

Establishing Coding Requirements for Non-Safety Critical C++ Systems

David Svoboda (Presenter)

Aaron Ballman (PI)



Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

CERT® is a registered mark of Carnegie Mellon University.

DM-0004094



Problem Statement and Focus

Writing secure C++ code is hard, existing coding standards are insufficient

MISRA C++:2008 and JSF++ (2005) focus on safety-critical systems; outdated

- CERT rules focus on modern concerns: C++11 and C++14.
 - Concurrency, lambdas, and other modern, high-impact C++ features

C++ Core Guidelines (2015) are modern, but subset the language; e.g.,

- ES.75: Avoid `do` statements
- I.11: Never transfer ownership by a raw pointer (\mathbb{T}^*)
- CERT rules do not subset the C++ language
 - Encourages adoption within legacy code bases as well as new

Enforceability of the rules is desirable.

- Demonstrate implementing checkers to help strengthen and enforce rules

Do not replicate rules from the CERT C Coding Standard

Our Results: Checkers



Contributed 15 new checkers to the Clang open source compiler (the C/C++ frontend to the LLVM compiler infrastructure)

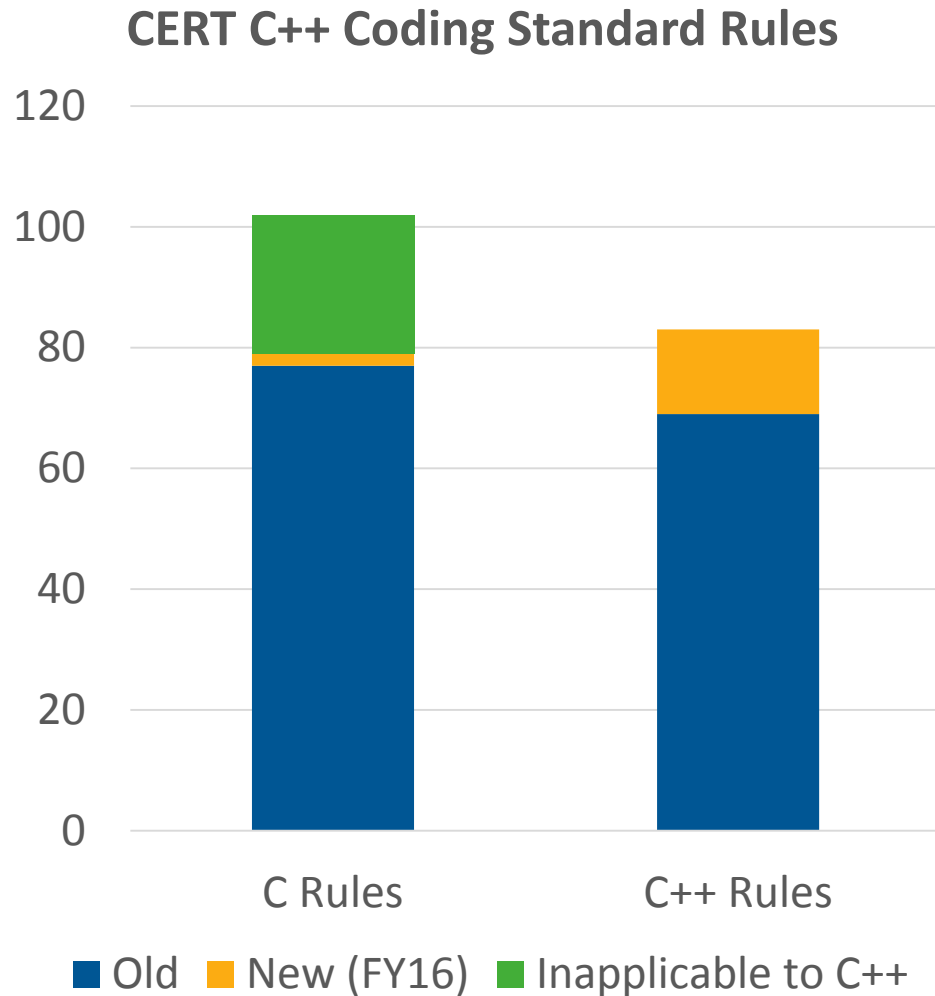
Clang community has shown significant interest in CERT's contributions

- Community members are making their own contributions based on our rules
- Demonstrated a desire to make it easier to enable all checks for CERT rules

Clang is used by 10s of millions of programmers to write 100s of millions of apps that are used by billions of users

- Primary compiler for MacOS, iOS, FreeBSD
- Supported by Microsoft Visual Studio, Linux

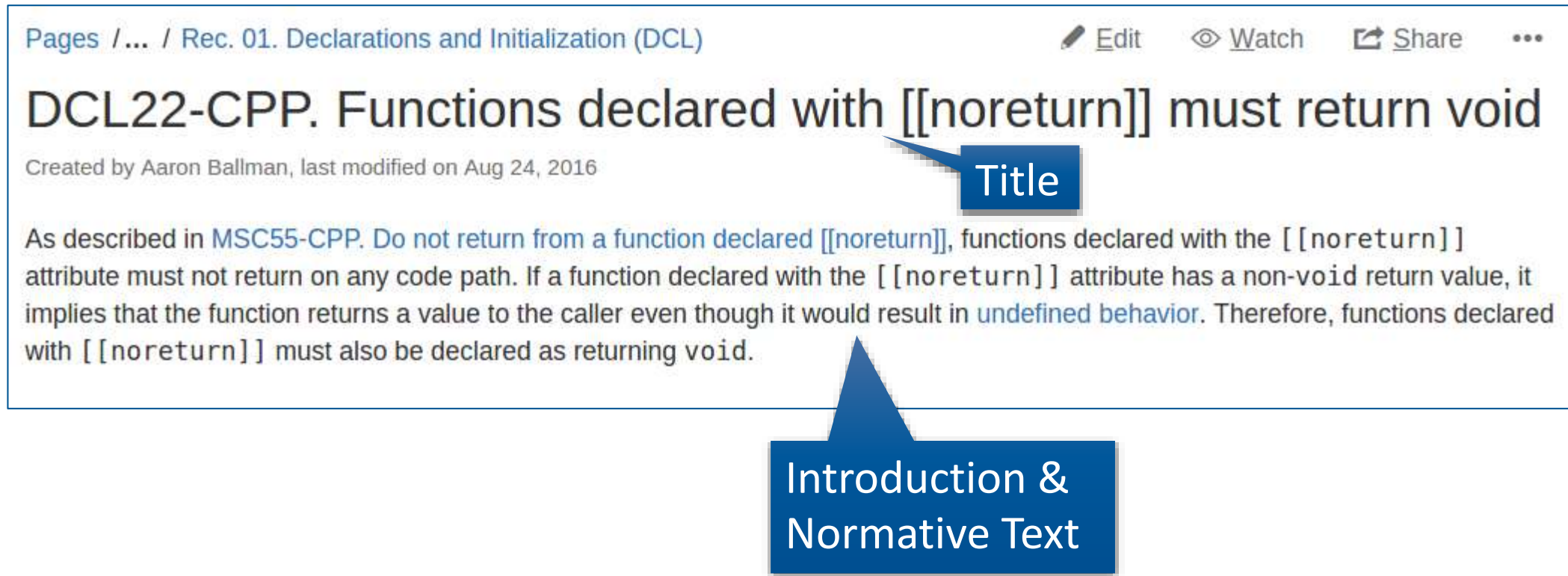
Our Results: Rules



1. Declarations and Initialization (DCL)
2. Expressions (EXP)
3. Integers (INT)
4. Containers (CTR)
5. Characters and Strings (STR)
6. Memory Management (MEM)
7. Input Output (FIO)
8. Exceptions and Error Handling (ERR)
9. Object Oriented Programming (OOP)
10. Concurrency (CON)
11. Miscellaneous (MSC)

All rules were heavily modified

Our Results: Rule Organization



The screenshot shows a web page for a rule titled "DCL22-CPP. Functions declared with `[[noreturn]]` must return void". The page includes a breadcrumb trail "Pages / ... / Rec. 01. Declarations and Initialization (DCL)", action buttons for "Edit", "Watch", "Share", and a menu icon. Below the title is the creation information: "Created by Aaron Ballman, last modified on Aug 24, 2016". The main text describes the rule's purpose and implications. Two blue callout boxes with white text are overlaid on the image: one pointing to the title and another pointing to the main text.

Pages / ... / Rec. 01. Declarations and Initialization (DCL) Edit Watch Share ...

DCL22-CPP. Functions declared with `[[noreturn]]` must return void

Created by Aaron Ballman, last modified on Aug 24, 2016

As described in [MSC55-CPP](#). Do not return from a function declared `[[noreturn]]`, functions declared with the `[[noreturn]]` attribute must not return on any code path. If a function declared with the `[[noreturn]]` attribute has a non-void return value, it implies that the function returns a value to the caller even though it would result in [undefined behavior](#). Therefore, functions declared with `[[noreturn]]` must also be declared as returning void.

Title

Introduction & Normative Text



Noncompliant Code Example

In this noncompliant code example, the function declared with `[[noreturn]]` claims to return an `int`:

```
#include <cstdlib>

[[noreturn]] int f() {
    std::exit(0);
    return 0;
}
```

Noncompliant Code
Don't try this at home!

This example does not violate [MSC55-CPP. Do not return from a function declared `\[\[noreturn\]\]`](#) because `std::exit()` is declared `[[noreturn]]`, so the `return 0;` statement can never be executed.

Compliant Solution

Because the function is declared `[[noreturn]]`, and no code paths in the function allow for a return in order to comply with [MSC55-CPP. Do not return from a function declared `\[\[noreturn\]\]`](#), the compliant solution declares the function as returning `void` and elides the explicit return statement:

```
#include <cstdlib>

[[noreturn]] void f() {
    std::exit(0);
}
```

Compliant Code
Fixes noncompliant code.



Risk Assessment

A function declared with a non-void return type and declared with the `[[noreturn]]` attribute is confusing to consumers of the function because the two declarations are conflicting. In turn, it can result in misuse of the API by the consumer or can indicate an implementation bug by the producer.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
DCL22-CPP	Low	Unlikely	Low	P3	L3

Automated Detection

Tool	Version	Checker	Description
Clang	3.9	-Winvalid-noreturn	



Related Vulnerabilities

Search for [vulnerabilities](#) resulting from the violation of this rule on the [CERT website](#).

Related Guidelines

SEI CERT C++ Coding Standard

MSC54-CPP. Value-returning functions must return a value from all exit paths
MSC55-CPP. Do not return from a function declared `[[noreturn]]`

Bibliography

[[ISO/IEC 14882-2014](#)]

Subclause 7.6.3, "Noreturn Attribute"

Our Process

- ISO WG21 (C++ Standards Committee)
- ISO C++14 Standard
- C++ Books
- MITRE CVEs
- CERT Vulnerability Database



Pages / ... / Rule 08. Exceptions and Error Handling (ERR) Edit Watch Share ...

ERR52-CPP. Do not use `setjmp()` or `longjmp()`

Created by Fred Long, last modified by Sandy Shrum about 3 hours ago

The C standard library facilities `setjmp()` and `longjmp()` can be used to simulate throwing and catching exceptions. However, these facilities bypass automatic resource management and can result in [undefined behavior](#), commonly including resource leaks, and [denial-of-service attacks](#).



```
E:\llvm\2015>clang-tidy -checks=-*,cert-* E:\Desktop\test1.cpp -- -std=c++14
2 warnings generated.
E:\Desktop\test1.cpp:7:7: warning: do not call 'setjmp'; consider using exception
handling instead [cert-err52-cpp]
    if (setjmp(env) == 0) {
        ^
```



THE END

