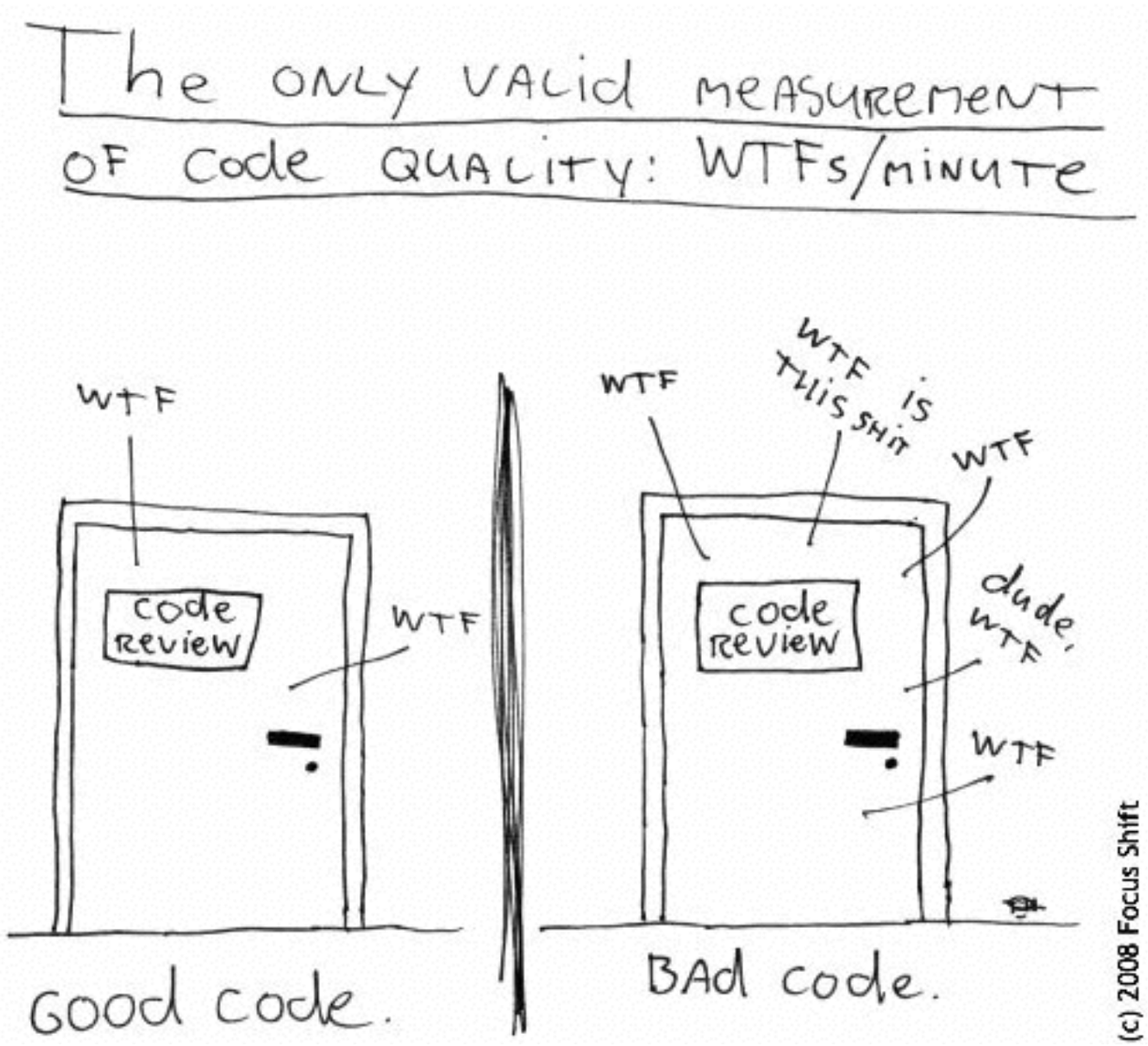
Writing good code



Overview

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

correct

extensible

maintainable

What is “good” code?

readable

robust

uncomplicated

consistent

reusable

Metrics

Balanced Scorecard	McCabe	Halstead
Code Coverage	LOC	DSQI
Function Points	Maintainability Index	...

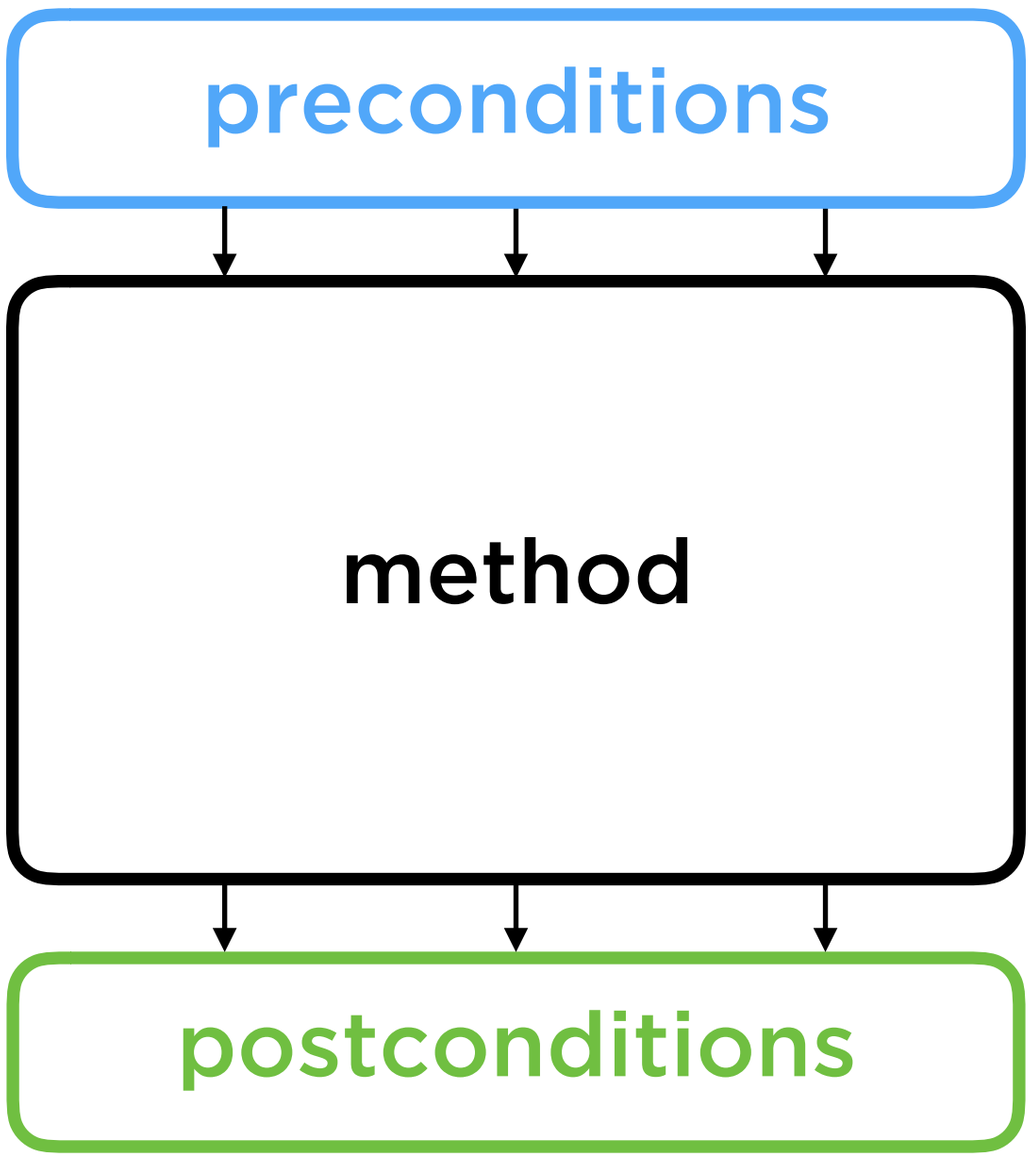
My Metrics

- Methods <= 20 LOC
- Classes (*.cpp) <= 1000 LOC
- If-nesting levels <= 4

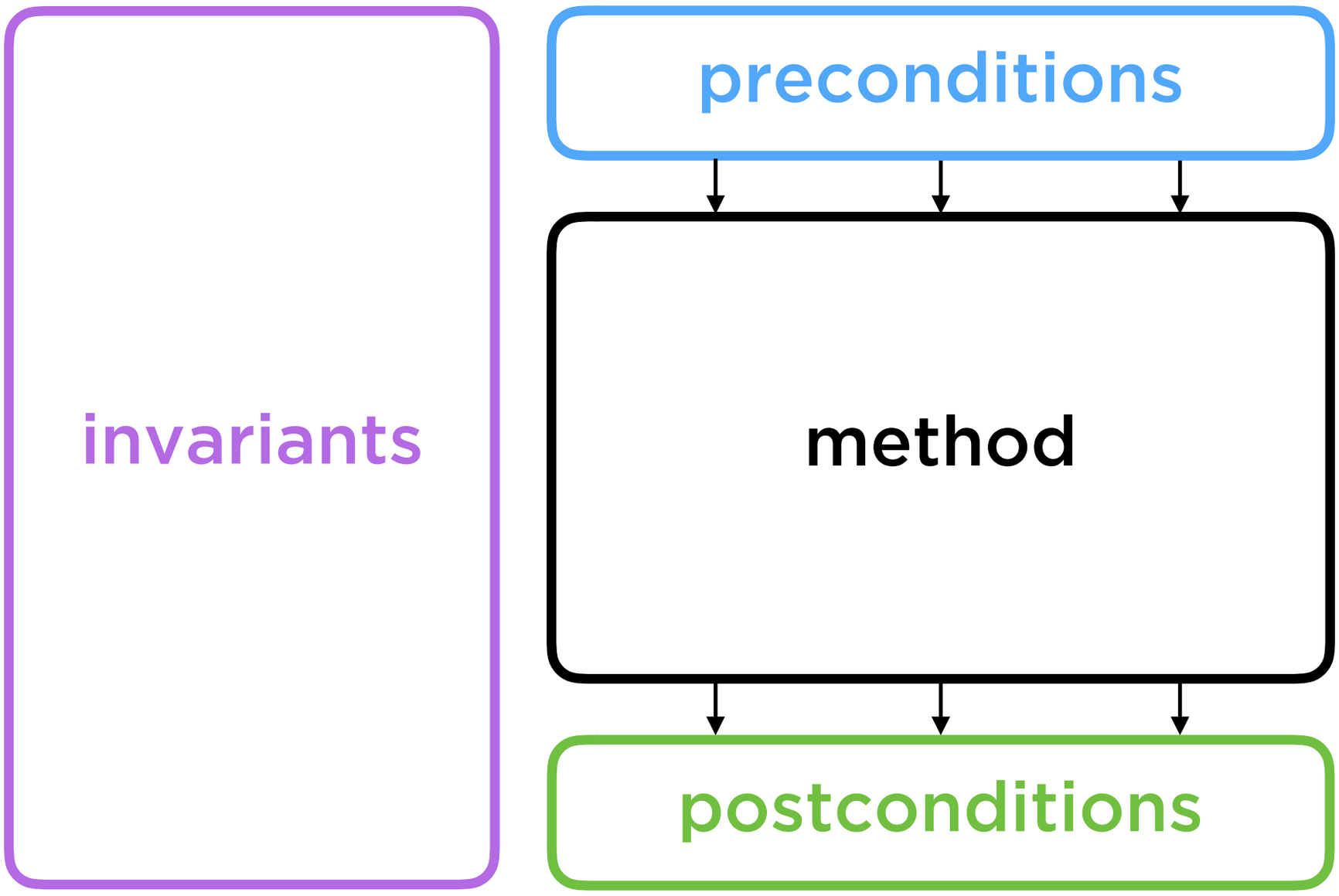
Design by Contract

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

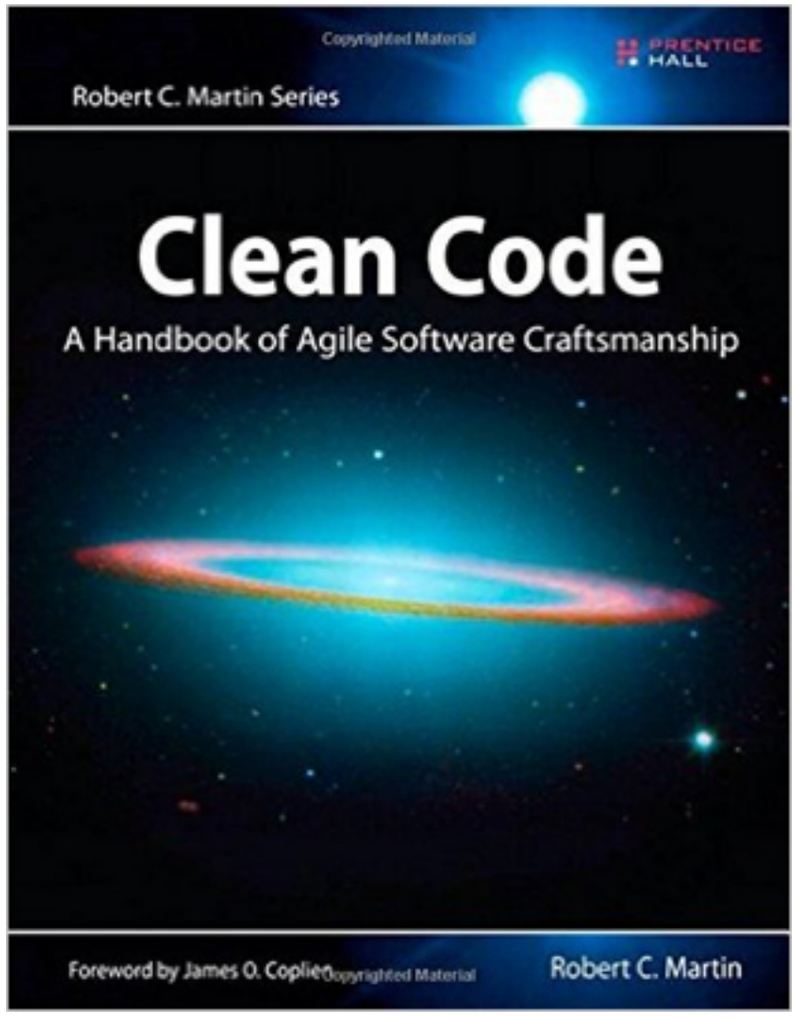
Quality management and design

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

http://clean-code-developer.de/


Copyrighted Material
PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Copyrighted Material
Robert C. Martin

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/

Qt Download Device Creation Application Development Services Developers Partners Blog

Wiki Documentation Forum Bug Reports Code Review

Qt Documentation

Google™ Custom Search

Qt 5.6 › Qt Widgets › C++ Classes › QMainWindow

QMainWindow Class

The QMainWindow class provides a main application window. More...

Header:	#include <QMainWindow>
qmake:	QT += widgets
Inherits:	QWidget

› List of all members, including inherited members

Public Types

enum	DockOption { AnimatedDocks, AllowNestedDocks, AllowTabbedDocks, ForceTabbedDocks, VerticalTabs, GroupedDragging }
flags	DockOptions

Properties

- animated : bool
- dockNestingEnabled : bool
- dockOptions : DockOptions
- documentMode : bool
- iconSize : QSize
- tabShape : QTabWidget::TabShape
- toolButtonStyle : Qt::ToolButtonStyle
- unifiedTitleAndToolBarOnMac : bool

› 58 properties inherited from QWidget

› 1 property inherited from QObject

Public Functions

	QMainWindow(QWidget *parent = Q_NULLPTR, Qt::WindowFlags flags = Qt::WindowFlags())
	~QMainWindow()

Contents

Reference

Getting Started

Static Code Analyzers

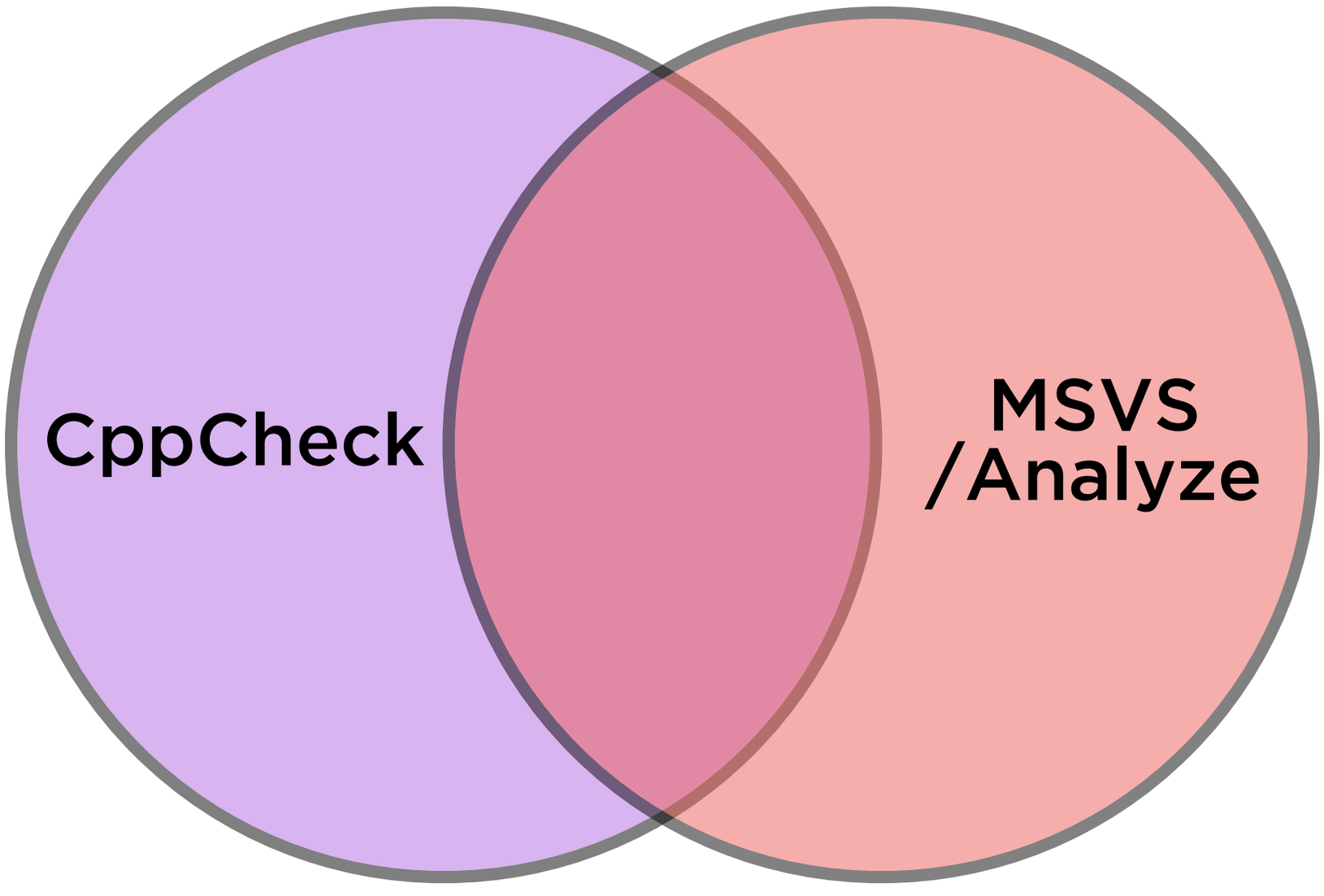
C++ Static Code Analyzers

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

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


STORCK®
EL CARAMELO RICO EN VITAMINAS PARA TODA LA FAMILIA • DAS MULTIVITAMIN-BONBON FÜR DIE GANZE FAMILIE
• THE MULTI-VITAMIN CANDY FOR THE WHOLE FAMILY •
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

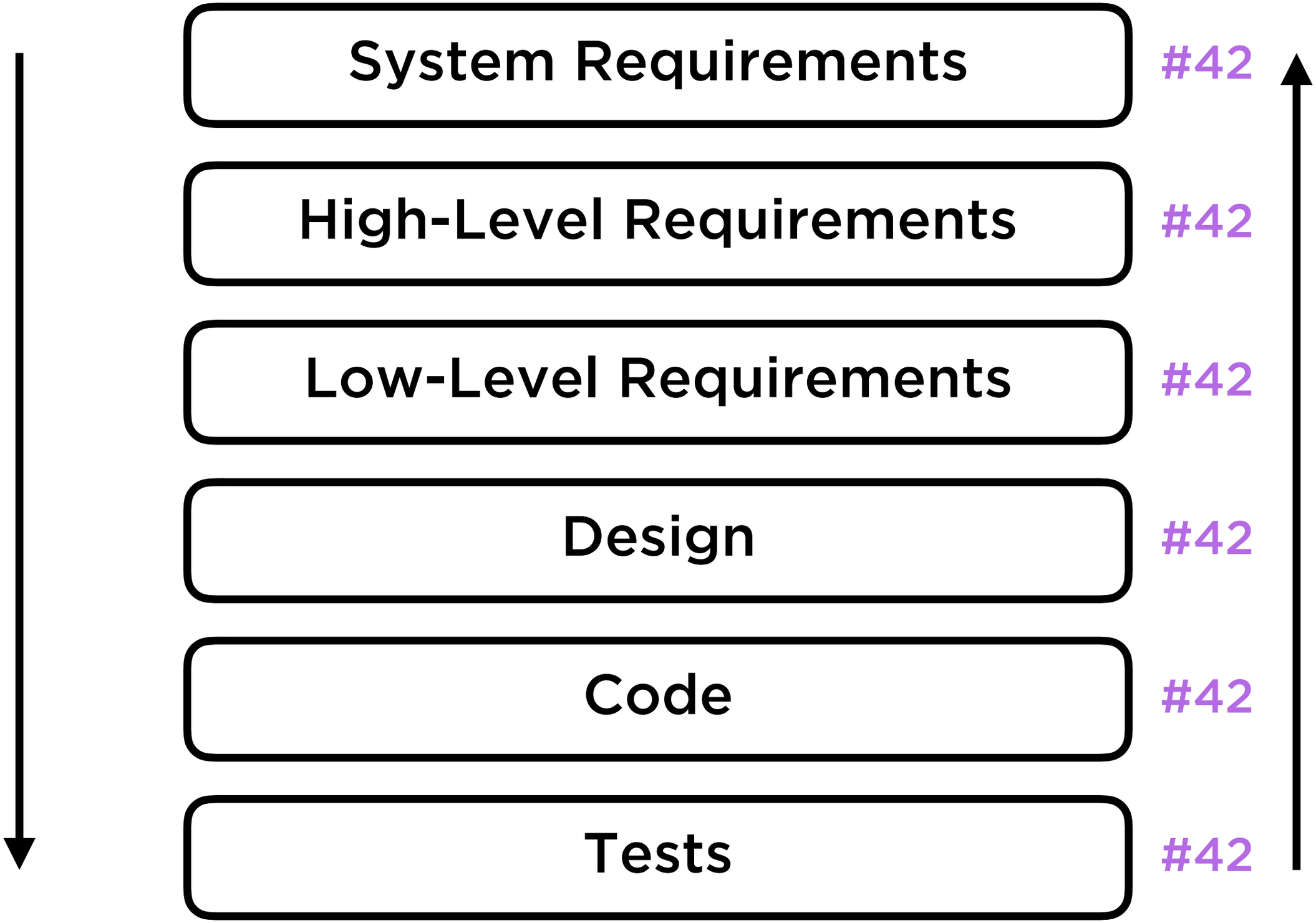
DO-178C

Software Considerations in Airborne Systems and Equipment Certification

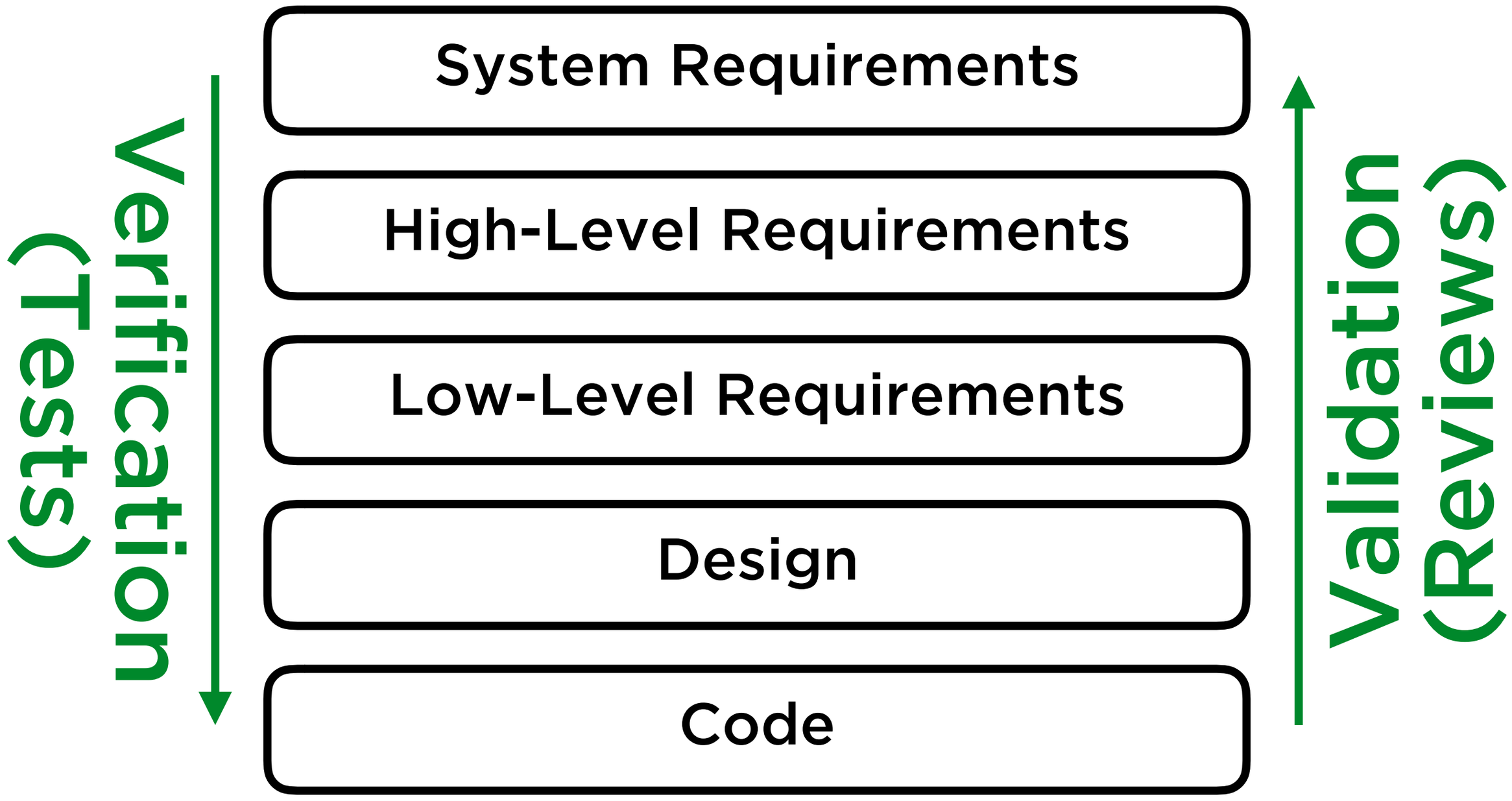
Design Assurance Level

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

Traceability


System Requirements
#42
High-Level Requirements
#42
Low-Level Requirements
#42
Design
#42
Code
#42
Tests
#42

Validation & Verification


System Requirements
High-Level Requirements
Low-Level Requirements
Design
Code
Verification
(Tests)
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

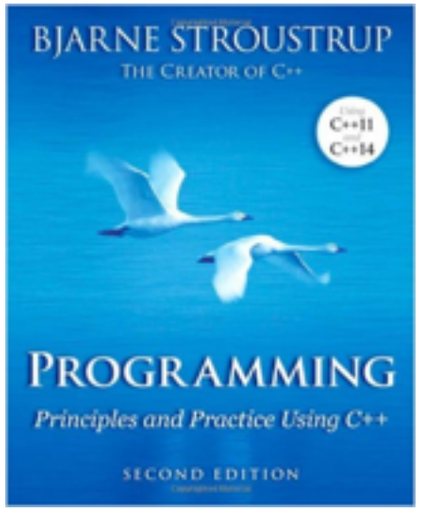
function macros

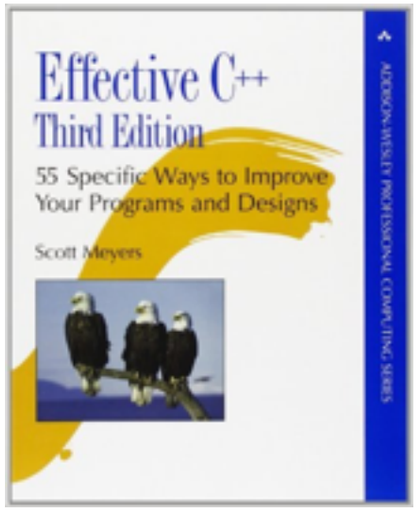
dead code

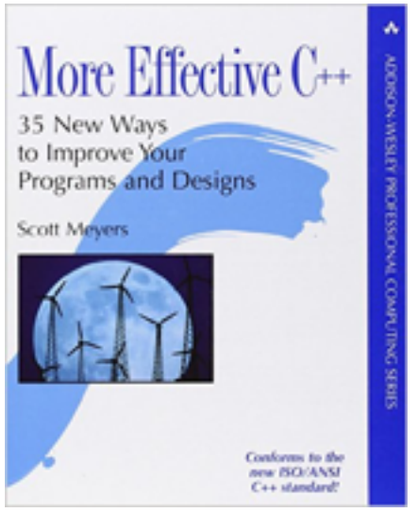
Complimentary Sources

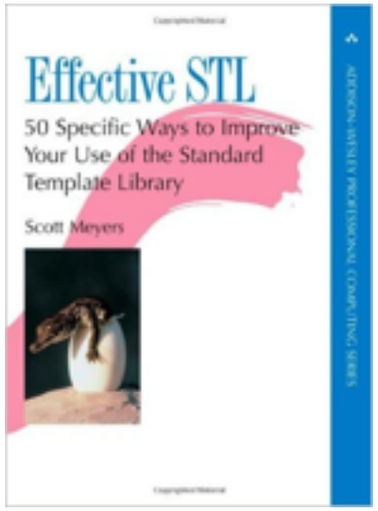
Literature

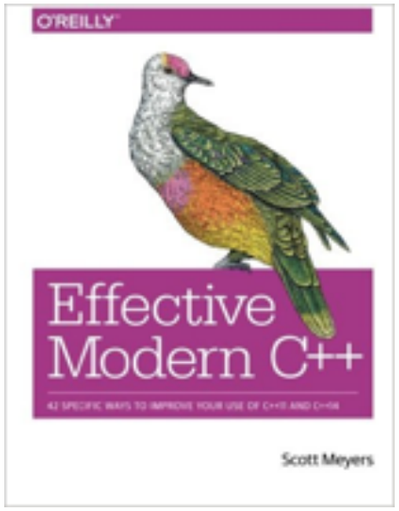
http://stackoverflow.com/questions/388242/the-definitive-c-book-guide-and-list

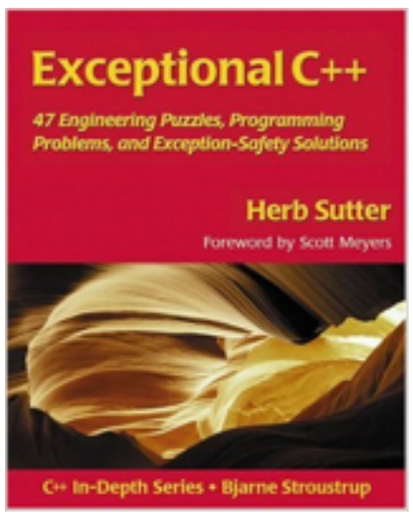


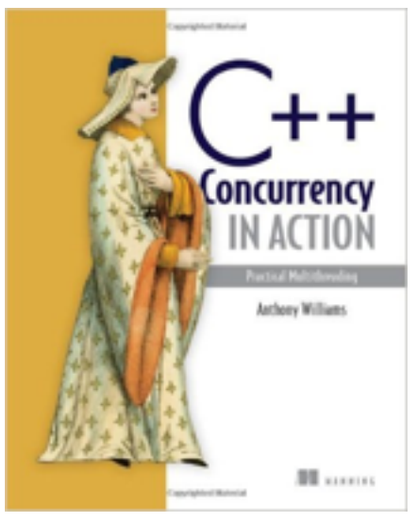


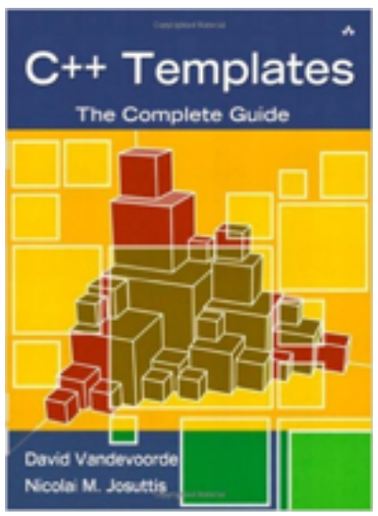


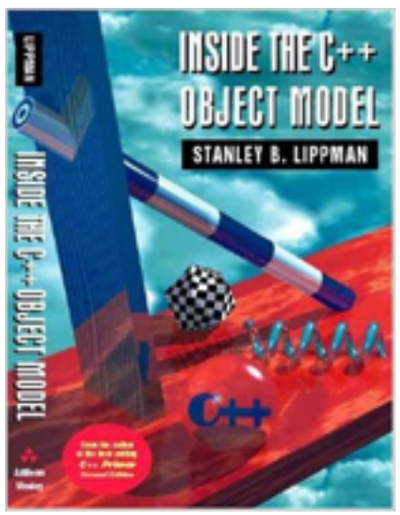












C++ Core Guidelines

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 pÅµÃoorly 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:

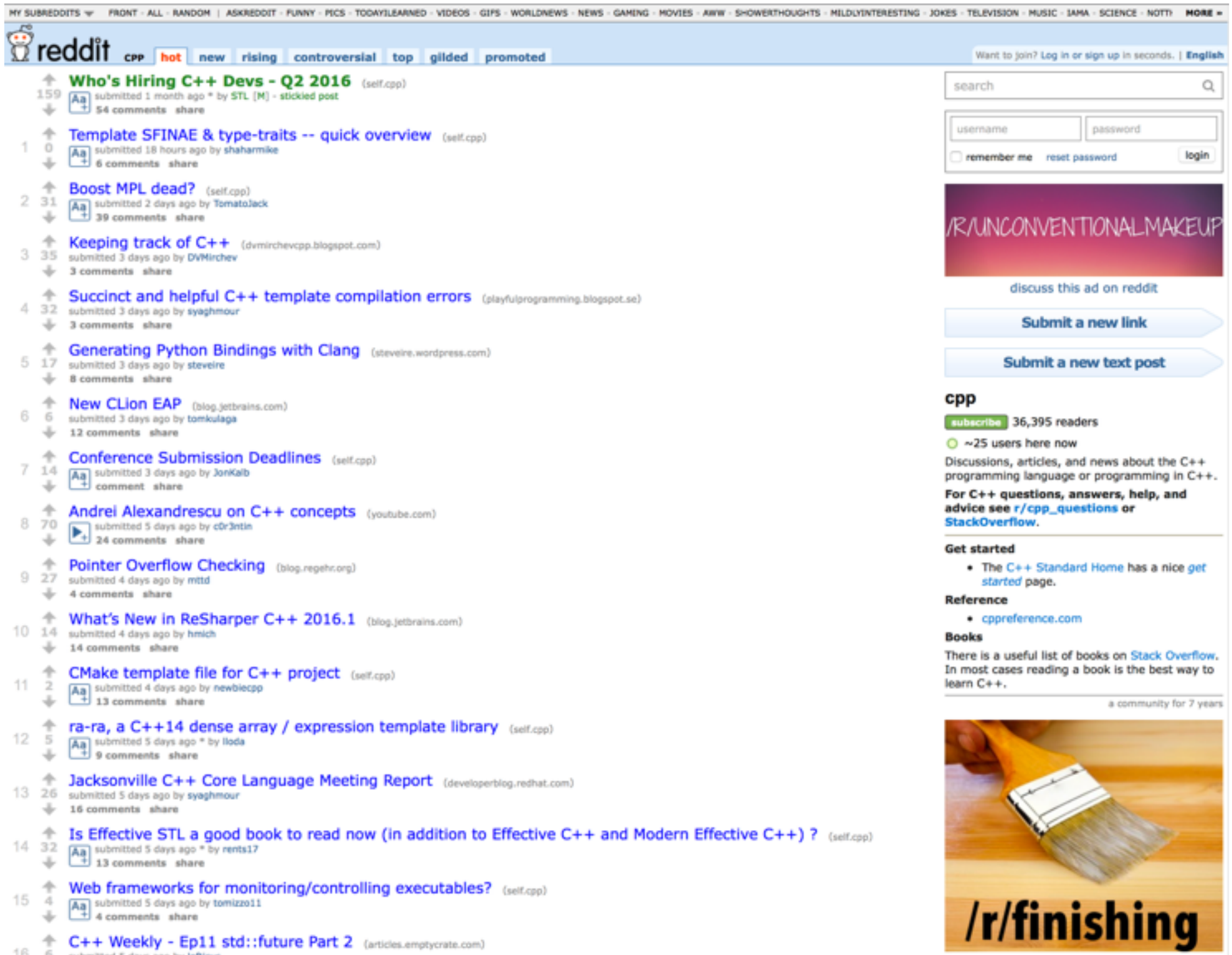
- The sets of rules have not been thoroughly checked for completeness, consistency, or enforceability.
- Triple question marks (???) mark known missing information
- Update reference sections; many pre-C++11 sources are too old.
- For a more-or-less up-to-date to-do list see: To-do: Unclassified proto-rules

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



C++ Talks

CppCon

https://www.youtube.com/user/CppCon

Meeting C++

https://www.youtube.com/user/MeetingCPP

C++Now

https://www.youtube.com/user/BoostCon

Questions?