Phantastic Code Smells

and where to find them

@arne_mertz

Arne Mertz

Software Engineer, mostly embedded

Trainer for modern C++ and clean code

Code Smells

No so phantastic after all

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

- Realtively easy to spot
- Not the actual problem

- Not *always* a problem

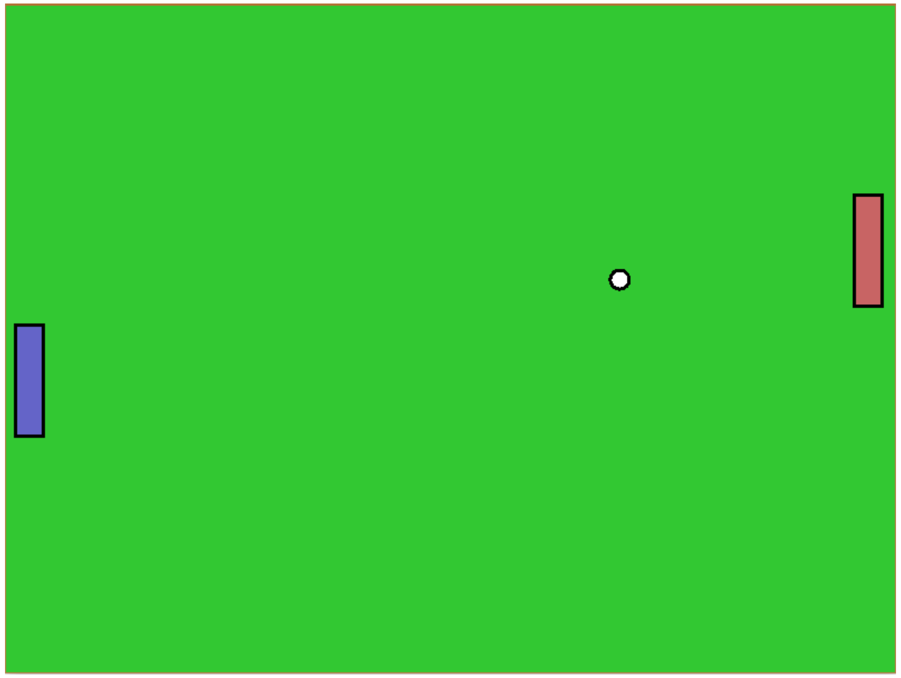
- Violation of principles
- Missing patterns, idioms, or abstractions
- Maintainability problem

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

Example code

SFML

```cpp
////////////////////////////////////////////////////////////
/// Entry point of application
///
/// \return Application exit code
///
////////////////////////////////////////////////////////////
int main()
{
    std::srand(static_cast<unsigned int>(std::time(NULL)));

    // Define some constants
    const float pi = 3.14159f;
    const int gameWidth = 800;
    const int gameHeight = 600;
    sf::Vector2f paddleSize(25, 100);
    float ballRadius = 10.f;

    // Create the window of the application
    sf::RenderWindow window(sf::VideoMode(gameWidth, gameHeight, 32), "SFML Pong",
                            sf::Style::Titlebar | sf::Style::Close);
```



https://github.com/SFML/SFML/blob/master/examples/pong/Pong.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

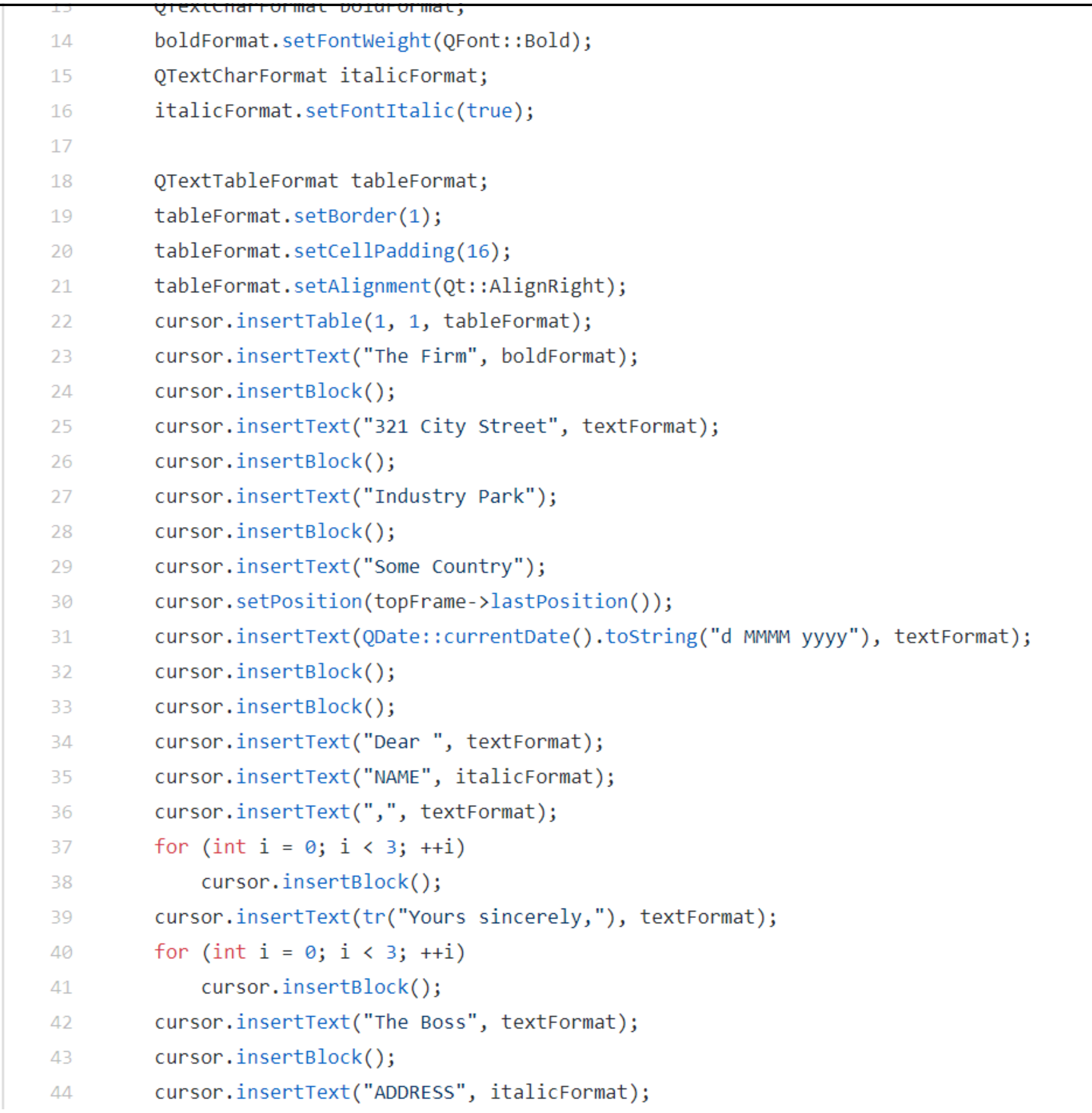
```
// Create the right paddle
sf::RectangleShape rightPaddle;
rightPaddle.setSize(paddleSize - sf::Vector2f(3, 3));
rightPaddle.setOutlineThickness(3);
rightPaddle.setOutlineColor(sf::Color::Black);
rightPaddle.setFillColor(sf::Color(200, 100, 100));
rightPaddle.setOrigin(paddleSize / 2.f);

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

// Load the text font
sf::Font font;
if (!font.loadFromFile(resourcesDir() + "sansation.ttf"))
    return EXIT_FAILURE;
```

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-5/qtwidgets-mainwindows-dockwidgets-example.html



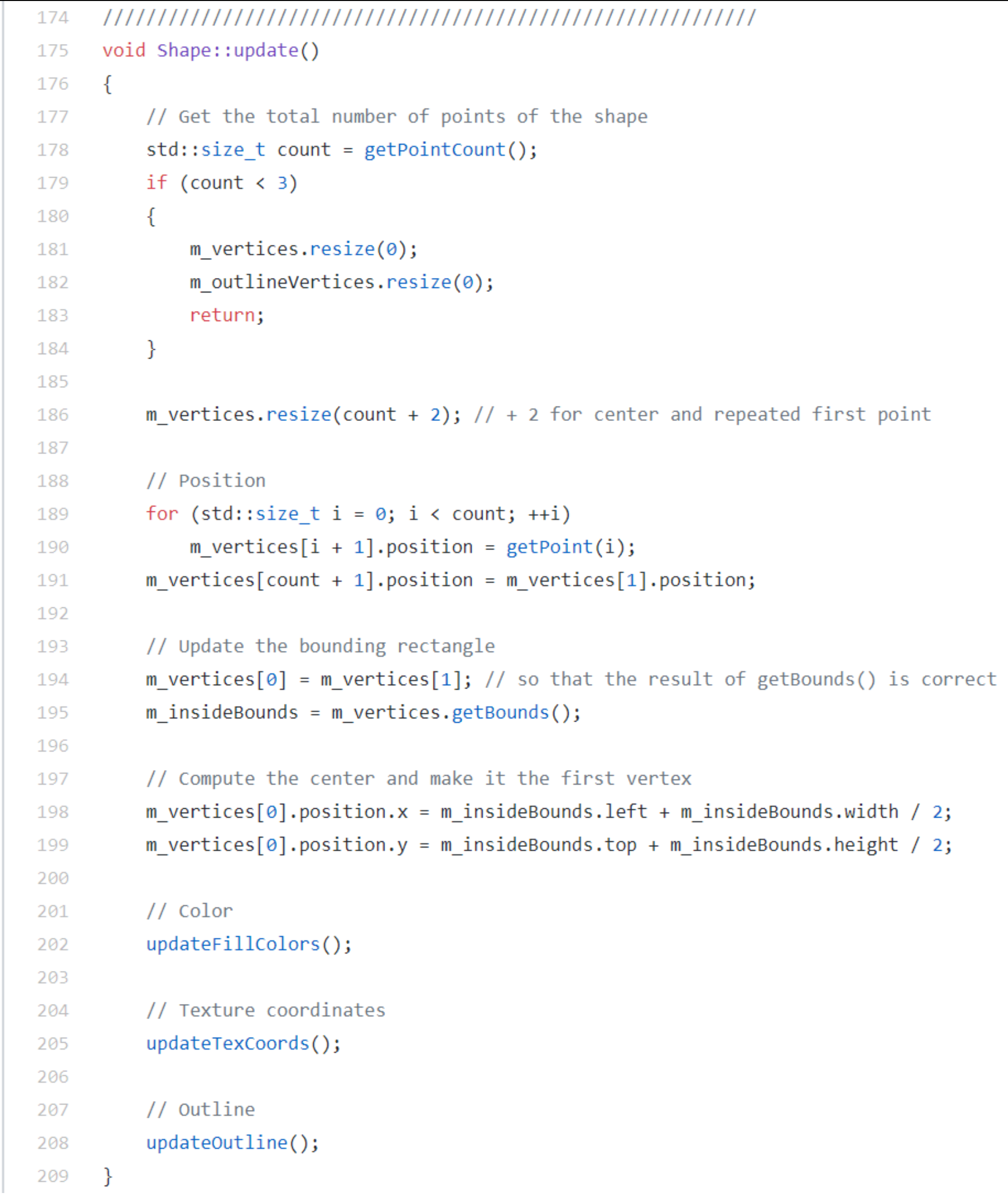
```
QTextCharFormat boldFormat;
boldFormat.setFontWeight(QFont::Bold);
QTextCharFormat italicFormat;
italicFormat.setFontItalic(true);

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");
cursor.insertBlock();
cursor.insertText("Some Country");
cursor.setPosition(topFrame->lastPosition());
cursor.insertText(QDate::currentDate().toString("d MMMM yyyy"), textFormat);
cursor.insertBlock();
cursor.insertBlock();
cursor.insertText("Dear ", textFormat);
cursor.insertText("NAME", italicFormat);
cursor.insertText(",", textFormat);
for (int i = 0; i < 3; ++i)
    cursor.insertBlock();
cursor.insertText(tr("Yours sincerely,"), textFormat);
for (int i = 0; i < 3; ++i)
    cursor.insertBlock();
cursor.insertText("The Boss", textFormat);
cursor.insertBlock();
cursor.insertText("ADDRESS", italicFormat);
```

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://github.com/SFML/SFML/blob/master/src/SFML/Graphics/Shape.cpp

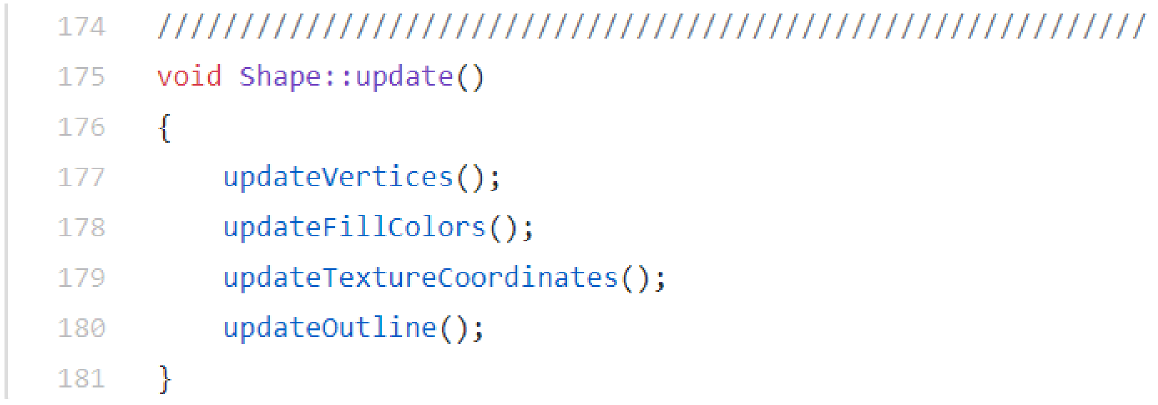


```cpp
////////////////////////////////////////////////////////////
void Shape::update()
{
    // Get the total number of points of the shape
    std::size_t count = getPointCount();
    if (count < 3)
    {
        m_vertices.resize(0);
        m_outlineVertices.resize(0);
        return;
    }

    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



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

```cpp
    // 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);

    // 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(3);
    ball.setOutlineColor(sf::Color::Black);
    ball.setFillColor(sf::Color::White);
    ball.setOrigin(ballRadius / 2, ballRadius / 2);
```

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

```cpp
const static sf::Color DARK_BLUE(100, 100, 200);
const static sf::Color DARK_RED(200, 100, 100);
```

```cpp
    sf::RectangleShape leftPaddle = createPaddle(DARK_BLUE);
    sf::RectangleShape rightPaddle = createPaddle(DARK_RED);
    sf::CircleShape ball = createBall();
```

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

```
Paddle leftPaddle(DARK_BLUE);
Paddle rightPaddle(DARK_RED);
Ball ball(sf::Color::White);
```

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

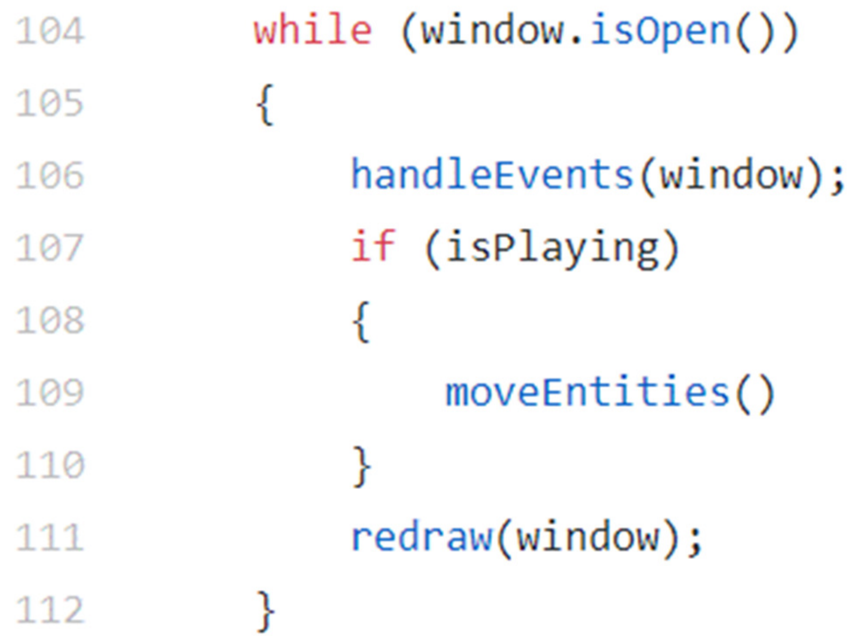
- Fix: keep it as simple as possible (but not simpler!)

```
Paddle leftPaddle(DARK_BLUE);
Paddle rightPaddle(DARK_RED);
Ball ball(sf::Color::White);
```

```cpp
    while (window.isOpen())
    {
        // Handle events
        sf::Event event;
        while (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)
                {
                    // Reset the ball angle
                    do
                    {
                        // Make sure the ball initial angle is not too much vertical
                        ballAngle = (std::rand() % 360) * 2 * pi / 360;
                    }
                    while (std::abs(std::cos(ballAngle)) < 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

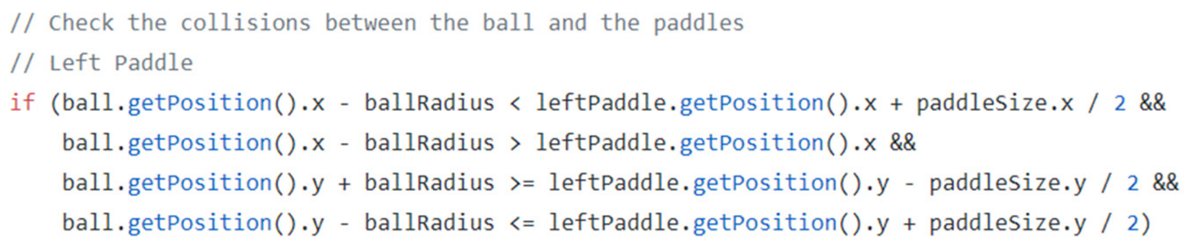


```
while (window.isOpen())
{
    handleEvents(window);
    if (isPlaying)
    {
        moveEntities()
    }
    redraw(window);
}
```

```
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 boolean 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 boolean expression

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

const float paddleLowerEdge = leftPaddle.getPosition().y - paddleSize.y / 2;
const float paddleUpperEdge = 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 ballTouchesOnLeft = ballLeftEdge < paddleRightEdge && ballLeftEdge > paddleMiddleX;
const bool ballIsSameHeight = ballIsAboveLowerEdge && ballIsBelowUpperEdge;
const bool ballHitsLeftPaddle = ballTouchesOnLeft && ballIsSameHeight;

if (ballHitsLeftPaddle)
```

Complicated boolean expression

```
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

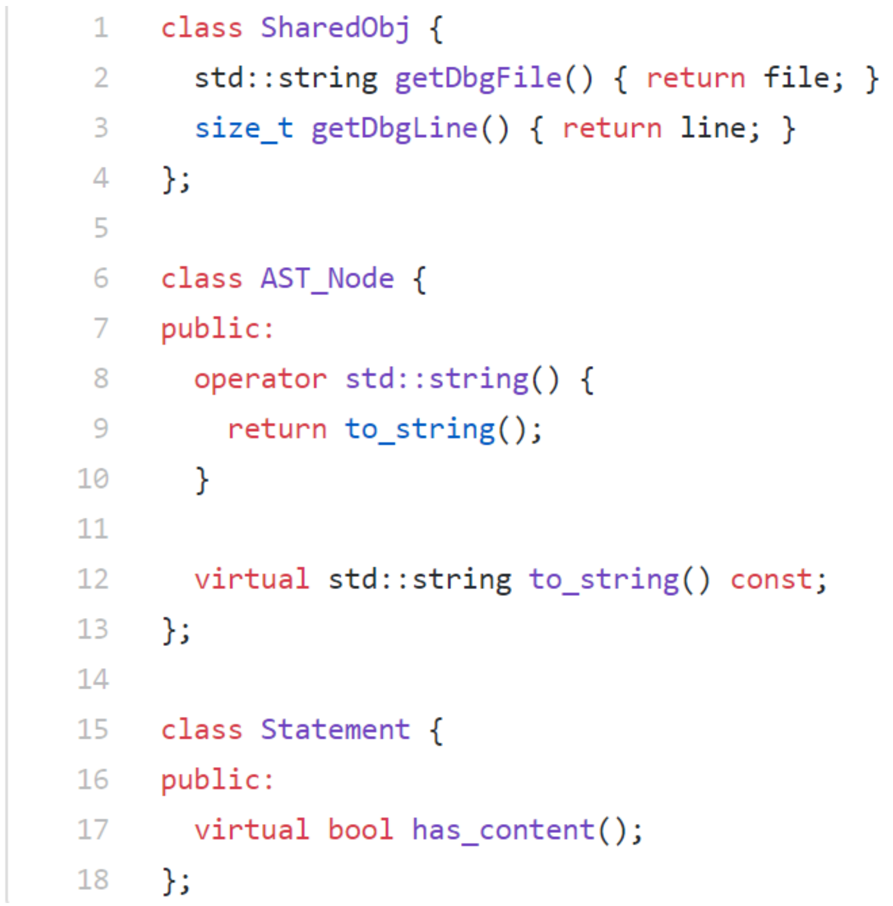
- „… ~~that can‘t be good for PERFORMACE!!~~“
  - How do you know?
  - Does it matter?
  - Trust your optimizer
  - Measure, use a profiler!

„Build Smell“: lack of tooling

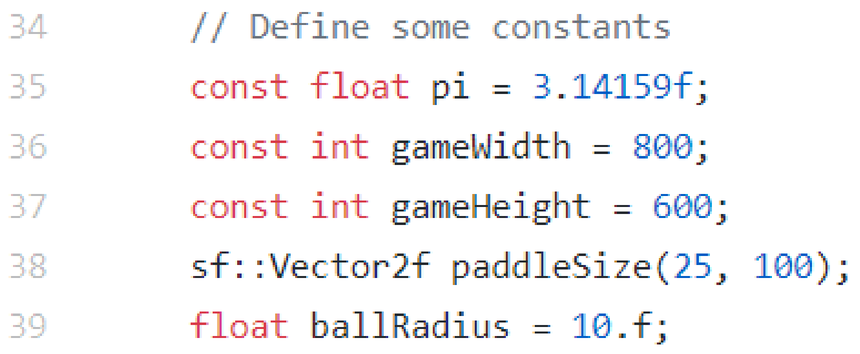
Know and use your tooling, in the build pipeline and locally

- 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



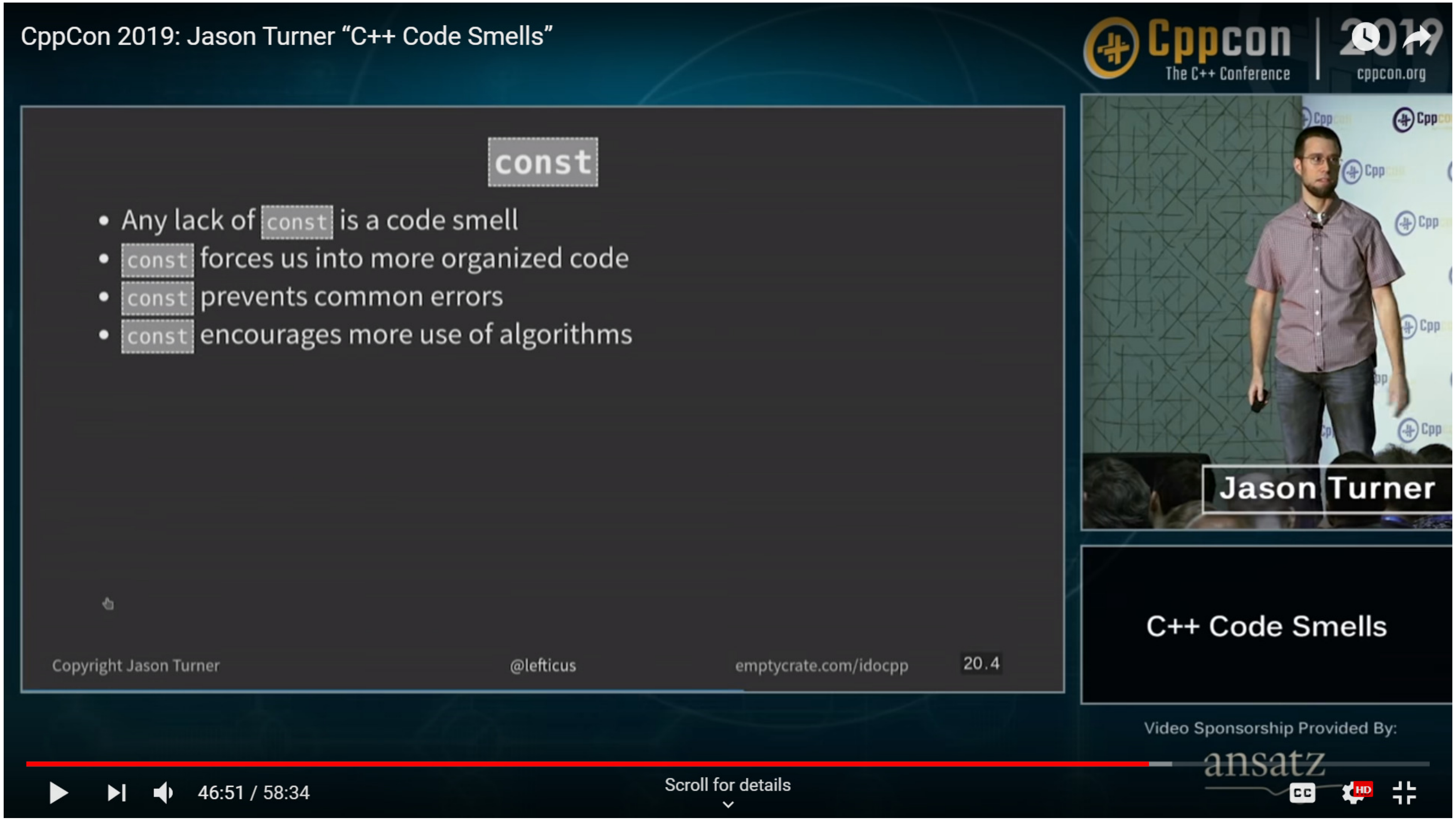
```cpp
class SharedObj {
  std::string getDbgFile() { return file; }
  size_t getDbgLine() { return line; }
};

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

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

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



```cpp
    // Define some constants
    const float pi = 3.14159f;
    const int gameWidth = 800;
    const int 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

https://github.com/sass/libsass



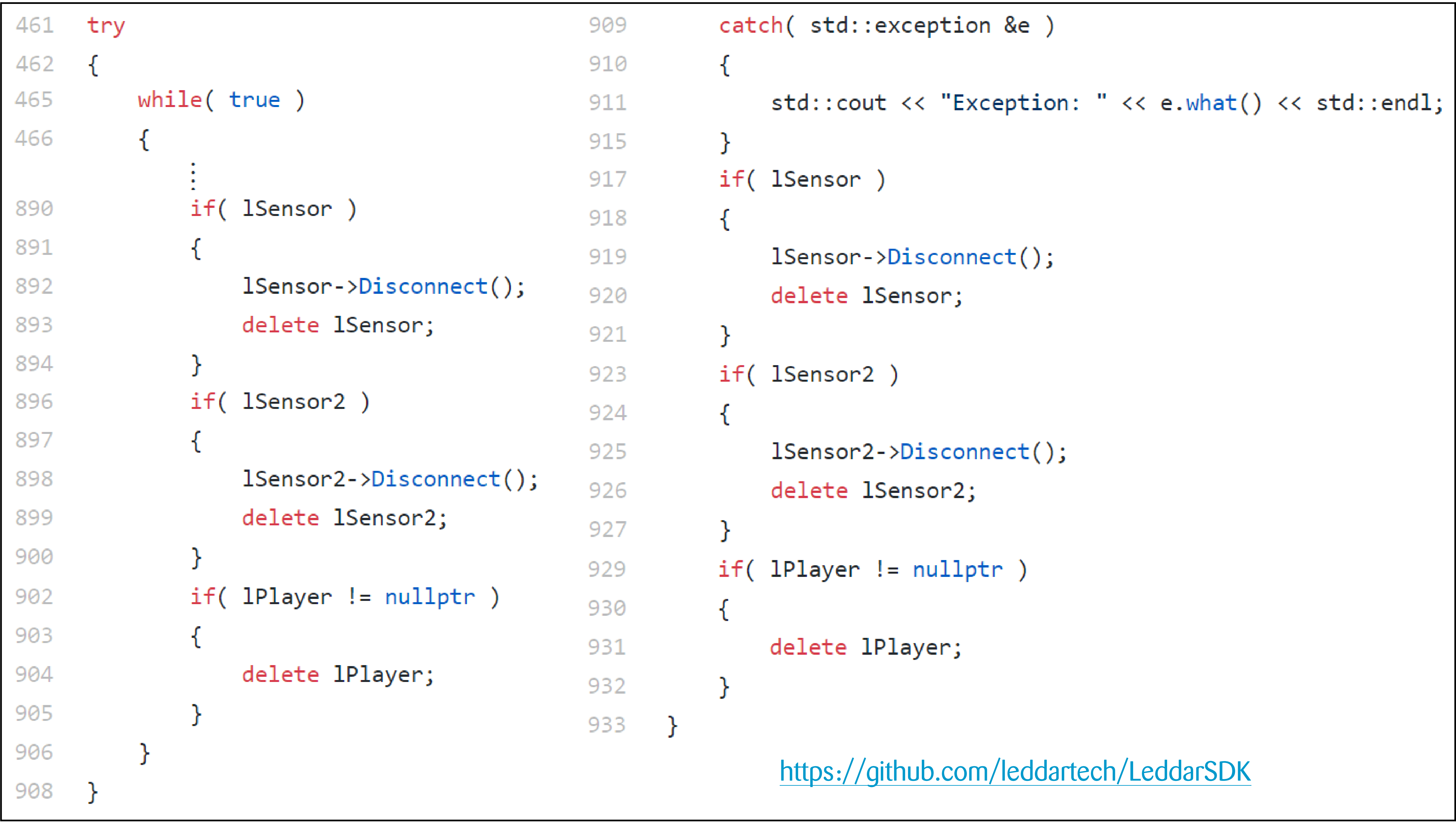
CppCon 2019: Jason Turner "C++ Code Smells"

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 20.4

CppCon The C++ Conference 2019 cppcon.org
Jason Turner
C++ Code Smells
Video Sponsorship Provided By:
ansatz
46:51 / 58:34
Scroll for details



```
    try
    {
        while( true )
        {
                ⋮
            if( lSensor )
            {
                lSensor->Disconnect();
                delete lSensor;
            }
            if( lSensor2 )
            {
                lSensor2->Disconnect();
                delete lSensor2;
            }
            if( lPlayer != nullptr )
            {
                delete lPlayer;
            }
        }
    }
    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 Accuisition 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

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

copyable?!

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

```cpp
void LeddarConnection::LdCanKomodo::Disconnect()
{
    km_disable( mHandle );
    km_close( mHandle );
    mHandle = 0;
}
```

C++ smell: Violating Rule of 3/5

- **Rule of 3/5**: If you have to define one of the Big 3/5, define the others as well.
  - Destructor
  - Copy Constructor and Assignment
  - Move Constructor and Assignment (since C++11)
- **Underlying problem**: Accidental bugs via compiler generated copies etc.
- Preferably `=default` or `=delete`

- Exception to the rule: defaulted virtual destructor in base classes

```cpp
bool JoystickImpl::isConnectedDInput(unsigned int index)
{
    // Check if a joystick with the given index is in the connected list
    for (std::vector<JoystickRecord>::iterator i = joystickList.begin();
         i != joystickList.end(); ++i)
    {
        if (i->index == index)
            return true;
    }
    return false;
}
```

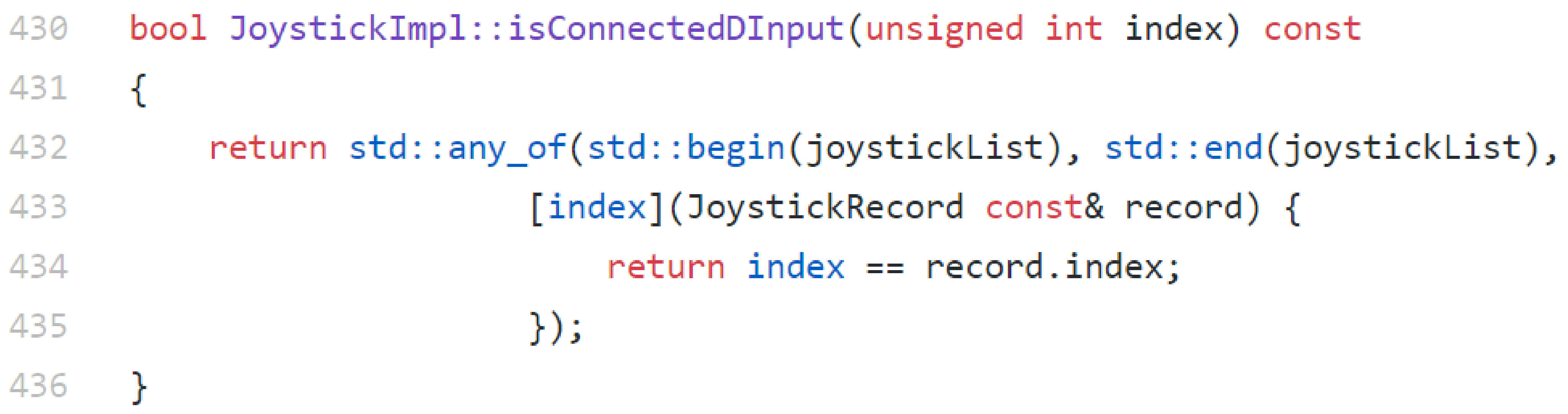
```cpp
    // Search for a joystick with the given index in the connected list
    for (std::vector<JoystickRecord>::iterator i = joystickList.begin();
         i != joystickList.end(); ++i)
    {
        if (i->index == index)
        {
            // Create device
            HRESULT result = directInput->CreateDevice(i->guid, &m_device, NULL);
            ⋮
        }
    }
    return false;
```

```cpp
struct SameIndex {
    unsigned int index;
    explicit SameIndex(unsigned int i) : index(i) {}
    bool operator()(JoystickRecord const& record) const {
        return record.index == index;
    }
};
```

```cpp
    // Search for a joystick with the given index in the connected list
    std::vector<JoystickRecord>::const_iterator found
        = std::find_if(joystickList.begin(), joystickList.end(), SameIndex(index));
    if (found == joystickList.end()) {
        return false;
    }

    // Create device
    HRESULT result = directInput->CreateDevice(found->guid, &m_device, NULL);
```

C++ Smell: raw loops

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



```
bool JoystickImpl::isConnectedDInput(unsigned int index) const
{
    return std::any_of(std::begin(joystickList), std::end(joystickList),
                        [index](JoystickRecord const& record) {
                            return index == record.index;
                        });
}
```

More loops

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

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

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

```cpp
std::vector employees(
  source.begin(),
  source.end()
);
```

More loops

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

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

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

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

Still more loops

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

transform_if

```cpp
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;
}
```

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

And ranges?

```cpp
auto salariesByName = employees

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

    | std::view::transform([](auto const& employee) {
        return std::make_pair(
          employee.uniqueName(),
          employee.salary()
        );
      })

    | 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 boolean expression
- Const(expr)-less code
- Missing RAII
- Missing rule of 3/5
- 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 C++(11+3n) does not mean our code needs to be smelly

- Jason Turner – CppCon 2019: „C++ Code Smells“
- Kate Grgory – CppCon 2019: „Naming is Hard: Let‘s Do Better“

zühlke
empowering ideas

Thank you 👋 Let‘s talk!

🌍 Simplify C++! – www.arne-mertz.de

@arne_mertz

arne.mertz@zuehlke.com

💬 #include<C++> Discord (includecpp.org)

Emojis by emojione.com