 0
  - Unhandled switch cases

Speeding the path to safety certifications

Certifications are **easier to achieve** when you can prove that your code conforms to a **coding standard**.

Testing reports show that the overall **number of defects** in the software **is low**, despite many hours of testing and **proves maturity** of **your development organization**

Code analysis also shows that your **results are repeatable** because you have a **process** in place **to find and fix defects**.

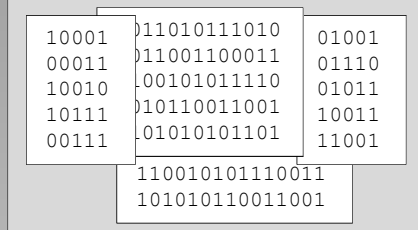
IAR Systems C-STAT and C-RUN solutions

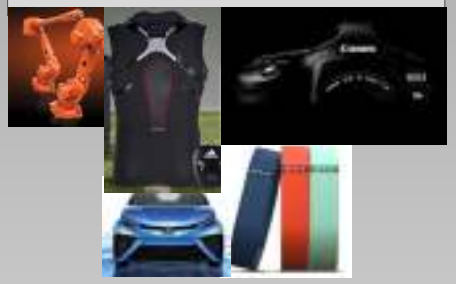
IAR SYSTEMS

Implement your design in code

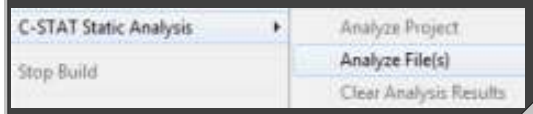
Build and debug the application

Release the application

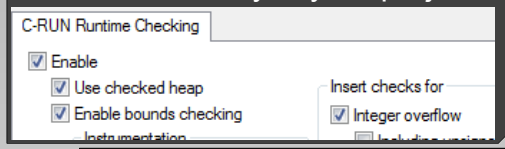




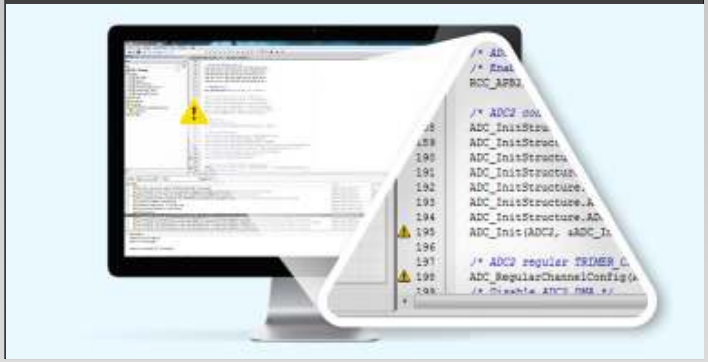
Let C-STAT analyze your code



Let C-RUN analyze your project



Review potential issues



Investigate runtime errors

Requirements → Design → Implementation → Verification → Maintenance

C-STAT STATIC ANALYSIS

ADD-ON PRODUCT AVAILABLE FOR IAR EMBEDDED WORKBENCH

IAR SYSTEMS

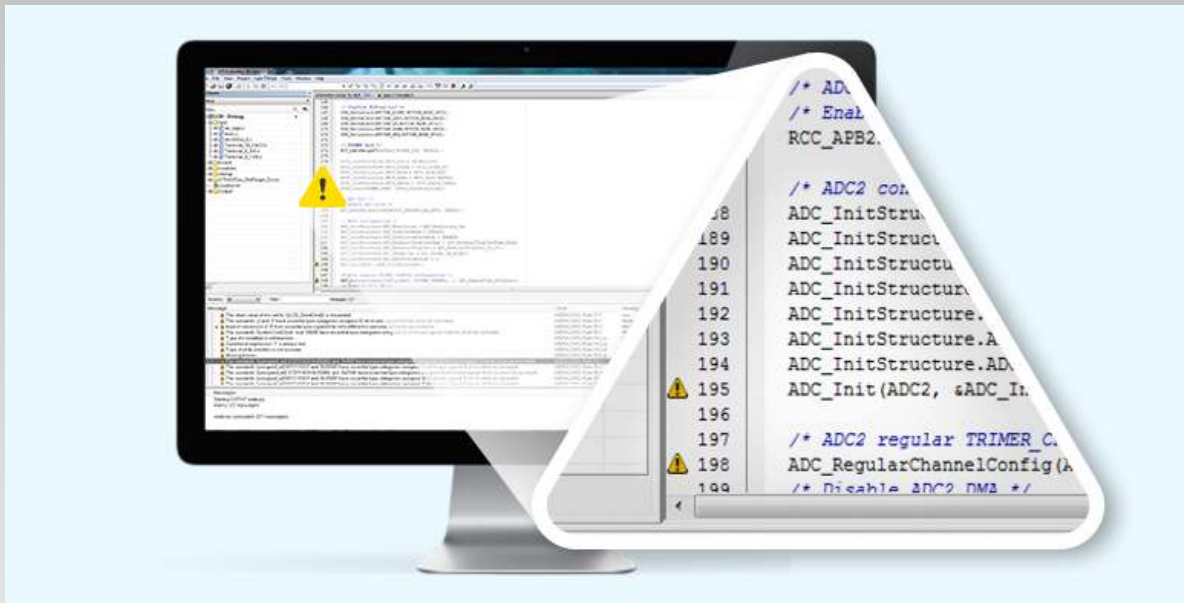
Analysis of **C and C++** code

Intuitive and **easy-to-use settings** with flexible rule selection

Checks compliance with rules as defined by MISRA C:2004, MISRA C++:2008 and **MISRA C:2012**

Includes ~250 checks mapping to hundreds of issues covered by **CWE** and **CERT C/C++**

Over 500 rules!



C-RUN RUNTIME ANALYSIS

ADD-ON PRODUCT AVAILABLE FOR IAR EMBEDDED WORKBENCH

Support for **C and C++** code

Intuitive and **easy-to-use settings**

Code correlation and graphical feedback in editor

Bounds checking to ensure accesses to arrays and other objects are within boundaries

Heap and memory leaks checking

Comprehensive and **detailed** runtime error information

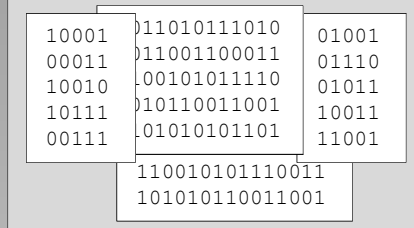
Demonstration of seamless workflow

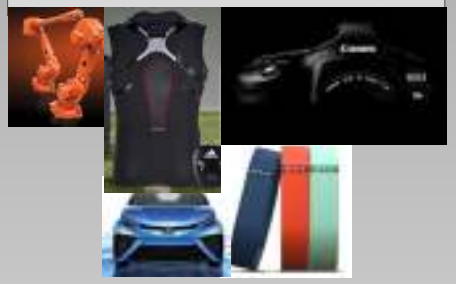
IAR SYSTEMS

Implement your design in code

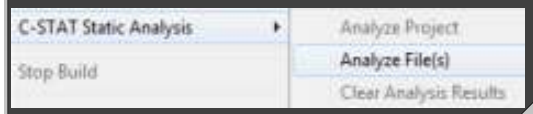
Build and debug the application

Release the application

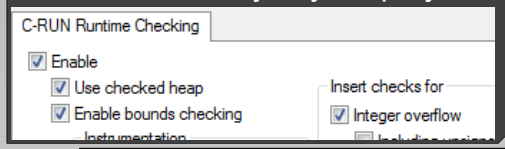




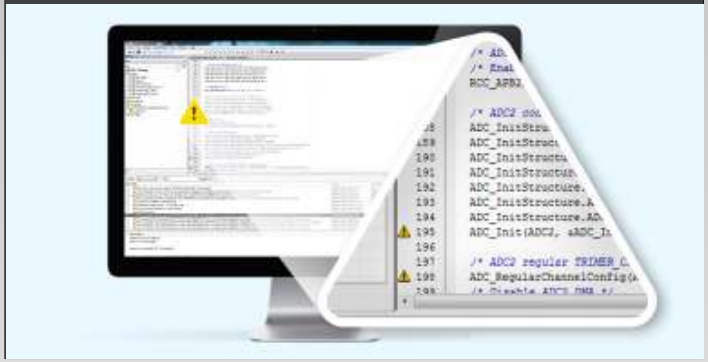
Let C-STAT analyze your code



Let C-RUN analyze your project



Review potential issues



Investigate runtime errors

Requirements → Design → Implementation → Verification → Maintenance

SUMMARY

- Why finding bugs early is important
- What defects can be found with static analysis
- How runtime analysis instruments your code
- Speeding the path to safety certification
- Seamless integration into your workflow
- Extensive customer support
- Code analysis is for every developer and every organization!