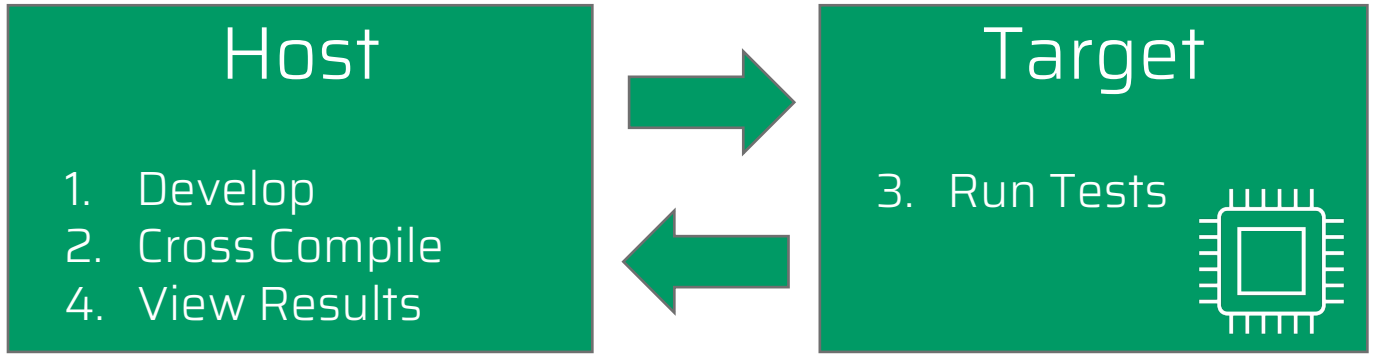
embeff
BETTER EMBEDDED.

The hidden risks in off-target testing

Requirements and workflows for safe and modern embedded development

Daniel Penning, embeff GmbH

Unit testing in embedded



“On-Target“ Tests

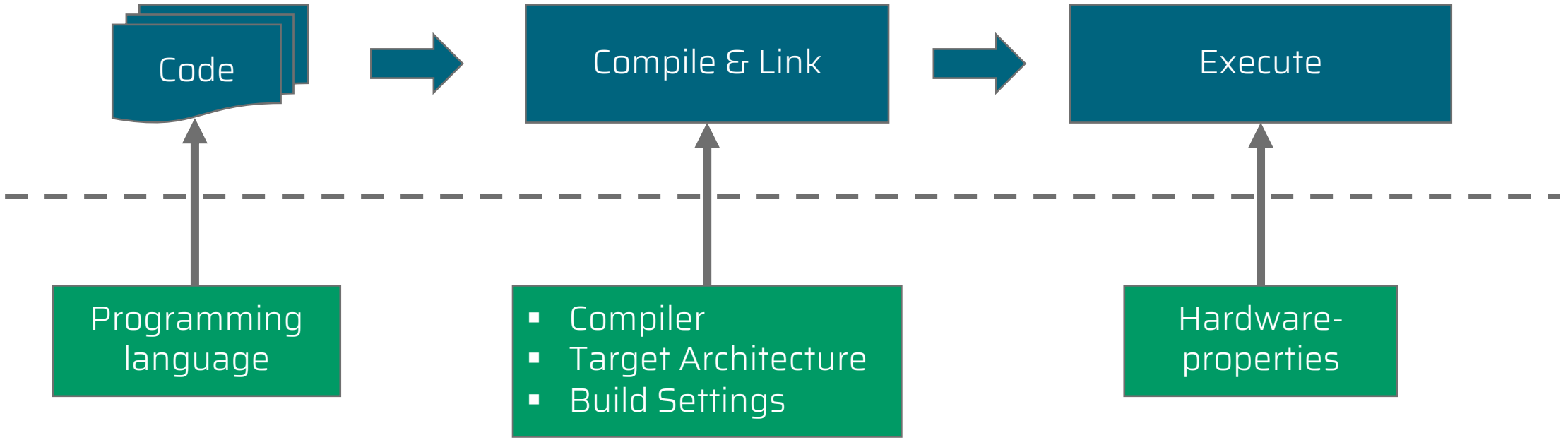
- slow
- difficult to automate

“Off-Target“ Tests

- fast
- easy to automate


MIND THE GAP

What is causing the gap?



(1) Different Evaluation Order

```
#include <iostream>

int fun1() { printf("fun1() \n"); return 0; }
int fun2() { printf("fun2() \n"); return 0; }

void foo(int x, int y) { printf("foo() \n"); }

int main() {
    foo(fun1(), fun2());
}
```

x86-64 with gcc 9.2

```
fun2()
fun1()
foo()
```

Cortex-M4 with arm-gcc-none-eabi 8

```
fun1()
fun2()
foo()
```

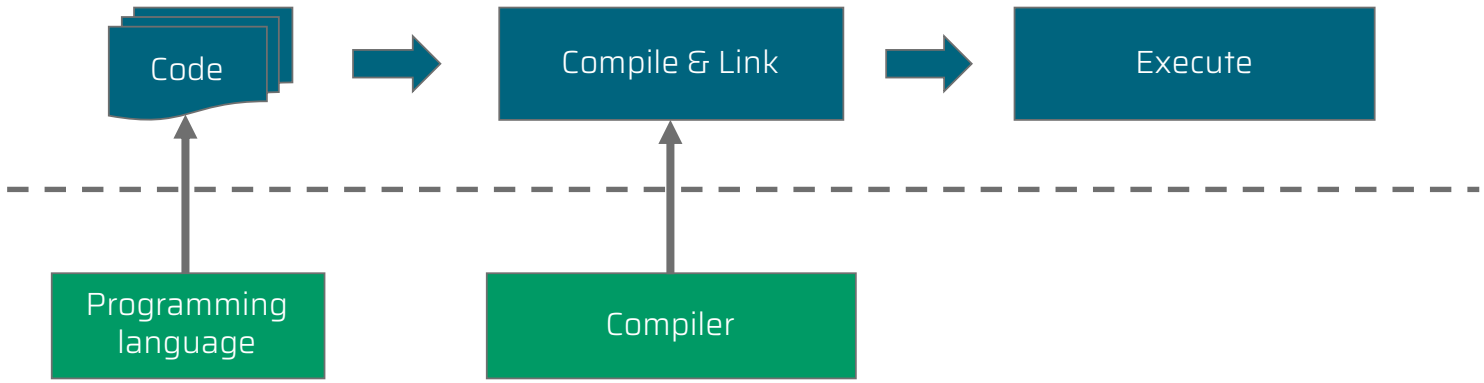
(1) Different Evaluation Order – Why?

```
int main() {
    foo(fun1(), fun2());
}
```

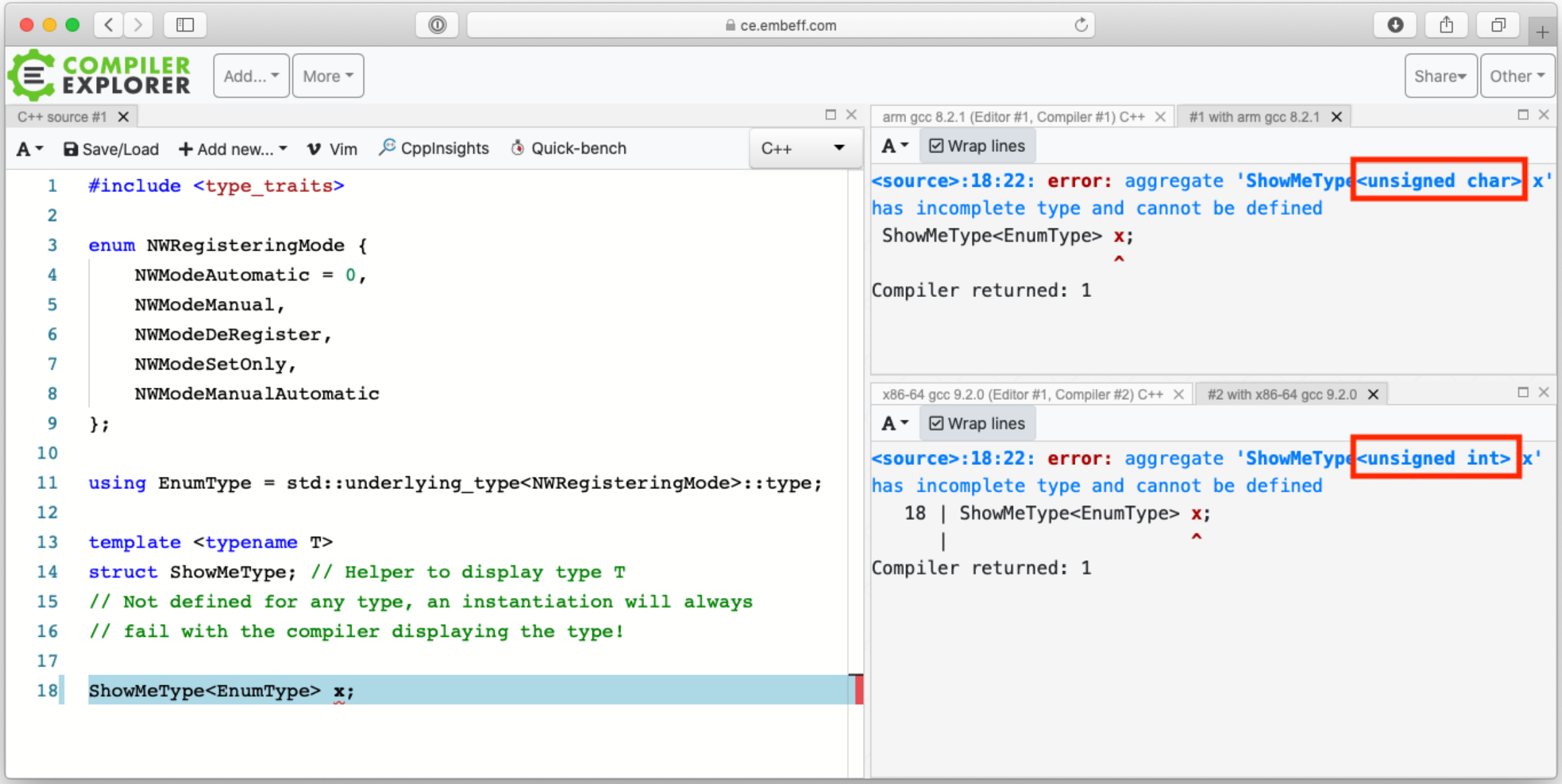
8.The order of evaluation of arguments is unspecified. All side effects of argument expression evaluations take effect before the function is entered.

C++98 Standard, 5.2.2 Function call [1]

unspecified behavior
behavior, for a well-formed program construct and correct data, that depends on the implementation.



(2) Different Enum Size



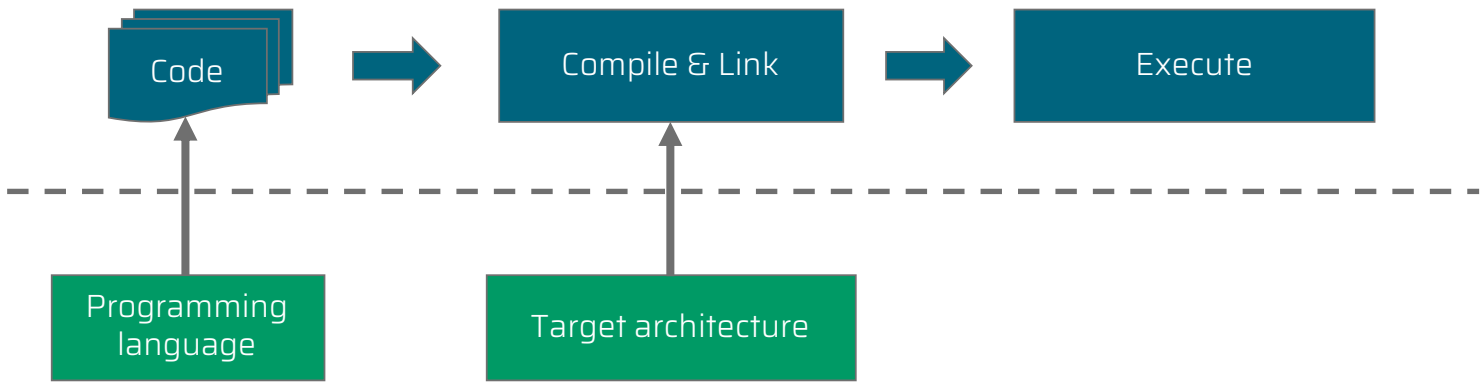
(2) Different Enum Size – Why?

```
enum NWRegisteringMode {
    NWModeAutomatic = 0,
    NWModeManual,
    NWModeDeRegister,
    NWModeSetOnly,
    NWModeManualAutomatic
};
```

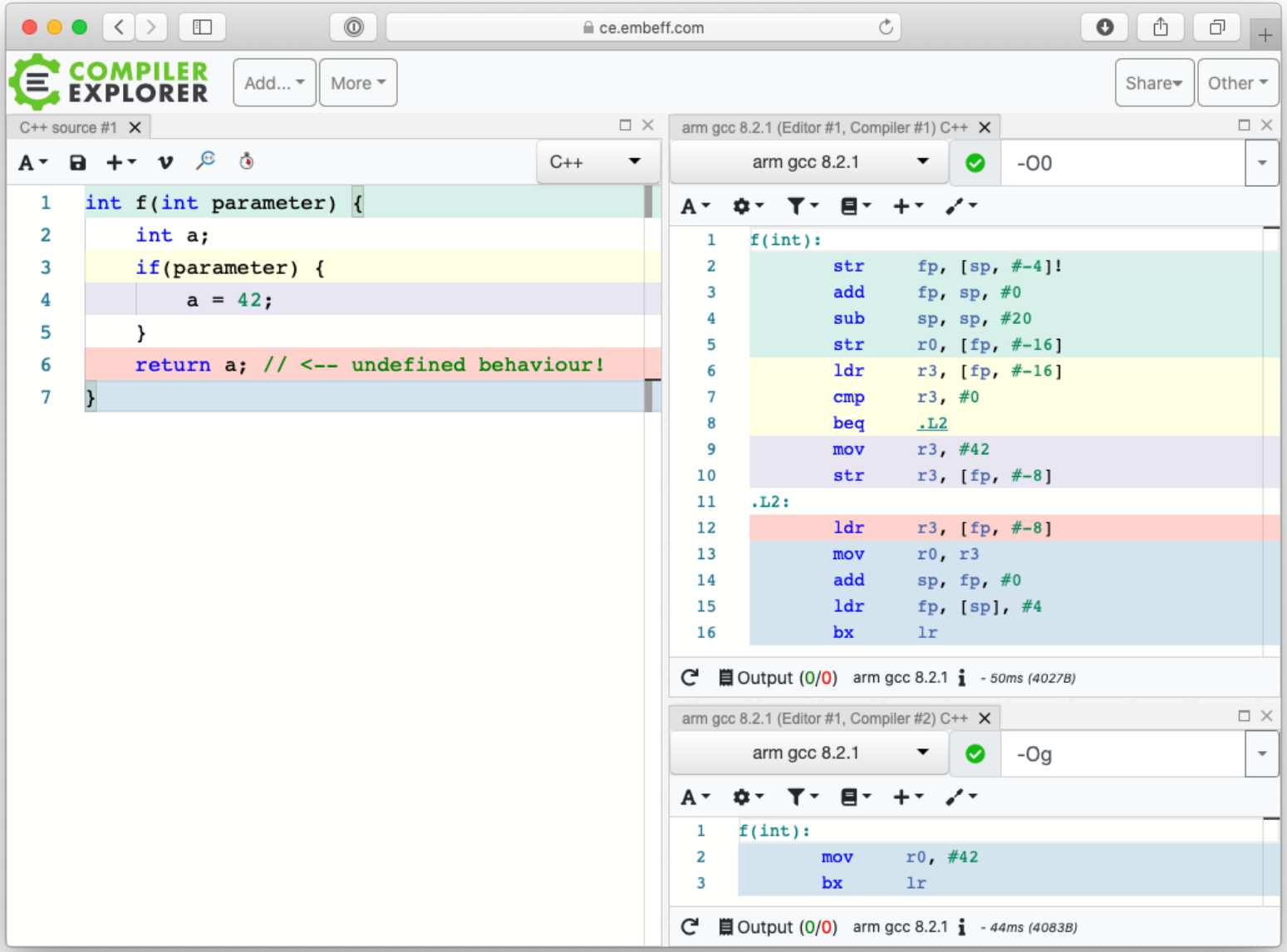
It is implementation-defined which integral type is used as the underlying type except that the underlying type shall not be larger than int unless the value of an enumerator cannot fit in an int or unsigned int.

C++17 Standard, 10.2 Enumeration declaration [2]

implementation-defined behaviour
behaviour, for a well-formed program construct and correct data, that depends on the implementation and which is documented at each implementation.



(3) Optimizer affects results



Unoptimized build

- parameter == 0:
  Return uninitialized stack memory
- parameter != 0:
  Return 42

Optimized build

- Return 42

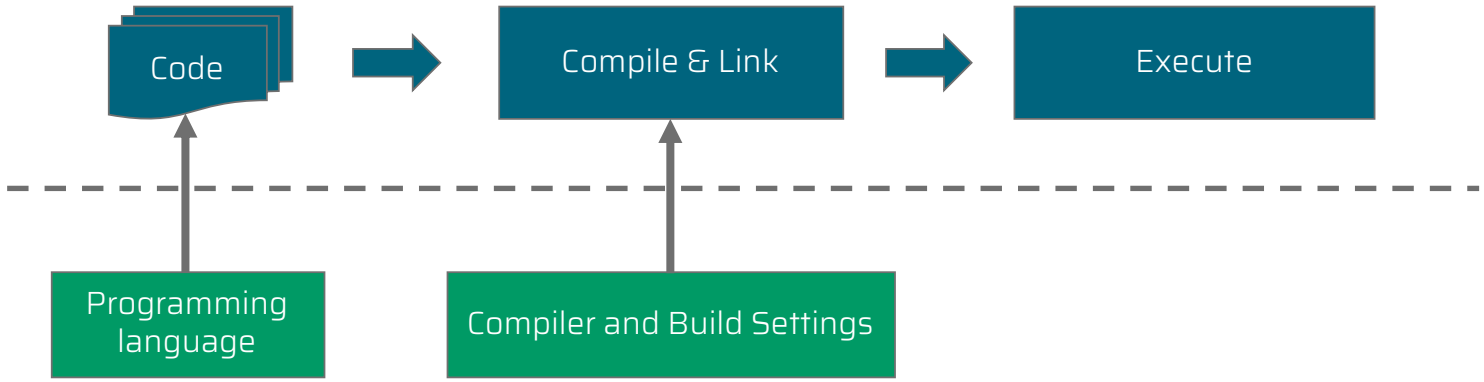
(3) Optimizer affects results – Why?

```
int f(int parameter) {
  int a;
  if(parameter) { a = 42; }
  return a; // <-- undefined behaviour!
}
```

Undefined behaviour renders the entire program meaningless if certain rules of the language are violated.

cppreference.com

Undefined behaviour
behaviour for which this International Standard imposes no requirements



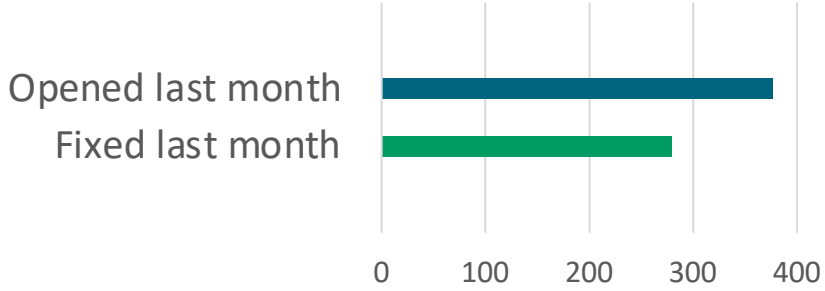
(4) Toolchain Bugs – An example

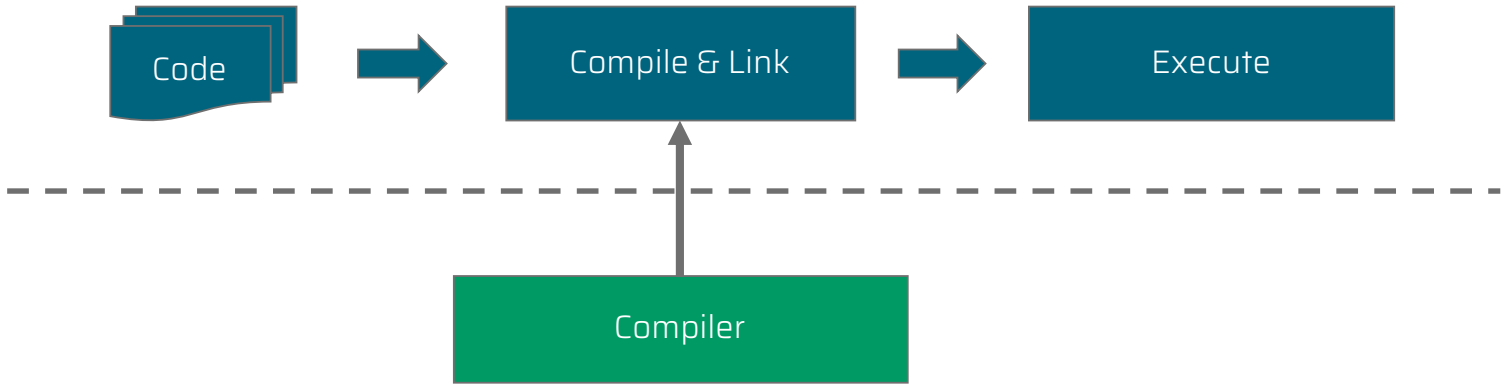
4.9 series reproducibly corrupts register R7

Code below does not exhibit problem when compiled with 4.8 2014q3. Code trashes register R7 with 4.9 2015q1, q2, or q3 and the flagged braces are uncommented. Commenting out the flagged braces removes the problem with the 4.9 series.

GNU Arm gcc #1527413 [3]

GNU gcc Toolchain
Total Bugs open: 14.318
November 2020 [4]





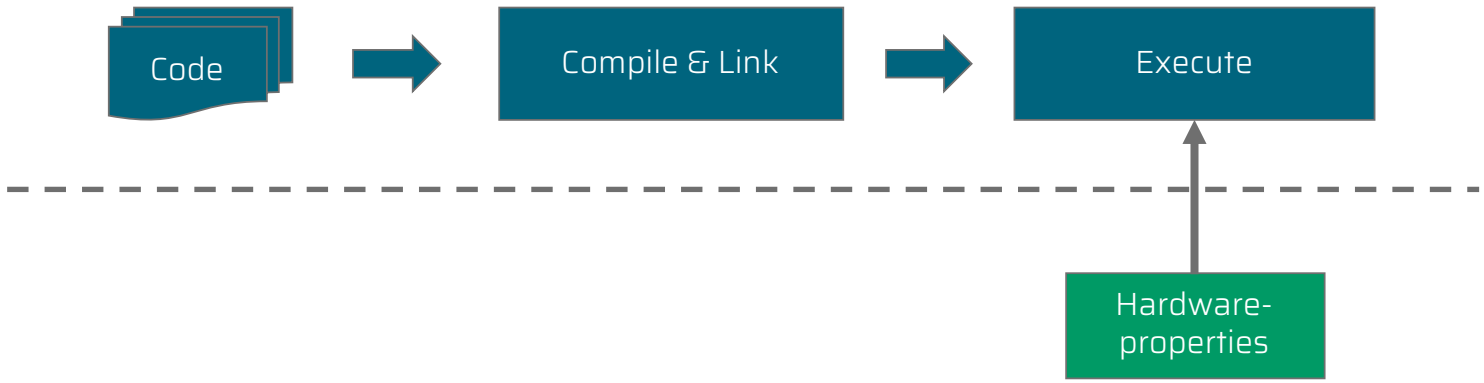
(5) Hardware Bugs – An example

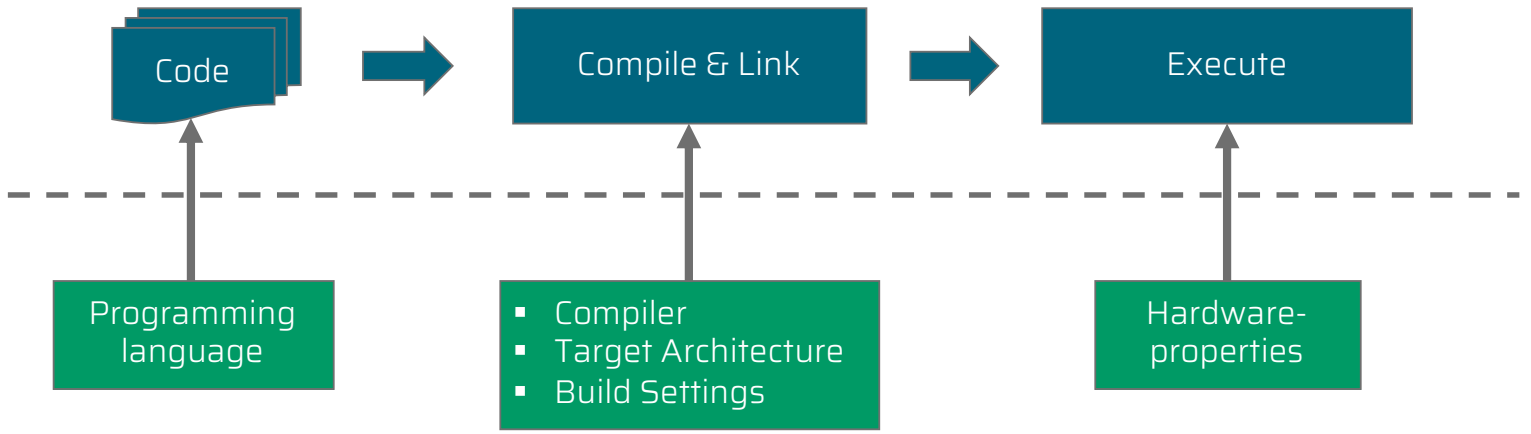
Fused MAC instructions give incorrect results for rare data combinations

The Cortex-M4 processor includes optional floating-point logic which supports the fused MAC instructions (VFNMA, VFNMS, VFMA, VFMS). This erratum causes fused MAC operations on certain combinations of operands to result in either or both of the following:

- A result being generated which is one Unit of Least Precision (ULP) greater than the correct result.
- Inexact or underflow flags written to the Floating-point Status Control Register, FPSCR, incorrectly.

Arm Errata 839676 [5]





Workflow recommendations

How to mitigate the off-target/on-target gap

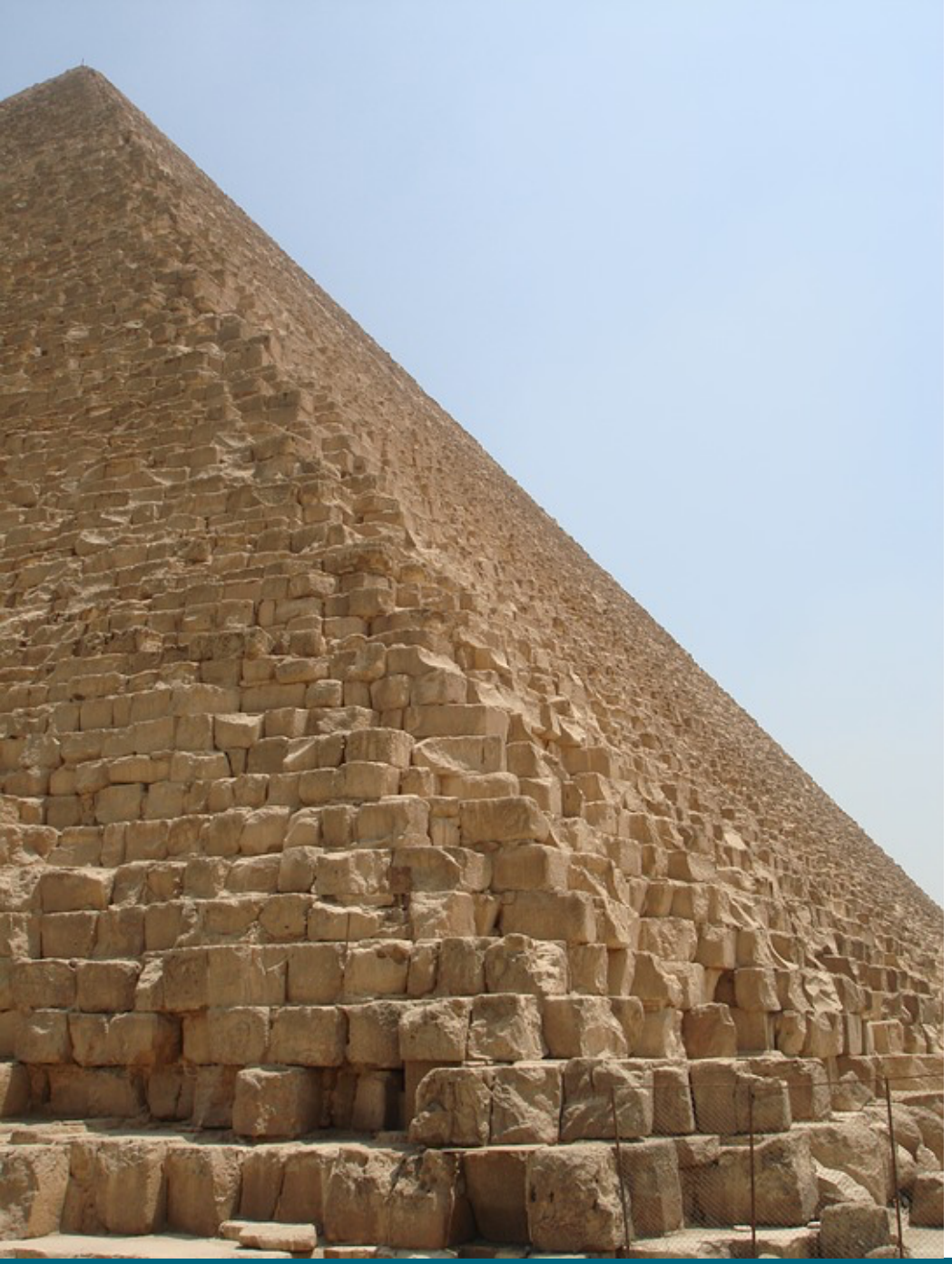
embeff
BETTER EMBEDDED.

I) Prevention

- Use a high warning level
- Treat warnings as errors

- Use multiple compilers
  (e.g gcc, clang, Visual Studio)

- Use static analyzers



II) Start Unit Testing

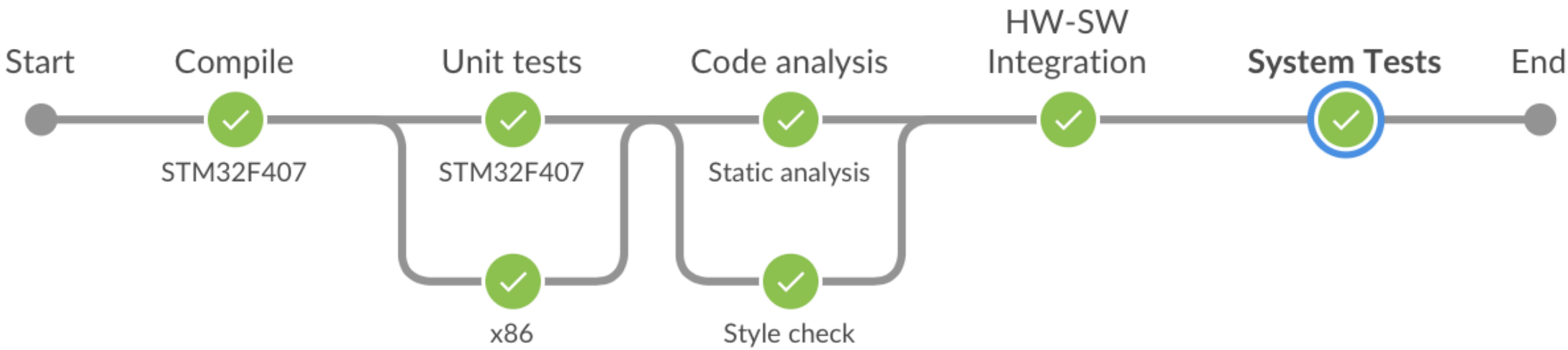
- Focus on areas with high density of logic

embeff
BETTER EMBEDDED.

III) Leverage Your Unit Tests

- Run them on-target
- Use build settings from production
- Test with unoptimized and optimized builds to expose undefined behavior („triangulate“ broken code)

IV) Automate everything in your CI



How to automate On-Target tests?

The embeff **ExecutionPlatform**

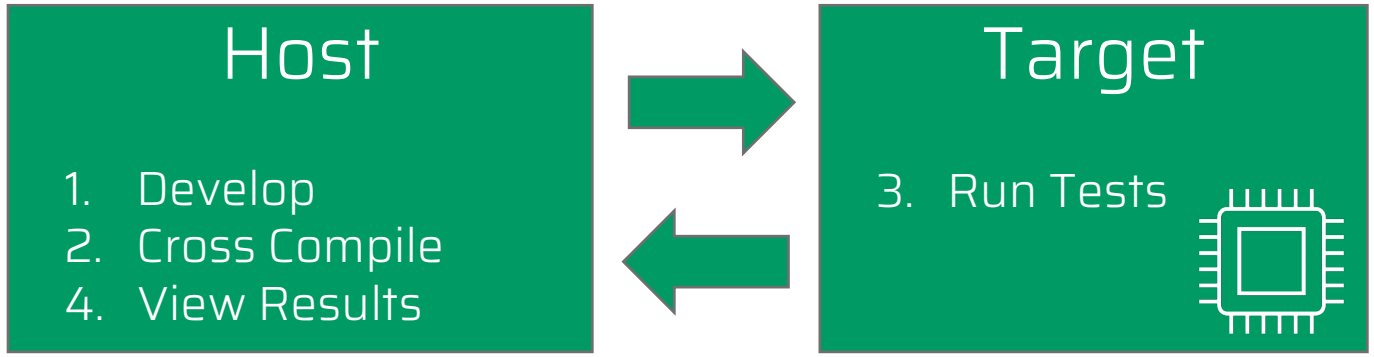
- Supports all major testing frameworks
- Supports On-Target unittests
- Supports your specific MCU
- No more hardware needed
- Easy Continuous Integration setup

https://embeff.com/executionplatform/

Summary

Problem	Mitigation
(1) Different Evaluation Order	Multiple Compilers, On-Target Tests
(2) Different Enum Size	Multiple Compilers, On-Target Tests
(3) Optimizer Affects Result	Warnings, Static Analyzer, On-Target Tests
(4) Toolchain Bug	On-Target Tests
(5) Hardware Bug	On-Target Tests

Assesment: Unit testing in embedded



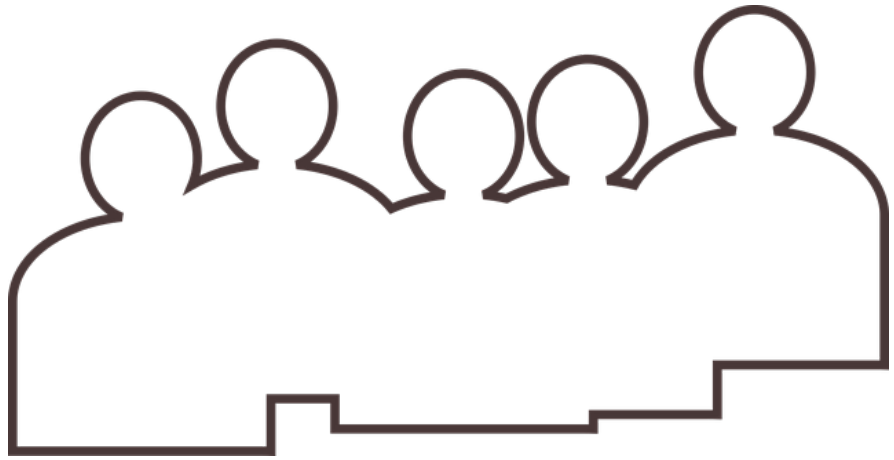
“On-Target“ Tests

- slow
- difficult to automate
- reliable

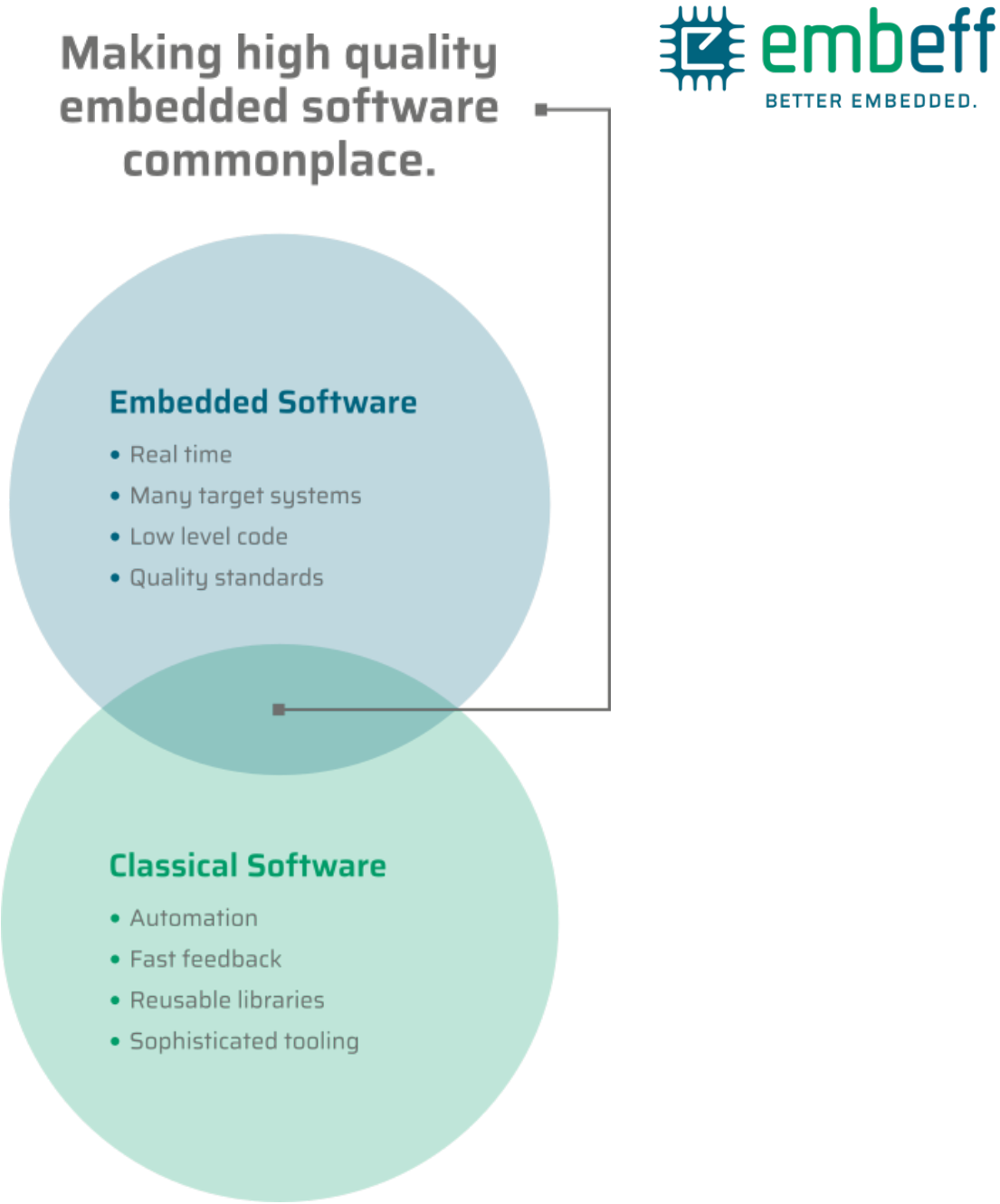
“Off-Target“ Tests

- fast
- easy to automate
- unreliable

Thanks for listening!



Contact
daniel.penning@embeff.com
Phone +49 (451) 16088698



Sources

[1] Free C++98 Standard Draft
- http://www.lirmm.fr/~ducour/Doc-objets/ISO+IEC+14882-1998.pdf

[2] Size of enum implementation defined
- https://timsong-cpp.github.io/cppwp/n4659/dcl.enum#7

[3] ARM gcc 4.9 series reproducibly corrupts Register R7
- https://bugs.launchpad.net/gcc-arm-embedded/+bug/1527413

[4] GNU gcc Weekly Bug Summary
- https://gcc.gnu.org/bugzilla/page.cgi?id=gcc/weekly-bug-summary.html

[5] Arm Cortex-M4 Errata
- https://static.docs.arm.com/epm039104/30/Cortex-M4_Software_Developers_Errata_Notice_v3.pdf