



27 July 2017

# Presenting effectively in meetings

Dirk Haun, ACCU 2017



Figure 1: 80%

## Meetup | Video

- ▶ Templates
- ▶ `enable_if`
- ▶ Concepts

## Post

- ▶ Valgrind integration
- ▶ Remote development support
- ▶ Better C++ language support
- ▶ Multiple toolchains
- ▶ Custom pretty printers for debugger
- ▶ GDB 8, LLDB 5
- ▶ Gutter icons for unit tests
- ▶ Improvements and fixes for Google Test and Catch

[YouTube](#) | [Post](#)

- ▶ Depends on:
  - ▶ C++17
  - ▶ concepts (C++20)
  - ▶ constexpr, if constexpr
  - ▶ compile-time meta-programming (P0589, P0633)
  - ▶ reflection (P0194, P0385, P0578, P0590, P0598)
- ▶ Start watching at 18m

# Metaclasses (cont.)

## Reflection

```
1 $T, $expr
```

## Compile-time programming

```
1 constexpr {  
2     for (auto m: $T.variables())  
3         if (m.name() == "xyzzzy")  
4             -> { int plugh; }  
5 }
```

# The Detection Idiom - a simpler way to SFINAE - Marshall Clow [ACCU 2017]

YouTube




void\_t and nonesuch

```
template <class...>
    using void_t = void;

struct nonesuch {
    nonesuch() = delete;
    ~nonesuch() = delete;
    nonesuch(nonesuch const&) = delete;
    void operator=(nonesuch const&) = delete;
};
```



# The Detection Idiom – a simpler way to SFINAE – Marshall Clow [ACCU 2017]



The exposition-only class DETECTOR

```
template <class Default, class AlwaysVoid,
         template<class...> class Op, class... Args>
struct DETECTOR
{
    using value_t = false_type;
    using type = Default;
};

template <class Default, template<class...>
         class Op, class... Args>
struct DETECTOR<Default, void_t<Op<Args...>>,
               Op, Args...>
{
    using value_t = true_type;
    using type = Op<Args...>;
};
```

Marshall Clow (Qualcomm Technology, Inc.) The Detection Idiom - a better way to SFINAE April 26, 2017 10 / 20

Figure 3: Inline 70%

# The Detection Idiom - a simpler way to SFINAE - Marshall Clow [ACCU 2017]





The bits that you do use

```
template <template<class...> class Op,  
         class... Args>  
using is_detected =  
    typename DETECTOR<nonesuch, void,  
                      Op, Args...>::value_t;  
  
template <template<class...> class Op,  
         class... Args>  
using detected_t =  
    typename DETECTOR<nonesuch, void,  
                      Op, Args...>::type;
```

Marshall Clow (Qualcomm Technology, Inc.) The Detection Idiom - a better way to SFINAE April 26, 2017 11 / 20

Figure 4: Inline 70%

# The Detection Idiom - a simpler way to SFINAE - Marshall Clow [ACCU 2017]



## How does it work?



It is surprisingly simple. You provide a 'type function'; a template that takes a type (or series of types) and returns a type.

Then you use `is_detected` to apply that type function to the types that you're interested in, and it returns to you: (a) whether or not this worked, and (b) if it did, what the result (type) was.

Marshall Clow (Qualcomm Technology, Inc.) The Detection Idiom - a better way to SFINAE April 26, 2017 12 / 20

Figure 5: Inline 70%

# The Detection Idiom - a simpler way to SFINAE - Marshall Clow [ACCU 2017]



## A slightly more practical example

```
struct Yes { typedef size_t size_type; };
struct No  { /* no 'size_type' */ };

template <typename T>
    using has_size = typename T::size_type;

template <typename T>
void doSomething( const T& t )
{
    detected_or_t<short, has_size, T> v = 123;
    // more with v
}
```

Marshall Clow (Qualcomm Technology, Inc.) The Detection Idiom - a better way to SFINAE April 26, 2017 7 / 20

Figure 6: Inline 70%

# The Detection Idiom - a simpler way to SFINAE - Marshall Clow [ACCU 2017]



## Variations on a theme

```
template <class Default, template<class...>
    class Op, class... Args>
    using detected_or =
        DETECTOR<Default, void, Op, Args...>;



template <class Expected, template<class...>
    class Op, class... Args>
    using is_detected_exact =
        is_same<Expected, detected_t<Op, Args...>>;

template <class To, template<class...>
    class Op, class... Args>
    using is_detected_convertible =
        is_convertible<detected_t<Op, Args...>, To>;
```

Marshall Clow (Qualcomm Technology, Inc.) The Detection Idiom - a better way to SFINAE April 26, 2017 14 / 20

Figure 7: Inline 70%

# The Detection Idiom - a simpler way to SFINAE - Marshall Clow [ACCU 2017]



## Copy-assignment example (1)

```
struct Yes { Yes &operator=(const Yes&)
              { return *this; } };
struct Huh { void operator=(const Huh&) {} };
struct No  { No  &operator=(const No&) = delete; };



template <class T>
    using copy_assign =
        decltype(std::declval<T&>() =
                  std::declval<T const &>());

static_assert( is_detected_v<copy_assign, Yes> );
static_assert( is_detected_v<copy_assign, Huh> );
static_assert(!is_detected_v<copy_assign, No> );
```

Marshall Clow (Qualcomm Technology, Inc.) The Detection Idiom - a better way to SFINAE April 26, 2017 16 / 20

Figure 8: Inline 70%

# The Detection Idiom - a simpler way to SFINAE - Marshall Clow [ACCU 2017]



## Copy-assignment example (2)

```
struct Yes { Yes &operator=(const Yes&)
            { return *this; } };
struct Huh { void operator=(const Huh&) {} };
struct No  { No  &operator=(const No&) = delete; };

template <class T>
using copy_assign =
    decltype(std::declval<T&>() =
              std::declval<T const &>());

static_assert(
    is_detected_exact_v<Yes&, copy_assign, Yes>);
static_assert(
    !is_detected_exact_v<Huh&, copy_assign, Huh>);
static_assert(
    !is_detected_exact_v<No&, copy_assign, No> );
```

Marshall Clow (Qualcomm Technology, Inc.) The Detection Idiom - a better way to SFINAE April 26, 2017 17 / 20

Figure 9: Inline 70%

## GitHub

Pystring is a collection of C++ functions which match the interface and behavior of python's string class methods using `std::string`. Implemented in C++, it does not require or make use of a python interpreter. It provides convenience and familiarity for common string operations not included in the standard C++ library. It's also useful in environments where both C++ and python are used.

Overlapping functionality (such as `index` and `slice/substr`) of `std::string` is included to match python interfaces.



## Video

- ▶ This is an “update” to the book “Modern C++ Design” by Andrei Alexandrescu
- ▶ Discusses updated template metaprogramming techniques:
  - ▶ policy-based class design;
  - ▶ replacing multiple templates with variadic templates.

### “Concept”-based polymorphism (1)

```
class drawable_concept{
public:
    drawable_concept() = default;
    virtual ~drawable_concept() = default;
    virtual void draw() = 0;
};

template<class T>
class drawable_model : public drawable_concept{
    T model_;
public:
    drawable_model(T const& model) : model_(model){}
    void draw(){
        model_.draw();
    }
    ~drawable_model() = default;
};
```

Figure 10: 150%

### “Concept”-based polymorphism (2)

```
class drawable{
    std::unique_ptr<drawable_concept> object_;
public:
    template <typename T>
    drawable(const T& x) : object_(
        new drawable_model<T>(x)) { }
    void draw(){
        object_->draw();
    };
};

struct my_widget{
    void draw(){};
};

int main(){
    drawable d{my_widget{}};
}
```

Transcript

## Post

- ▶ [Microsoft GSL](#)
- ▶ [GSL-Lite](#) - works with C++98 and C++03
- ▶ Quick overview of the classes available

# The Ultimate Question of Programming, Refactoring, and Everything

by Andrey Karpov, PVS Studio

## Post

- ▶ Tips and tricks on C++ programming
- ▶ Examples of bugs and bad practices
- ▶ Lots of code snippets
- ▶ Based on PVS-Studio diagnostics

&& == and, & == bitand



**Joe Groff**

@jckarter

Following



Of \*course\* you can use the C++ `and` and `bitand` keywords as reference qualifiers

```
struct RValue {  
    RValue(const RValue bitand);  
};  
  
RValue and rvalue(RValue and) and;
```

5:30 PM - 25 Jul 2017

19 Retweets 44 Likes

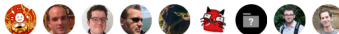


Figure 12: 70%

●●●●○ 02-UK 4G

17:33

📶 🔔 🔌 100% 🔋

[← Timeline](#)

Detail



**Stephan T. Lavavej** @StephanTLavavej

What's the most complicated keyword in C++?

Small brain: class

Normal brain: template

Big brain: double

Galactic brain: friend



 Rosyna Keller

106 Likes

26 Retweets