Single-entry Single-exit

A closer look at the rule…

Lightning talk @ Meeting C++ (2022-08-31)

Mirosław Opoka (opoka.tech)

The rule

“subprograms should have a single entry and a single exit only”

IEC 61508 (1997) → Functional safety of electrical/ electronic/ programmable electronic safety-related systems.

Part 7 → Overview of techniques and measures

Section C.2.9 → Modular approach (page 69),
Reference: “Structured Design” book from 1979

The rule background

- the idea to **simplify program flow** can be traced back to "Notes on structure programming" by E. W. Dijkstra (1970),

- back then most programming was done in **assembler**, **Fortran** or **Cobol** (there was no C (1972), not to mention C++ (1985)),

- **jumping around** in the code was a standard (goto),

- this style led to **problems with readability** and thus with **maintainability** of the code,

- the rule was postulated to address this problem

Selected standards and the rule

- **IEC 61508**: *Subprograms should have a single entry and a single exit only,*
- **MISRA**: *A function should have a single point of exit at the end,*
- **Google style guide**: not mentioned,
- **Autosar**: *Multiple points of exit are permitted by AUTOSAR C++ Coding Guidelines,*
- **ISO C++ coding guidelines**: *Don’t insist to have only a single `return`-statement in a function*

The rule in practice

```
Result processMessage(const System &system, const Message &msg)
{
    Result retValue = Result::UNKNOWN;

    if (system.isConnected())
    {
        if (system.isInitialized())
        {
            if (msg.isValid())
            {
                // system is connected, initialized and the message is valid
                // ... process the message
                retValue = Result::OK;
            }
            else
            {
                retValue = Result::INVALID_MESSAGE;
            }
        }
        else
        {
            retValue = Result::NOT_INITIALIZED;
        }
    }
    else
    {
        retValue = Result::NOT_CONNECTED;
    }

    return retValue;
}
```

```
Result processMessage(const System &system, const Message &msg)
{
    if (!system.isConnected())
    {
        return Result::NOT_CONNECTED;
    }

    if (!system.isInitialized())
    {
        return Result::NOT_INITIALIZED;
    }

    if (!msg.isValid())
    {
        return Result::INVALID_MESSAGE;
    }

    // system is connected, initialized and the message is valid
    // ... process the message
    return Result::OK;
}
```

The rule - pros and cons

	Single-entry/Single-exit	non Single-entry/Single-exit
Readability	quickly gets **too nested** and thus **harder to understand**	exiting early: - makes the rest of the code **less nested** and thus **easier to read**, - **frees "mental resources"** for following the code
Resources	**one place for freeing resources** (clean up at the exit)	resource allocation and **cleaning up must be taken care of** (for example using RAII)
Debugging	might be **easier to debug** (breakpoint at exit)	breakpoints at few places

References

Standards:

- IEC 61508-7 referencing “Structured Design” book from 1979
- MISRA C++ 2008
- Google C++ guide
- Autosar
- ISO C++ core guidelines

Other:

- Example of multi-entries and multi-exit code in Cobol
- “Notes on structured programming” by Prof.dr. E. W. Dijkstra