




# Identifying Common Code Smells

in C++

 @arne\_mertz



Arne Mertz ( @arne\_mertz)  
Software Engineer, mostly embedded  
Learning C++ for almost two decades  
Trainer for C++ and maintainable code

# Code Smells - What's that?

A code smell is a surface indication that usually corresponds to a deeper problem in the system.

*Martin Fowler*

- Relatively easy to spot
- Not the actual problem
- Not *always* a problem
- Violation of principles
- Missing patterns, idioms, or abstractions
- Maintainability problem

<https://martinfowler.com/bliki/CodeSmell.html>

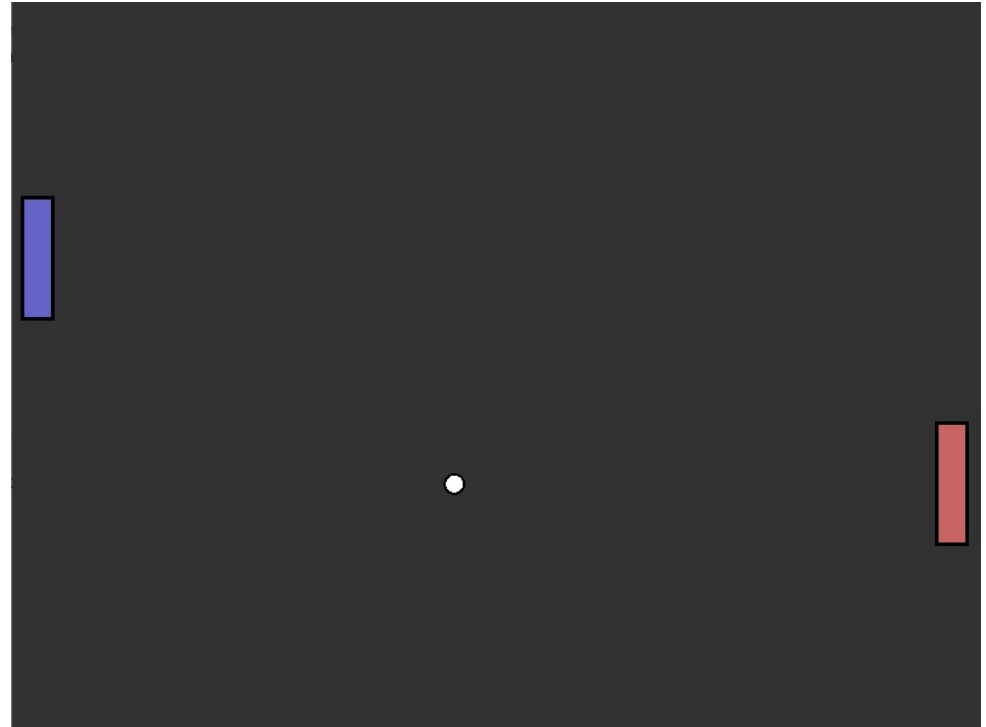
## About the code snippets

- All examples shown are open source
- This is not about picking on someone else, or their code
  - It's about showing that these smells can be found anywhere
- Most snippets are not production code but usage examples
  - But that does not mean that they should be less maintainable
  - On the contrary – they should be understandable for *everyone*

# Example code

## SFML, Tennis example

```
////////////////////////////////////  
/// Entry point of application  
///  
/// \return Application exit code  
///  
////////////////////////////////////  
int main()  
{  
    std::random_device rd;  
    std::mt19937 rng(rd());  
  
    // Define some constants  
    const float gameWidth = 800;  
    const float gameHeight = 600;  
    sf::Vector2f paddleSize(25, 100);  
    float ballRadius = 10.f;  
  
    // Create the window of the application  
    sf::RenderWindow window(sf::VideoMode({static_cast<unsigned int>(gameWidth), static_cast<unsigned int>(gameHeight)}), 32, "SFML Tennis",  
                             sf::Style::Titlebar | sf::Style::Close);  
    window.setVerticalSyncEnabled(true);  
  
    // Load the sounds used in the game  
    sf::SoundBuffer ballSoundBuffer;  
    if (!ballSoundBuffer.loadFromFile(resourcesDir() / "ball.wav"))  
        return EXIT_FAILURE;  
    sf::Sound ballSound(ballSoundBuffer);  
  
    // Create the SFML logo texture:  
    sf::Texture sfmlLogoTexture;
```



<https://github.com/SFML/SFML/blob/757cb3/examples/tennis/Tennis.cpp>

# Long function

A common code smell

- **Deeper problem:** violating Single Responsibility and Single Level of Abstraction Principles
- **Surface indication:** a function that is *too* long
  - Secondary indicator: blocks with single line „what“ comments

```
// Create the right paddle
```

```
sf::RectangleShape rightPaddle;  
rightPaddle.setSize(paddleSize - sf::Vector2f(3, 3));  
rightPaddle.setOutlineThickness(3);  
rightPaddle.setOutlineColor(sf::Color::Black);  
rightPaddle.setFillColor(sf::Color(200, 100, 100));  
rightPaddle.setOrigin(paddleSize / 2.f);
```

```
// Create the ball
```

```
sf::CircleShape ball;  
ball.setRadius(ballRadius - 3);  
ball.setOutlineThickness(2);  
ball.setOutlineColor(sf::Color::Black);  
ball.setFillColor(sf::Color::White);  
ball.setOrigin({ballRadius / 2.f, ballRadius / 2.f});
```

```
// Load the text font
```

```
sf::Font font;  
if (!font.loadFromFile(resourcesDir() / "tuffy.ttf"))  
    return EXIT_FAILURE;
```

```
// Initialize the pause message
```

```
sf::Text pauseMessage;  
pauseMessage.setFont(font);  
pauseMessage.setCharacterSize(40);  
pauseMessage.setPosition({170.f, 200.f});  
pauseMessage.setFillColor(sf::Color::White);
```

# Long function

How long is too long?

- Depends on the content
- Not quantifiable
  - 10 lines can be too long
  - 20 lines can be just long enough
  - 100 lines is definitely too long (maybe?)

<https://doc.qt.io/qt-6/qtwidgets-mainwindows-dockwidgets-example.html>

```
void MainWindow::newLetter()
{
    textEdit->clear();

    QTextCursor cursor(textEdit->textCursor());
    cursor.movePosition(QTextCursor::Start);
    QTextFrame *topFrame = cursor.currentFrame();
    QTextFrameFormat topFrameFormat
        = topFrame->frameFormat();
    topFrameFormat.setPadding(16);
    topFrame->setFrameFormat(topFrameFormat);

    QTextCharFormat textFormat;
    QTextCharFormat boldFormat;
    boldFormat.setFontWeight(QFont::Bold);
    QTextCharFormat italicFormat;
    italicFormat.setFontItalic(true);

    QTextTableFormat tableFormat;
    tableFormat.setBorder(1);
    tableFormat.setCellPadding(16);
    tableFormat.setAlignment(Qt::AlignRight);
    cursor.insertTable(1, 1, tableFormat);
    cursor.insertText("The Firm", boldFormat);
    cursor.insertBlock();
    cursor.insertText("321 City Street", textFormat);
    cursor.insertBlock();
    cursor.insertText("Industry Park").
```

# Long function

How do we fix it?

- Factor out functions
  - → reuse is not the only reason for functions!
- Block comments often are hints for good function names

```
void Shape::update( )
{
    updateVertices( );
    updateFillColors( );
    updateTextureCoordinates( );
    updateOutline( );
}
```

```
void Shape::update( )
{
    // Get the total number of points of the shape
    std::size_t count = getPointCount( );
    if (count < 3)
    {
        m_vertices.resize(0);
        m_outlineVertices.resize(0);
        return;
    }

    m_vertices.resize(count + 2); // + 2 for center and repeated first point

    // Position
    for (std::size_t i = 0; i < count; ++i)
        m_vertices[i + 1].position = getPoint(i);
    m_vertices[count + 1].position = m_vertices[1].position;

    // Update the bounding rectangle
    m_vertices[0] = m_vertices[1]; // so that the result of getBounds() is correct
    m_insideBounds = m_vertices.getBounds( );

    // Compute the center and make it the first vertex
    m_vertices[0].position.x = m_insideBounds.left + m_insideBounds.width / 2;
    m_vertices[0].position.y = m_insideBounds.top + m_insideBounds.height / 2;

    // Color
    updateFillColors( );

    // Texture coordinates
    updateTexCoords( );

    // Outline
    updateOutline( );
}
```



# Long function

How do we fix it?

- Factor out functions
  - → reuse is not the only reason for functions!
- Block comments often are hints for good function names
- Consider classes for data with complex functionality

```
// Create the left paddle
sf::RectangleShape leftPaddle;
leftPaddle.setSize(paddleSize - sf::Vector2f(3, 3));
leftPaddle.setOutlineThickness(3);
leftPaddle.setOutlineColor(sf::Color::Black);
leftPaddle.setFillColor(sf::Color(100, 100, 200));
leftPaddle.setOrigin(paddleSize / 2.f);

constexpr sf::Color PADDLE_BLUE{100, 100, 200};
constexpr sf::Color PADDLE_RED{200, 100, 100};

// ...

Paddle leftPaddle{PADDLE_BLUE};
Paddle rightPaddle{PADDLE_RED};
Ball ball{sf::Color::White};

// Create the ball
sf::CircleShape ball;
ball.setRadius(ballRadius - 3);
ball.setOutlineThickness(2);
ball.setOutlineColor(sf::Color::Black);
ball.setFillColor(sf::Color::White);
ball.setOrigin({ballRadius / 2.f, ballRadius / 2.f});
```

# Premature generalization

„What if...“

- **Surface indication:**

- Needless or unused parameters, callbacks, etc
- Templates that get instantiated with only one type
- Base classes with only one derived class (except for dependency inversion)

- **Underlying problem:**

- Violation of KISS and YAGNI
- Overly complex design, harder to maintain
- Explosion of test cases or missing tests

- **Fix:** keep it as simple as possible (but not simpler!)

```
Paddle leftPaddle{PADDLE_BLUE};  
Paddle rightPaddle{PADDLE_RED};  
Ball ball;
```

# Deeply nested control flow

```
while (window.isOpen())
{
    // Handle events
    for (sf::Event event; window.pollEvent(event);)
    {
        // ...

        // Space key pressed: play
        if (((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Space)) ||
            (event.type == sf::Event::TouchBegan))
        {
            if (!isPlaying)
            {
                // ... Set up for new game...

                // Reset the ball angle
                do
                {
                    // Make sure the ball initial angle is not too much vertical
                    ballAngle = sf::degrees(std::uniform_real_distribution<float>(0, 360)(rng));
                }
                while (std::abs(std::cos(ballAngle.asRadians())) < 0.7f);
            }
        }
    }
}
```

# Deeply nested control flow

- **Problems:**

- too much to keep in mind („how did we get here?“)
- SRP and SLoA violation

- Usually found together with long functions

- **Fix:**

- Factor out functions
- Invert conditions for early returns

```
while (window.isOpen())  
{  
    handleEvents(window);  
    if (isPlaying)  
    {  
        moveEntities();  
    }  
    redraw(window);  
}
```

# Complicated expression

```
if (ball.getPosition().x - ballRadius < leftPaddle.getPosition().x + paddleSize.x / 2 &&  
    ball.getPosition().x - ballRadius > leftPaddle.getPosition().x &&  
    ball.getPosition().y + ballRadius >= leftPaddle.getPosition().y - paddleSize.y / 2 &&  
    ball.getPosition().y - ballRadius <= leftPaddle.getPosition().y + paddleSize.y / 2)
```

# Complicated expression

- **Deeper problem:** violating Single Level of Abstraction
- **Fix:** factor out variables/functions

*// Check the collisions between the ball and the paddles*

*// Left Paddle*

```
if (ball.getPosition().x - ballRadius < leftPaddle.getPosition().x + paddleSize.x / 2 &&  
    ball.getPosition().x - ballRadius > leftPaddle.getPosition().x &&  
    ball.getPosition().y + ballRadius >= leftPaddle.getPosition().y - paddleSize.y / 2 &&  
    ball.getPosition().y - ballRadius <= leftPaddle.getPosition().y + paddleSize.y / 2)
```

# Complicated expression

```
const float ballUpperEdge = ball.getPosition().y + ballRadius;
const float ballLowerEdge = ball.getPosition().y - ballRadius;
const float ballLeftEdge = ball.getPosition().x - ballRadius;

const float paddleUpperEdge = leftPaddle.getPosition().y + paddleSize.y / 2;
const float paddleLowerEdge = leftPaddle.getPosition().y - paddleSize.y / 2;
const float paddleRightEdge = leftPaddle.getPosition().x + paddleSize.x / 2;
const float paddleMiddleX = leftPaddle.getPosition().x;

const bool ballIsAboveLowerEdge = ballUpperEdge >= paddleLowerEdge;
const bool ballIsBelowUpperEdge = ballLowerEdge <= paddleUpperEdge;
const bool ballTouchesPaddleOnLeft = (ballLeftEdge < paddleRightEdge)
                                     && (ballLeftEdge > paddleMiddleX);
const bool ballIsAtSameHeight = ballIsAboveLowerEdge && ballIsBelowUpperEdge;

const bool ballHitsLeftPaddle = ballTouchesPaddleOnLeft && ballIsAtSameHeight;

if (ballHitsLeftPaddle)
```

# Complicated expression

```
const bool ballIsAboveLowerEdge = ball.upperEdge() >= leftPaddle.lowerEdge();
const bool ballIsBelowUpperEdge = ball.lowerEdge() <= leftPaddle.upperEdge();
const bool ballTouchesOnLeft = (ball.leftEdge() < leftPaddle.rightEdge())
                                && (ball.leftEdge() > leftPaddle.middle().x);
const bool ballIsAtSameHeight = ballIsAboveLowerEdge && ballIsBelowUpperEdge;

const bool ballHitsLeftPaddle = ballTouchesOnLeft && ballIsAtSameHeight;

if (ballHitsLeftPaddle())
```



# „But...“

- „... that's a lot of code!“
  - It's a lot of detail that has been figured out

*I'm too lazy to type that much*

- „... ~~that can't be good for PERFORMANCE!!~~“
  - How do you know?
  - Does it matter?
  - Trust your optimizer
  - Measure, use a profiler!

# „Build Smell“: lack of tooling

Know and use your tooling, in the build pipeline and locally

- Compiler warnings (-Wall -Werror -pedantic)
- Optimizers and Profilers
- Static analysis (clang-tidy, cppcheck, ...)
- Sanitizers (run tests sanitized!)
- IDE tooling (e.g. refactoring tooling)

# C++ smell: Const(expr)-less

```
class SharedObj {
    sass::string getDbgFile() { return file; }
    size_t getDbgLine() { return line; }
};

class AST_Node {
    operator sass::string() {
        return to_string();
    }

    virtual sass::string to_string() const;
};

class Statement {
    virtual bool has_content();
};
```

```
// Define some constants
const float gameWidth = 800;
const float gameHeight = 600;
sf::Vector2f paddleSize(25, 100);
float ballRadius = 10.f;
```

- **Surface indication:** Functions and objects that could be marked constexpr or const aren't
- **Deeper problem:** Unclear semantics, accidental modifications

## `const`

- Any lack of `const` is a code smell
- `const` forces us into more organized code
- `const` prevents common errors
- `const` encourages more use of algorithms



Copyright Jason Turner

@lefticus

[emptycrate.com/idocpp](https://emptycrate.com/idocpp)


20.4



Jason Turner

## C++ Code Smells

Video Sponsorship Provided By:

 ansatz



```

try
{
    while( true )
    {
        // ... 500 LOC ...

        if( lSensor )
        {
            lSensor->Disconnect();
            delete lSensor;
            lSensor = nullptr;
        }
        if( lSensor2 )
        {
            lSensor2->Disconnect();
            delete lSensor2;
            lSensor2 = nullptr;
        }
        if( lPlayer != nullptr )
        {
            delete lPlayer;
            lPlayer = nullptr;
        }
    }
}

```

```

catch( std::exception &e )
{
    std::cout << "Exception: " << e.what()
               << std::endl;
}
if( lSensor )
{
    lSensor->Disconnect();
    delete lSensor;
}
if( lSensor2 )
{
    lSensor2->Disconnect();
    delete lSensor2;
}
if( lPlayer != nullptr )
{
    delete lPlayer;
}

```

<https://github.com/leddartech/LeddarSDK>

# C++ smell: missing RAII

*Responsibility Acquisition Is Initialization*

- **Underlying problem:** Resource leaks, other cleanup/reset bugs
- Use existing RAII classes from the standard library (e.g. smart pointers, locks, ...)
- Use destructors in your own classes to clean up
- Write RAII wrappers where you can't

```

class LdCanKomodo : public LdInterfaceCan
{
public:
    explicit LdCanKomodo( const LdConnectionInfoCan *aConnectionInfo );
    virtual ~LdCanKomodo();
private:
    int mHandle; // mHandle > 0 if it is valid
};

```

*copyable?!*

```

LeddarConnection::LdCanKomodo::~~LdCanKomodo()
{
    if( mMaster == nullptr && mHandle != 0 )
    {
        LdCanKomodo::Disconnect();
    }
}

```

```

void LeddarConnection::LdCanKomodo::Disconnect()
{
    if( mHandle != 0 )
    {
        km_disable( mHandle );
        km_close( mHandle );
        mHandle = 0;
    }
}

```

# C++ smell: Violating Rule of 5

- **Rule of 5:** If you have to define one of the „Big 5“, define the others as well.
  - Destructor
  - Copy Constructor and Assignment
  - Move Constructor and Assignment
- **Underlying problem:** Accidental bugs via compiler generated copies etc.
- Preferably `=default` or `=delete`



```
bool JoystickImpl::isConnectedDInput(unsigned int index)
{
    // Check if a joystick with the given index is in the connected list
    for (const JoystickRecord& record : joystickList)
    {
        if (record.index == index)
            return true;
    }
    return false;
}
```

```
// Search for a joystick with the given index in the connected list
for (const JoystickRecord& record : joystickList)
{
    if (record.index == index)
    {
        // Create device
        HRESULT result = directInput->CreateDevice(record.guid, &m_device, nullptr);

        // ... 280 LOC ...
        return true;
    }
}
return false;
```

# C++ Smell: missing algorithms

- Prefer range based for over „raw“ for loops
- Prefer <algorithm> over for loops

```
JoystickList::const_iterator findByIndex(JoystickList const &joystickList, unsigned index)
{
    return std::find_if(std::begin(joystickList),
                       std::end(joystickList),
                       [index](JoystickRecord const &jr) { return jr.index == index; });
}
```

```
bool JoystickImpl::isConnectedDInput(unsigned int index)
{
    return findByIndex(joystickList, index) != std::end(joystickList);
}
```

```
// Search for a joystick with the given index in the connected list
```

```
auto const found = findByIndex(joystickList, index);
```

```
if (found == std::end(joystickList)) {
```

```
    return false;
```

```
}
```

```
auto const& record = *found;
```

```
// Create device
```

```
HRESULT result = directInput->CreateDevice(record.guid, &m_device, nullptr);
```

```
// ...
```

```
return true;
```

# More loops

```
OtherContainer<Employee> source;  
// ...
```

```
std::vector<Employee> employees;  
// reserve...  
for (auto const& employee : source) {  
    employees.push_back(employee);  
}
```

```
std::vector<Employee> employees(  
    std::begin(source),  
    std::end(source)  
);
```

```
std::copy(std::begin(source),  
          std::end(source),  
          std::back_inserter(employees)  
);
```

# More loops

```
std::map<std::string, unsigned>
    salariesByName;

for (auto const& employee : employees) {
    salariesByName[employee.uniqueName()]
        = employee.salary();
}

for (auto const& employee : employees) {
    salariesByName.emplace(
        employee.uniqueName(),
        employee.salary()
    );
}
```

```
std::transform(
    std::begin(employees),
    std::end(employees),
    std::inserter(salariesByName,
        std::end(salariesByName)),
    [](auto const& employee) {
        return std::pair(
            employee.uniqueName(),
            employee.salary()
        );
    }
);
```

# Still more loops

```
for (auto const& employee : employees) {  
    if (!employee.isManager()) {  
        salariesByName.emplace(employee.uniqueName(), employee.salary());  
    }  
}
```

# transform\_if

```
template <typename InIter, typename OutIter,
          typename UnaryOp, typename Pred>
OutIter transform_if(InIter first, InIter last,
                    OutIter result, UnaryOp unaryOp,
                    Pred pred) {
    for(; first != last; ++first) {
        if(pred(*first)) {
            *result = unaryOp(*first);
            ++result;
        }
    }
    return result;
}
```

```
transform_if(
    std::begin(employees),
    std::end(employees),
    std::inserter(salariesByName,
                  std::end(salariesByName)),
    [](auto const& employee) {
        return std::make_pair(
            employee.uniqueName(),
            employee.salary()
        );
    },
    [](auto const& employee) {
        return !employee.isManager();
    }
);
```

# And ranges?

```
auto salariesByName = employees

| std::views::filter([](auto const& employee) {
    return !employee.isManager();
})

| std::views::transform([](auto const& employee)
    return std::pair(
        employee.uniqueName(),
        employee.salary()
    );
})

| std::ranges::to<std::map>;
```



# Back to the loops?

- It's still a smell
- Not every smell needs fixing
  - At least not right now

A code smell is a surface indication that usually corresponds to a deeper problem in the system.

*Martin Fowler*

# Conclusion

- Long function
- Premature generalization
- Deeply nested control flow
- Complicated expression
- Const(expr)-less code
- Missing RAI
- Missing rule of 5
- Raw loops
- <https://sourcemaking.com/refactoring/smells>
- Code smells can be found in every code base
  - The examples shown here are not necessarily bad code!
- Not always an error
- Not having access to C++(11+3n) does not mean our code needs to be smelly

- Jason Turner – CppCon 2019: „C++ Code Smells“
- Kate Gregory – CppCon 2019: „Naming is Hard: Let's Do Better“
- Kate Gregory – ACCUConf 2022: „Abstraction Patterns: Making Code Reliably Better Without Deep Understanding“

Thank you  Let's talk!

 Simplify C++! – [www.arne-mertz.de](http://www.arne-mertz.de)

 @arne\_mertz

 [arne.mertz@zuehlke.com](mailto:arne.mertz@zuehlke.com)

 [#include<C++>](https://discord.com/channels/417747578276679680/417747578276679680) Discord ([includecpp.org](http://includecpp.org))