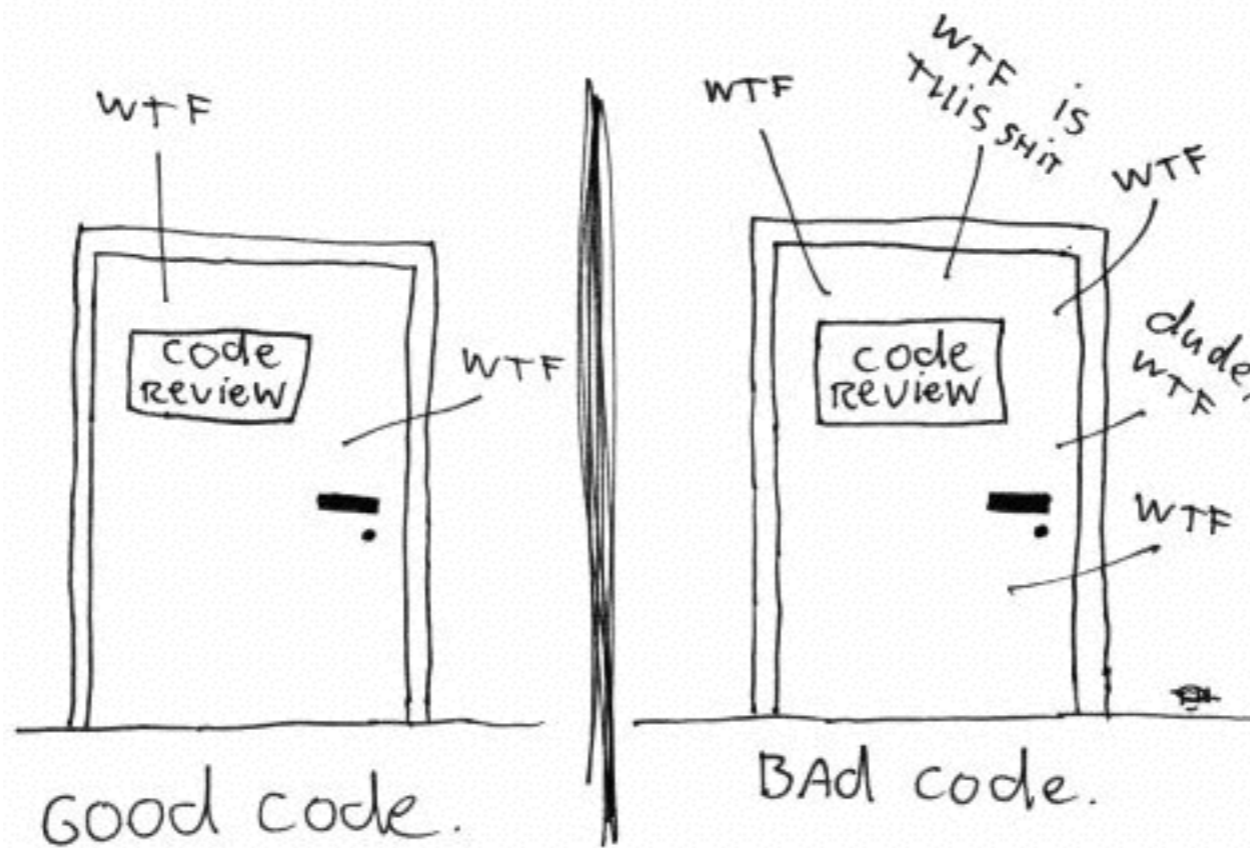


Writing good code

The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/MINUTE



(c) 2008 Focus Shift

Overview

- **Metrics**
- **Design by Contract**
- **Personal Software Process**
- **Clean Code**
- **Code Documentation**
- **Analyzers**
- **C++ in aviation**
- **Complimentary sources**

correct

extensible

maintainable

readable

**What is
“good”
code?**

robust

uncomplicated

consistent

reusable

Metrics

**Balanced
Scorecard**

McCabe

Halstead

**Code
Coverage**

LOC

DSQI

**Function
Points**

**Maintainability
Index**

...

My Metrics

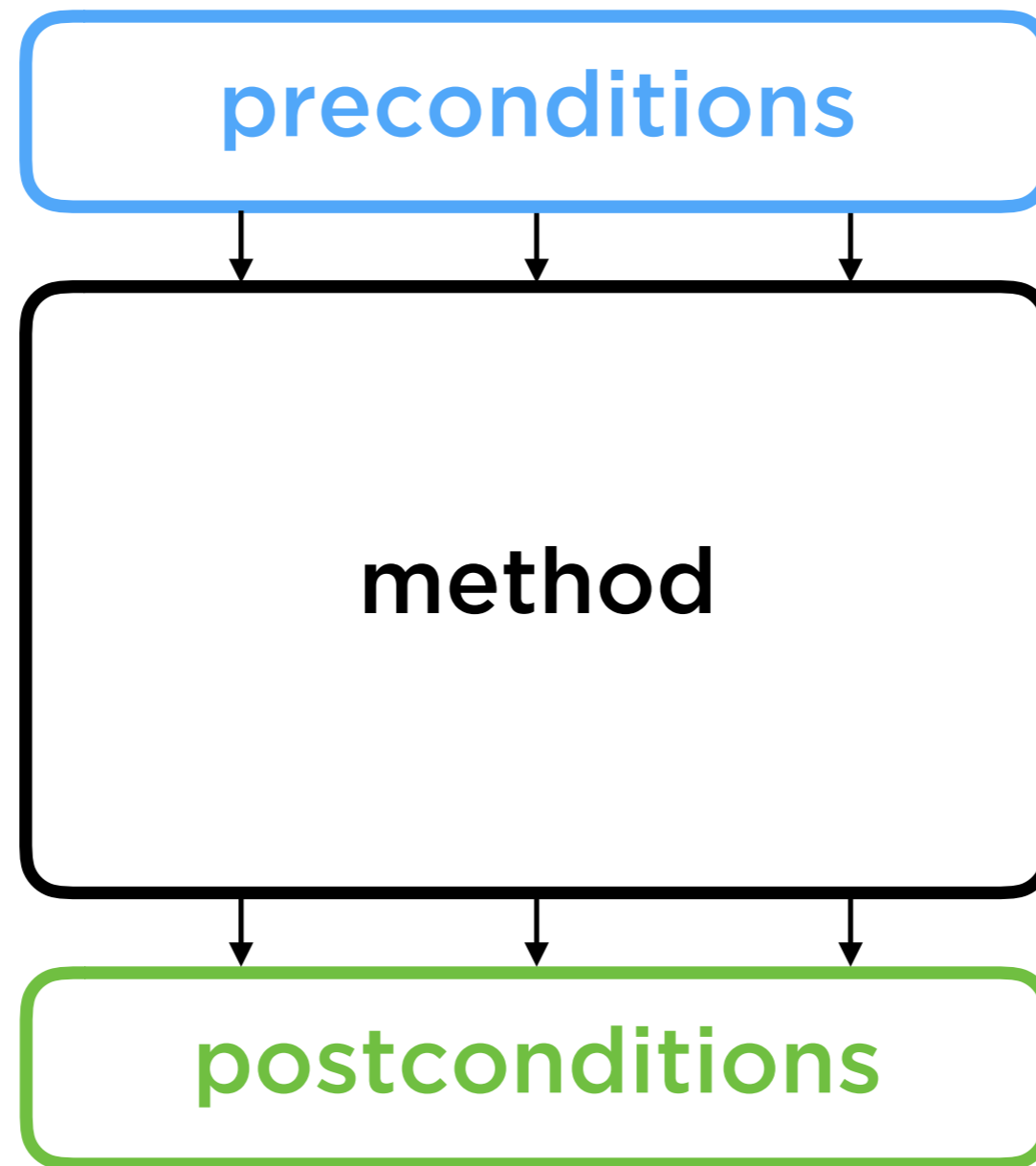
- **Methods \leq 20 LOC**
- **Classes (*.cpp) \leq 1000 LOC**
- **If-nesting levels \leq 4**

Design by Contract

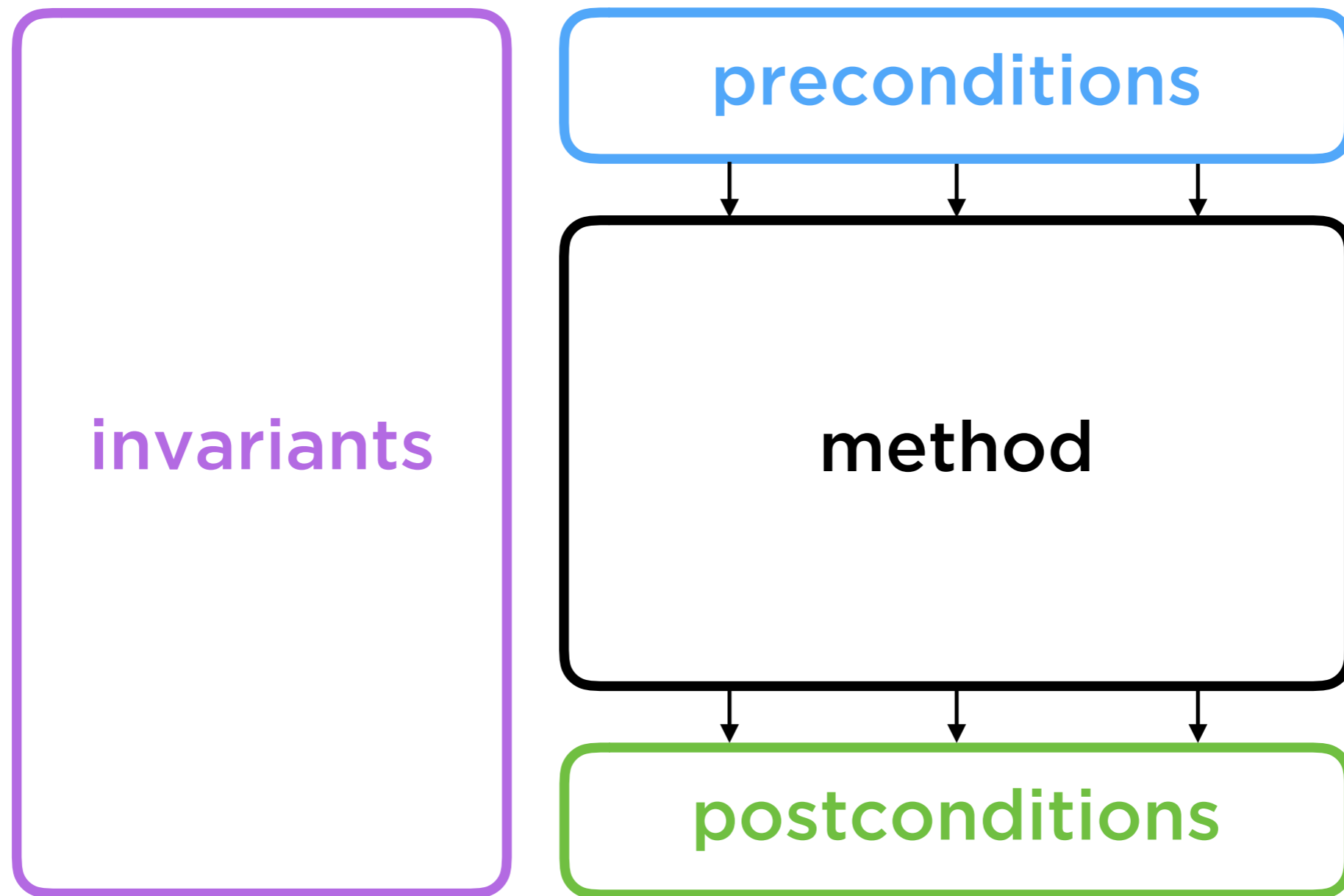
Contracts???



Design by Contract



Design by Contract



Example

(taken from my personal hobby project)

Star Wars X-Wing Simulator



Example:

Precondition

```
template <typename T>
std::vector<T> roll_dice(const unsigned int number_of_dice,
                       const std::initializer_list<T>& die)
{
    std::vector<T> roll;
    for (auto i = 0U; i < number_of_dice; ++i)
    {
        roll.push_back(roll_die(die));
    }
    return roll;
}
```

Example:

Precondition

```
template <typename T>
std::vector<T> roll_dice(const unsigned int number_of_dice,
                       const std::initializer_list<T>& die)
{
    assert(die.size() > 0);

    std::vector<T> roll;
    for (auto i = 0U; i < number_of_dice; ++i)
    {
        roll.push_back(roll_die(die));
    }
    return roll;
}
```

Let's talk about

assert(...)

Use static and dynamic assertions as sanity checks.

JPL Institutional Coding Standard for the C Programming Language (NASA)

+1 Add as Friend

`assert(...)`

+1 Add as Friend

`static_assert(...)`

Example:

Precondition

```
template <typename T>
std::vector<T> roll_dice(const unsigned int number_of_dice,
                       const std::initializer_list<T>& die)
{
    assert(die.size() > 0);

    std::vector<T> roll;
    for (auto i = 0U; i < number_of_dice; ++i)
    {
        roll.push_back(roll_die(die));
    }
    return roll;
}
```

Example:

Precondition

```
template <typename T>
std::vector<T> roll_dice(const unsigned int number_of_dice,
                        const std::initializer_list<T>& die)
{
    assert(die.size() > 0);
    if (die.size() == 0) return std::vector<T>();

    std::vector<T> roll;
    for (auto i = 0U; i < number_of_dice; ++i)
    {
        roll.push_back(roll_die(die));
    }
    return roll;
}
```

Example:

Postcondition

```
void squad::add_pilot_with_upgrades(const pilot& pilot, const
std::vector<upgrade>& upgrades)
{
    auto pilot_with_upgrades = std::make_pair(pilot, upgrades);
    _cost += combined_cost(pilot_with_upgrades);

    const auto position =
        std::lower_bound(_pilots_with_upgrades.begin(),
                        _pilots_with_upgrades.end(),
                        pilot_with_upgrades,
                        compare_pilot_skills);

    _pilots_with_upgrades.insert(position, pilot_with_upgrades);
}
```

Example:

Postcondition

```
void squad::add_pilot_with_upgrades(const pilot& pilot, const
std::vector<upgrade>& upgrades)
{
    auto pilot_with_upgrades = std::make_pair(pilot, upgrades);
    _cost += combined_cost(pilot_with_upgrades);

    const auto position =
        std::lower_bound(_pilots_with_upgrades.begin(),
                        _pilots_with_upgrades.end(),
                        pilot_with_upgrades,
                        compare_pilot_skills);

    _pilots_with_upgrades.insert(position, pilot_with_upgrades);

    assert(std::is_sorted(_pilots_with_upgrades.begin(),
                          _pilots_with_upgrades.end(),
                          compare_pilot_skills));
}
```

Example: Invariant

```
void squad::add_pilot_with_upgrades(const pilot& pilot, const
std::vector<upgrade>& upgrades)
{
    // ...

    assert(is_unique(_pilots_with_upgrades));
    assert(combined_cost(_pilots_with_upgrades) == _cost);
}
```

PSP

Personal Software Process

Came from Six Sigma



PSP Overview

PSP0, PSP0.1

Process discipline and measurement

PSP1, PSP1.1

Estimating and planning

PSP2, PSP2.1

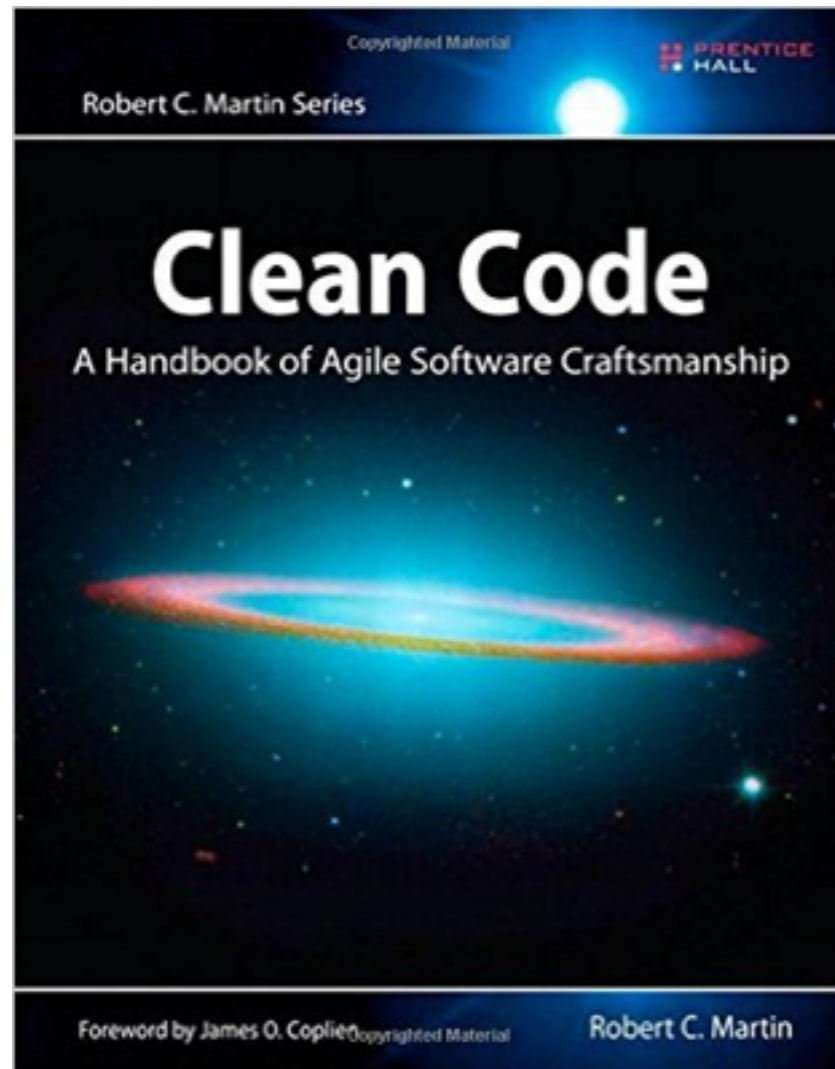
Quality management and design



PSP Checklist

- Code Review
 - range-based for
 - QPointer access
 - underflow/overflow
 - ...
- Analyzer run
- Release/Debug build all
- Requirements/Design/Tests complete
- Code Documentation
- ...

Clean Code



The seven grades

0) Black grade

1) Red grade

2) Orange grade

3) Yellow grade

4) Green grade

5) Blue grade

6) White grade

Keep it simple, ~~stupid~~ smart (KISS)

Don't Repeat Yourself (DRY)

You Ain't Gonna Need It (YAGNI)

Pfadfinderregel

Single Responsibility Principle (SRP)

Law of Demeter

Root Cause Analysis

Single Level of Abstraction (SLA)

Favour Composition over Inheritance (FCoI)

Interface segregation principle

Automatisierte Integrationstests

Boy Scout Rule



*“Always leave the
campground cleaner than
you found it.”*

Single Level of Abstraction (SLA)

```
void game::combat_phase()
{
    for (auto attacker : _all_pilots)
    {
        pilot* defender = nullptr;
        if (_player1.contains(*attacker))
        {
            defender = *std::min_element(_player2.begin(),
                                         _player2.end(),
                                         hull_and_shield_value);
        }
        else
        {
            defender = *std::min_element(_player1.begin(),
                                         _player1.end(),
                                         hull_and_shield_value);
        }

        auto attack_dice = roll_attack_dice(*attacker);
        modify_attack_dice(attack_dice);
        // ...
    }
}
```


Single Level of Abstraction (SLA)

```
void game::combat_phase()
{
    for (auto attacker : _all_pilots)
    {
        auto defender = declare_target(*attacker);

        auto attack_dice = roll_attack_dice(*attacker);
        modify_attack_dice(attack_dice);

        auto defense_dice = roll_defense_dice(defender);
        modify_defense_dice(defense_dice);

        const auto result = compare_results(attack_dice,
                                            defense_dice);
        deal_damage(defender, result);
    }
}
```

Single Responsibility Principle (SRP)

```
pilot& game::declare_target(pilot& attacker)
{
    pilot* target = nullptr;
    if (_player1.contains(attacker))
    {
        target = *std::min_element(_player2.begin(),
                                   _player2.end(),
                                   hull_and_shield_value);
    }
    else
    {
        target = *std::min_element(_player1.begin(),
                                   _player1.end(),
                                   hull_and_shield_value);
    }
    return *target;
}
```

Single Responsibility Principle (SRP)

```
pilot& game::declare_target(pilot& attacker)
{
    if (_player1.contains(attacker))
    {
        return ai::declare_target(attacker, _player2);
    }
    else
    {
        return ai::declare_target(attacker, _player1);
    }
}
```

Favour Composition over Inheritance (FCoI)

```
class pilot  
{  
    // ...  
};
```

Favour Composition over Inheritance (FCoI)

```
class pilot
{
public:
    // ...

    void move(double x_direction, double y_direction);

private:
    struct position
    {
        double x = 0.0;
        double y = 0.0;
    } _position;
};
```

Favour Composition over Inheritance (FCoI)

```
class movable
{
public:
    void move(double x_direction, double y_direction);

private:
    struct position
    {
        double x = 0.0;
        double y = 0.0;
    } _position;
};

class pilot : public movable
{
public:
    // ...
};
```

Favour Composition over Inheritance (FCoI)

```
class object
{
protected:
    struct position
    {
        double x = 0.0;
        double y = 0.0;
    } _position;
};

class movable : public object
{
public:
    void move(double x_direction, double y_direction);
};

class asteroid : public object
{
};

class pilot : public movable{
public:
    // ...
};
```

Favour Composition over Inheritance (FCoI)

```
class game_board
{
public:
    void add_pilot(const pilot& pilot, const double x, const double y);
    void move_pilot(const pilot& pilot, Maneuver direction);

    // ...

private:
    struct position
    {
        double x = 0.0;
        double y = 0.0;
    };

    std::map<pilot*, position> _pilot_positions;
    std::map<asteroid*, const position> _asteroid_positions;
};
```


Code

Documentation

Code Documentation:

Elaborate version

```
/*! Rolls a given number of attack dice.  
 *  
 * @param number_of_dice the number of attack dice to roll  
 * @return a vector with the result of the attack dice  
 */  
std::vector<Attack> roll_attack_dice(unsigned int number_of_dice);
```

Code Documentation:

Compact version

```
/*! Rolls a given number of attack dice.  
 */  
std::vector<Attack> roll_attack_dice(unsigned int number_of_dice);
```

Code Documentation: At least do classes!

```
/*!  
 * This class represents a single pilot card in Star Wars X Wing.  
 */  
class pilot  
{  
    //...  
};
```

Code Documentation

<http://doc.qt.io/>

The screenshot shows the Qt Documentation website for the QMainWindow Class. The page is structured as follows:

- Page Header:** Qt logo, navigation links (Download, Device Creation, Application Development, Services, Developers), and icons for Partners and Blog.
- Navigation:** Wiki, Documentation, Forum, Bug Reports, Code Review.
- Search:** Google Custom Search bar.
- Breadcrumbs:** Qt 5.6 > Qt Widgets > C++ Classes > QMainWindow
- Section Header:** QMainWindow Class
- Description:** The QMainWindow class provides a main application window. [More...](#)
- Code Snippets:**

Header:	#include <QMainWindow>
qmake:	QT += widgets
Inherits:	QWidget
- Links:** > List of all members, including inherited members
- Public Types:**

enum	DockOption { AnimatedDocks, AllowNestedDocks, AllowTabbedDocks, ForceTabbedDocks, VerticalTabs, GroupedDragging }
flags	DockOptions
- Properties:**

> animated : bool	> iconSize : QSize
> dockNestingEnabled : bool	> tabShape : QTabWidget::TabShape
> dockOptions : DockOptions	> toolButtonStyle : Qt::ToolButtonStyle
> documentMode : bool	> unifiedTitleAndToolBarOnMac : bool
- Inheritance:**
 - > 58 properties inherited from QMainWindow
 - > 1 property inherited from QObject
- Public Functions:**

	QMainWindow(QWidget *parent = Q_NULLPTR, Qt::WindowFlags flags = Qt::WindowFlags())
	~QMainWindow()
- Right Sidebar:**
 - Contents:** Public Types, Properties, Public Functions, Public Slots, Signals, Reimplemented Protected Functions, Detailed Description (Qt Main Window Framework, Creating Main Window Components, Storing State).
 - Reference:** All Qt C++ Classes, All QML Types, All Qt Modules, Qt Creator Manual, All Qt Reference Documentation.
 - Getting Started:** Getting Started with Qt, What's New in Qt 5, Examples and Tutorials, Supported Platforms.

Static Code Analyzers

C++ Static Code Analyzers

- CppCheck
- PC lint
- Clang Static Analyzer
- MSVS /Analyze
- PVS Studio
- ...

**But which one is the best/
which one should I take?**



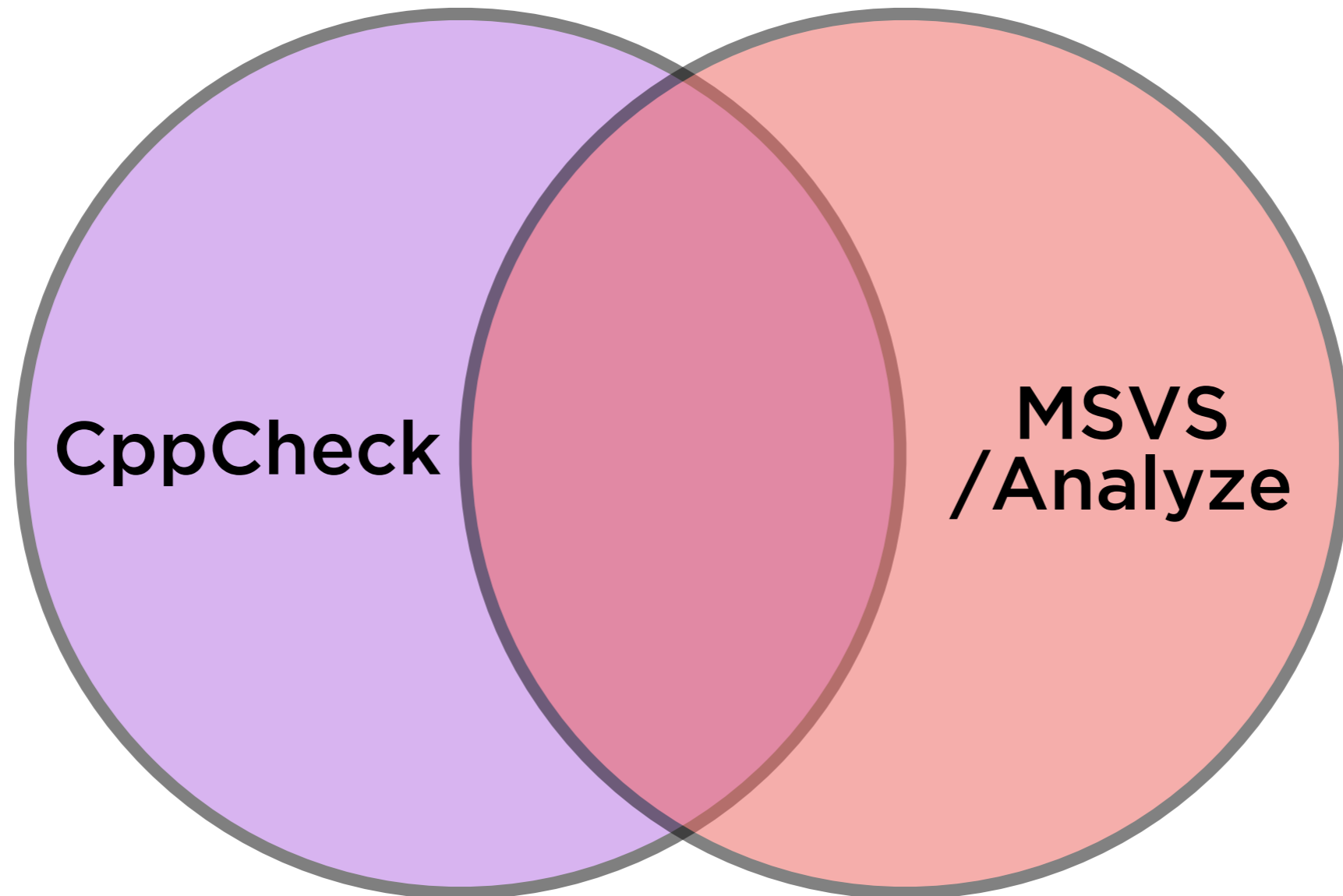
STORCK®

nimm 2

Caramelos rellenos de naranja y limón con 9 vitaminas
Orange and lemon candies with 9 essential vitamins
Orangen- und Zitronenbonbons mit wertvollen Vitaminen

Con zumo de fruta en el relleno
With fruit juice and dextrose in the filling
Mit Fruchtsaft und Traubenzucker in der Füllung

**Let one find bugs the other
does not and vice versa!**



C++ in Aviation



DO-178C

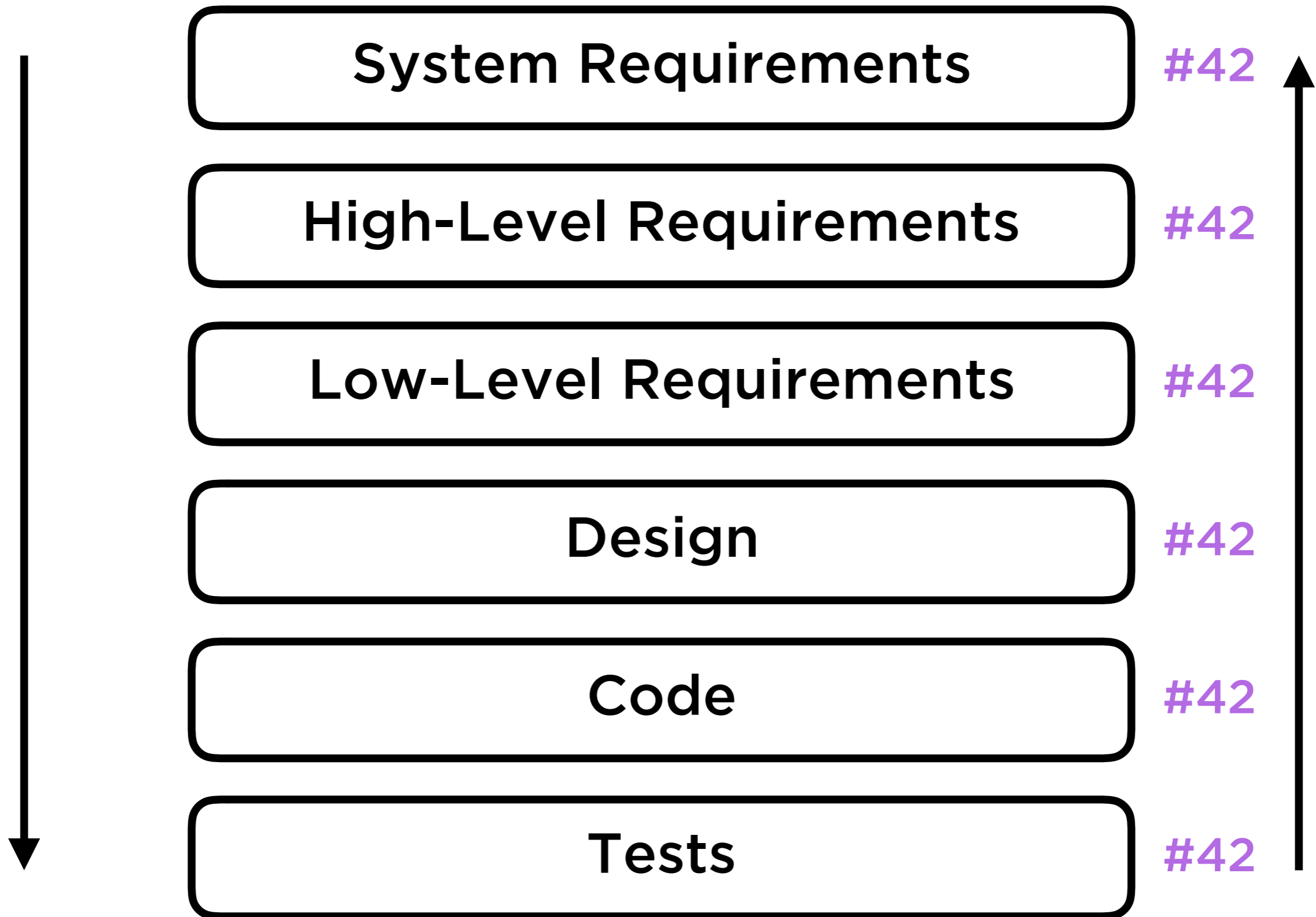
**Software Considerations in Airborne
Systems and Equipment Certification**

Design Assurance Level

- Level E: No safety effect
- Level D: Minor
- Level C: Major
- Level B: Hazardous
- Level A: Catastrophic

Traceability





Validation & Verification

**Verification
(Tests)**



System Requirements

High-Level Requirements

Low-Level Requirements

Design

Code



**Validation
(Reviews)**

C++ Coding Standards

- MISRA C++
- JSF
- HIC++
- Google C++ Style Guide
- C++ Core Guidelines



The Motor Industry Software Reliability Association

~~Multiple
return~~

~~Heap~~

~~Recursion~~

~~using
directives~~

~~unions~~

~~Multiple
break/
continue~~

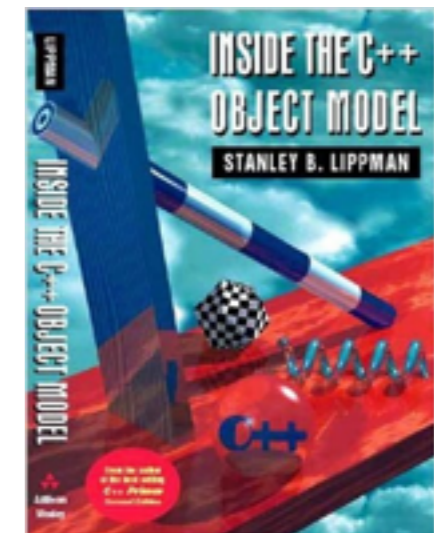
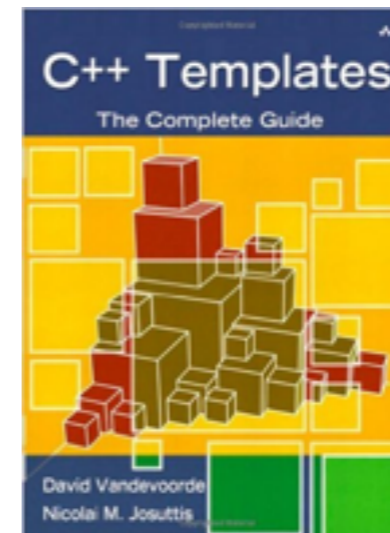
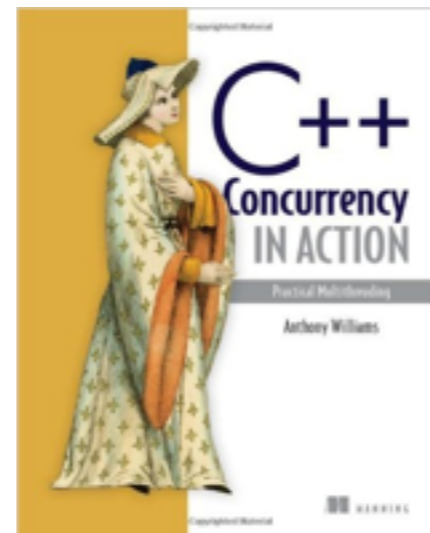
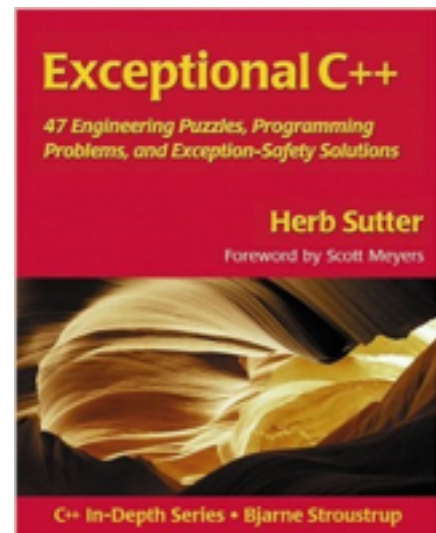
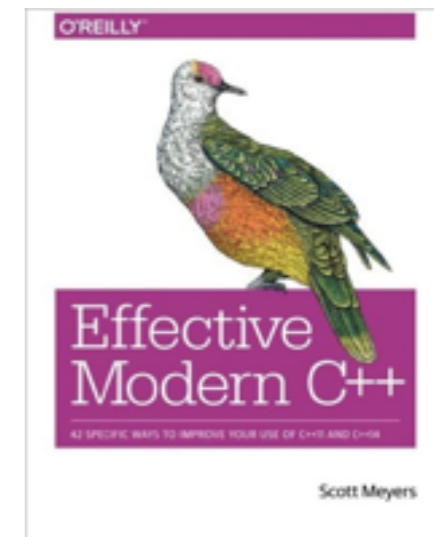
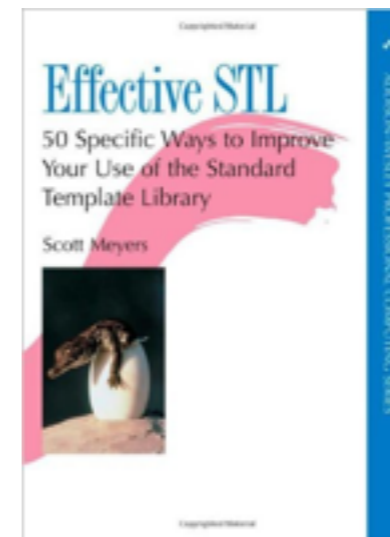
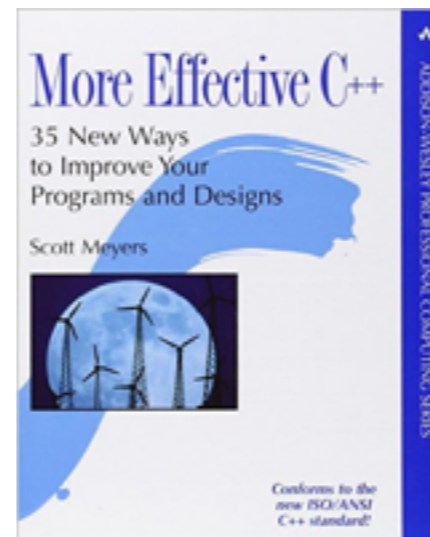
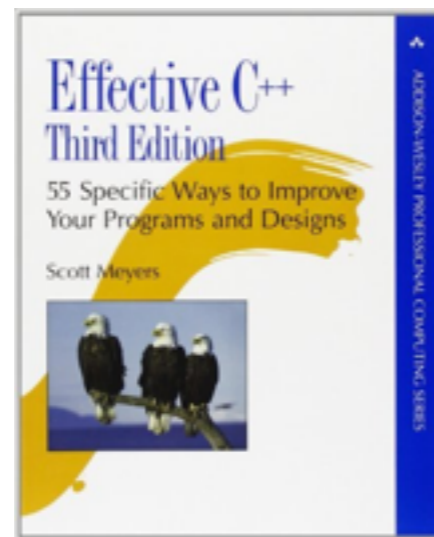
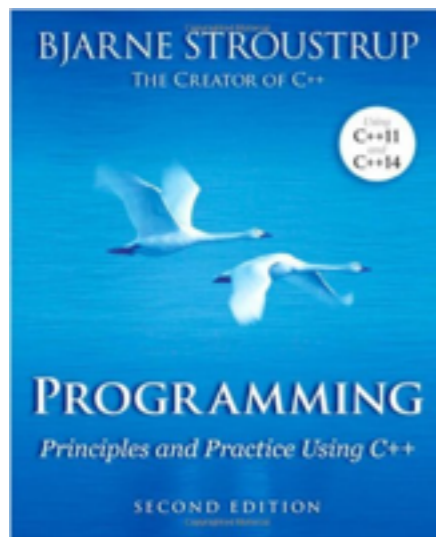
~~function
macros~~

~~dead
code~~

Complimentary Sources

Literature

<http://stackoverflow.com/questions/388242/the-definitive-c-book-guide-and-list>



C++ Core Guidelines

<https://github.com/isocpp/CppCoreGuidelines>



Bjarne



Herb

C++ Core Guidelines

<https://github.com/isocpp/CppCoreGuidelines>

C.130: Redefine or prohibit copying for a base class; prefer a virtual *clone* function instead

Copying a base is usually slicing. If you really need copy semantics, copy deeply: Provide a virtual *clone* function that will copy the actual most-derived type and return an owning pointer to the new object, and then in derived classes return the derived type (use a covariant return type).

C++ Core Guidelines

<https://github.com/isocpp/CppCoreGuidelines>

C++ Core Guidelines

April 23, 2016

Editors:

- [Bjarne Stroustrup](#)
- [Herb Sutter](#)

This document is a very early draft. It is inkorrekt, incompleat, and pÅuÃoorly formatted. Had it been an open source (code) project, this would have been release 0.6. Copying, use, modification, and creation of derivative works from this project is licensed under an MIT-style license. Contributing to this project requires agreeing to a Contributor License. See the accompanying [LICENSE](#) file for details. We make this project available to "friendly users" to use, copy, modify, and derive from, hoping for constructive input.

Comments and suggestions for improvements are most welcome. We plan to modify and extend this document as our understanding improves and the language and the set of available libraries improve. When commenting, please note [the introduction](#) that outlines our aims and general approach. The list of contributors is [here](#).

Problems:

- The sets of rules have not been thoroughly checked for completeness, consistency, or enforceability.
- Triple question marks (???) mark known missing information
- Update reference sections; many pre-C++11 sources are too old.
- For a more-or-less up-to-date to-do list see: [To-do: Unclassified proto-rules](#)

You can read an [explanation of the scope and structure of this Guide](#) or just jump straight in:

- [In: Introduction](#)
- [P: Philosophy](#)
- [I: Interfaces](#)
- [F: Functions](#)
- [C: Classes and class hierarchies](#)
- [Enum: Enumerations](#)
- [R: Resource management](#)
- [ES: Expressions and statements](#)
- [PER: Performance](#)
- [CP: Concurrency](#)
- [E: Error handling](#)
- [Con: Constants and immutability](#)
- [T: Templates and generic programming](#)
- [CPL: C-style programming](#)

C++ Subreddit

<https://www.reddit.com/r/cpp>

The image shows a screenshot of the C++ subreddit page on Reddit. The page is titled "C++ Subreddit" and includes the URL "https://www.reddit.com/r/cpp". The main content area displays a list of posts, with the top post being "Who's Hiring C++ Devs - Q2 2016" (self.cpp) submitted 1 month ago by STL [M], which is a sticky post with 159 upvotes and 54 comments. Other posts include "Template SFINAE & type-traits -- quick overview", "Boost MPL dead?", "Keeping track of C++", "Succinct and helpful C++ template compilation errors", "Generating Python Bindings with Clang", "New CLion EAP", "Conference Submission Deadlines", "Andrei Alexandrescu on C++ concepts", "Pointer Overflow Checking", "What's New in ReSharper C++ 2016.1", "CMake template file for C++ project", "ra-ra, a C++14 dense array / expression template library", "Jacksonville C++ Core Language Meeting Report", "Is Effective STL a good book to read now (in addition to Effective C++ and Modern Effective C++)?", "Web frameworks for monitoring/controlling executables?", and "C++ Weekly - Ep11 std::future Part 2".

On the right side of the page, there is a search bar, a login form with fields for "username" and "password", and a "login" button. Below the login form is an advertisement for "/r/unconventionalmakeup" with a "discuss this ad on reddit" link and buttons for "Submit a new link" and "Submit a new text post".

Below the advertisement, there is a section for the "cpp" subreddit, showing 36,395 readers and ~25 users online. It includes a description: "Discussions, articles, and news about the C++ programming language or programming in C++." and a link to "r/cpp_questions or StackOverflow".

There are also sections for "Get started", "Reference", and "Books". The "Get started" section lists "The C++ Standard Home" as a resource. The "Reference" section lists "cppreference.com". The "Books" section mentions a list of books on Stack Overflow.

At the bottom of the page, there is a small image of a hand holding a paintbrush, with the text "/r/finishing" overlaid on it.



C++ Talks

CppCon

[https://
www.youtube.com/user/
CppCon](https://www.youtube.com/user/CppCon)

CppCon 2015
105 videos • 38,308 views • Last updated on Dec 14, 2015

- 1 CppCon 2015: Björn Stroustrup "Writing Good C++14" 1:40:46
- 2 CppCon 2015: Herb Sutter "Writing Good C++14... By Default" 1:29:06
- 3 CppCon 2015: Sean Parent "Better Code: Data Structures" 1:04:00
- 4 CppCon 2015: Chandler Carruth "Tuning C++: Benchmarks, and CPUs, and Compilers Oh My!" 1:29:54
- 5 CppCon 2015: Ericniebler "Ranges for the Standard Library" 1:34:07
- 6 CppCon 2015: Vittorio Romeo "Implementation of a component-based entity system in modern C++" 1:04:46
- 7 CppCon 2015: Joel Falou PART 1 "Expression Templates - Past, Present, Future" 1:06:06
- 8 CppCon 2015: Scott Warden "Memory and C++ debugging at Electronic Arts" 57:50
- 9 CppCon 2015: André Bergner "Faster Complex Numbers" 32:34
- 10 CppCon 2015: Louis Dionne "C++ Metaprogramming: A Paradigm Shift" 1:08:11
- 11 CppCon 2015: Pramod Gupta "C++ Multi-dimensional Arrays..." 38:40
- 12 CppCon 2015: Steve Carroll - Ayman Shouky "What's New in Visual C++ 2015 and Future Directions" 1:03:37
- 13 CppCon 2015: Arthur O'Dwyer "Lambdas from First Principles: A Whitehead Tour of C++" 58:54

Meeting C++

[https://
www.youtube.com/user/
MeetingCPP](https://www.youtube.com/user/MeetingCPP)

Meeting C++ 2015
37 videos • 3,007 views • Last updated on Apr 26, 2015

- 1 Meeting C++ 2015: keynote speakers interview 11:31
- 2 Meeting C++ Lightning Talks - Vittorio Romeo - 'static_if' in C++14 9:36
- 3 Meeting C++ Lightning Talks - Vittorio Romeo - 'Meaningful casts' 11:40
- 4 Meeting C++ Lightning Talks - Gbén Holmes - Modern special function regular abstraction 12:14
- 5 Günter Obstschäpzig - 15 years of Poco C++ Libraries - Meeting C++ 2015 Lightning Talks 11:21
- 6 John Melas - boost::string_ref - Meeting C++ 2015 Lightning Talks 8:48
- 7 Nicolai Bahmann - Choosing the correct vectorization method - Meeting C++ 2015 Lightning Talks 10:59
- 8 Ben Huckvale - A perfect async RPC framework? - Lightning Talks - Meeting C++ 2015 8:40
- 9 Christoph Wankke - Generic Binding - Meeting C++ 2015 Lightning Talks 11:56
- 10 J. Falou & E. Alligand - Introduction to brigand - Meeting C++ 2015 Lightning Talks 11:15
- 11 Keep your code sane with clang-tidy - Daniel Jasper - Meeting C++ 2015 Lightning Talks 13:22
- 12 Understanding Compiler Optimization - Chandler Carruth - Opening Keynote Meeting C++ 2015 1:50:15

C++Now

[https://
www.youtube.com/user/
BoostCon](https://www.youtube.com/user/BoostCon)

C++Now 2015
47 videos • 106 views • Last updated on Mar 14, 2015

- 1 Andrew Sutton's C++Now 2015 Keynote: generic-programming-with-concepts 1:37:43
- 2 Tiago Quintino's C++Now 2015 Keynote: Numerical Weather Prediction: Facing the Future with C++ 1:31:22
- 3 Charles Bay: Your CPU is binary 1:35:24
- 4 Pablo Colapinto: Functional Geometry: Producing Pure Spaces 1:25:07
- 5 Ben Deane: Testing Battle.net (before deploying to millions of players) 1:16:15
- 6 Jean-Louis Leroy: Yommi1: Open Multi-Methods for C++11 1:30:33
- 7 Diego Rodriguez-Losada: Blicode, a C/C++ dependency manager with a hosting service 1:27:39
- 8 Scott Schurr: conestpr: C++ At Compile Time 1:33:50
- 9 David Stone: Type Deduction in C++14 1:31:41
- 10 Agustín Berge, Thomas Helles, Hartmut Kaiser: Back to the Future 1:34:51
- 11 Boris Kalpakov: New Build System for New C++ 1:49:36
- 12 John Lakos: Large-Scale C++: Advanced Levelization Techniques, Part I 1:29:04
- 13 John Lakos: Large-Scale C++: Advanced Levelization Techniques, Part II 1:23:22

Questions?