How I Code and Why

C++ Now! 2012 Talk by Jon Kalb

Catch Bugs at Compile time

- Much easier to find and fix
- Much less costly

Advice

- Put these (or other appropriate) tests right into your release code:

```
struct A
{
    std::string s_;
    std::vector<int> v_;
    A(const A&) = default;
};
```

```
// Howard says put these tests in!   Or else!!!
static_assert(std::is_nothrow_default_constructible<A>::value, "");
static_assert(std::is_copy_constructible<A>::value, "");
static_assert(std::is_copy_assignable<A>::value, "");
static_assert(std::is_nothrow_move_constructible<A>::value, "");
static_assert(std::is_nothrow_move_assignable<A>::value, "");
static_assert(std::is_nothrow_destructible<A>::value, "");
```

The Swapperator

```
#include "esc.hpp"
struct MyType…
{
    …
    void AnyMember() {esc::check_swap(this); …}
    …
}
```

The Swapperator

```
template <typename T> void check_swap(T* const t = 0)
{
    static_assert(noexcept(delete t), "msg...");
    static_assert(noexcept(T(std::move(*t))), "msg...");
    static_assert(noexcept(*t = std::move(*t)), "msg...");
    using std::swap;
    static_assert(noexcept(swap(*t, *t)), "msg...");
}
```

C++ 2011

The Swapperator

```
template <typename T> void check_swap(T* const t = 0)
{
    ...
    static_assert(
      std::is_nothrow_move_constructible<T>::value, "msg...");
    static_assert(
      std::is_nothrow_move_assignable<T>::value, "msg...");
   ...
}
```

boost::check_decl

- Just invented last night
- Both interface and implementation are "soft"
  - Please share your ideas and comments

C++ 2011

Using check_decl

```
struct empty_t
{
private: void attributes_() {check_decl(this);};
};
```

Using check_decl

```
struct movable_t
{
    movable_t(movable_t&&) = default;
    movable_t& operator=(movable_t&&) = default;
    movable_t(movable_t const&) = delete;
    movable_t& operator=(movable_t const&) = delete;
private:
    void attributes_()
{check_decl<attribute(movable | not_copyable)>(this);};
};
```

C++ 2011

check_decl

```
enum attribute
{
   minimal,
   copyable,
   movable = 2,
   swapable = 6,
   default_ctor = 8,
   all = 15,
   not_copyable = 16,
   not_movable = 32,
   singleton = 48
};
```

check_decl

```
template <attribute a = all, typename T>
void check_decl(T* const t = nullptr)
{
    static_assert(std::is_nothrow_destructible<T>::value, "");

    static_assert((not (copyable & a)) or
                  std::is_copy_constructible<T>::value, "");
    static_assert((not (copyable & a)) or
                  std::is_copy_assignable<T>::value, "");
```

C++ 2011

check_decl (cont.)

```
static_assert((not (movable & a)) or
        std::is_nothrow_move_constructible<T>::value, "");
static_assert((not (movable & a)) or
        std::is_nothrow_move_assignable<T>::value, "");
 static_assert((not (default_ctor & a)) or
        std::is_nothrow_default_constructible<T>::value, "");
static_assert((not (not_copyable & a)) or
     (not std::is_copy_constructible<T>::value), "");
```

check_decl (cont.)

```
    static_assert((not (not_copyable & a)) or
       (not std::is_copy_assignable<T>::value), "");
   static_assert((not (not_movable & a)) or
       (not std::is_move_constructible<T>::value), "");
   static_assert((not (not_movable & a)) or
       (not std::is_move_assignable<T>::value), "");
  static_assert((not (swapable & a)) or
         detail_::is_nothrow_swappable_<T>::value, "");
}
```

Using check_decl

```
check_decl<all, std::string>();
check_decl<all, std::vector<std::string>>();
```

Catch Bugs at Compile time

- Much easier to find and fix
- Much less costly
- C++11 is the language for this