Programming quick class

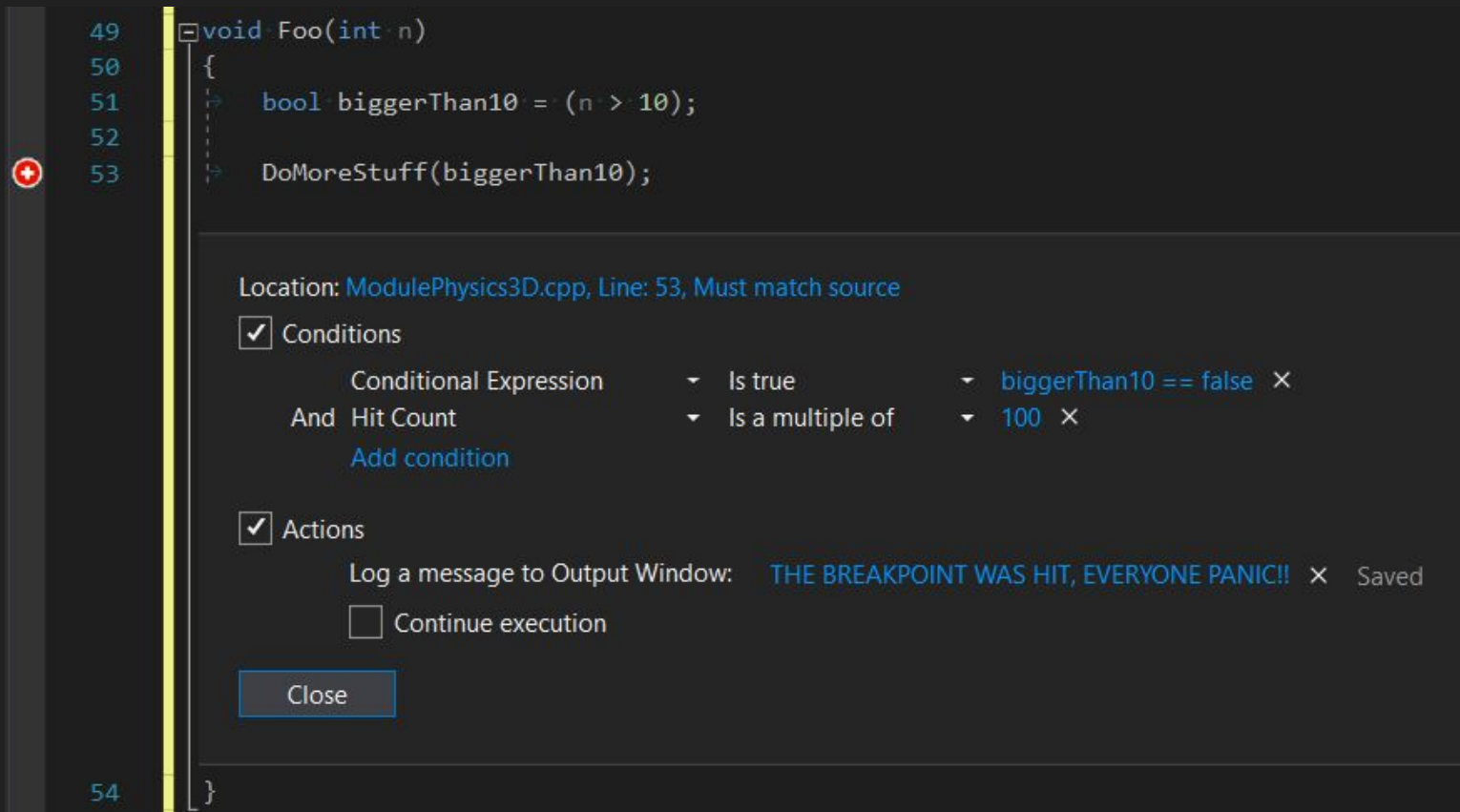Rapid-fire tips and reminders

I'd recommend taking notes

- We'll be covering a wide variety of topics, most unrelated to each other.
- All of them will be useful at some point or another.
- I'll go through quick, or we won't have time!

However, if you have any doubts or need some extra info, just ask.
We're here to improve!

Visual studio tips and tricks

Conditional breakpoints

Call stack is your best friend against crashes

Call Stack
Name
➡️ 3D Physics - class3 Handout.exe!ModulePhysics3D::PreUpdate(float dt) Line 84
3D Physics - class3 Handout.exe!Application::Update() Line 93
3D Physics - class3 Handout.exe!SDL_main(int argc, char * * argv) Line 55
3D Physics - class3 Handout.exe!main_utf8(int argc, char * * argv) Line 126
3D Physics - class3 Handout.exe!WinMain(HINSTANCE__ * hInst, HINSTANCE__ * hPrev, char * szCmdLine, int sw) Line 189
[External Code]
[Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll]

"Watch" variables

```
38    void Foo(int n, const vec3& vec)
39    {
40        const int var = n + (vec.x / vec.y) * vec.z;
41        const int array[] = { n, vec.x, vec.y, vec.z, var };
42        const int* pointer = array;
43
44        Func(var);
```

Watch 1

Name	Value	Type
n	10	int
vec	{x=1.00000000 y=-2.00000000 z=5.00000000 …}	const vec3 &
var	7	const int
array	0x00a0f4fc {10, 1, -2, 5, 7}	const int[5]
pointer	0x00a0f4fc {10}	const int *
pointer, [5]	0x00a0f4fc {10, 1, -2, 5, 7}	const int[5]
n > 5	true	bool
Func(n)	true	bool

Rename variable

Ctrl + R, Ctrl + R

Allows you to rename a variable and all the references to it at once.

Some good reads

Game programming patterns

Must read.

Light read, with knowledge about architecture and ways to organize your code.

It's free online. Go get it.

Effective C++, Third edition

More technical book.

Still, has some invaluable knowledge inside.

Specific tips and tricks to improve your programming skills.
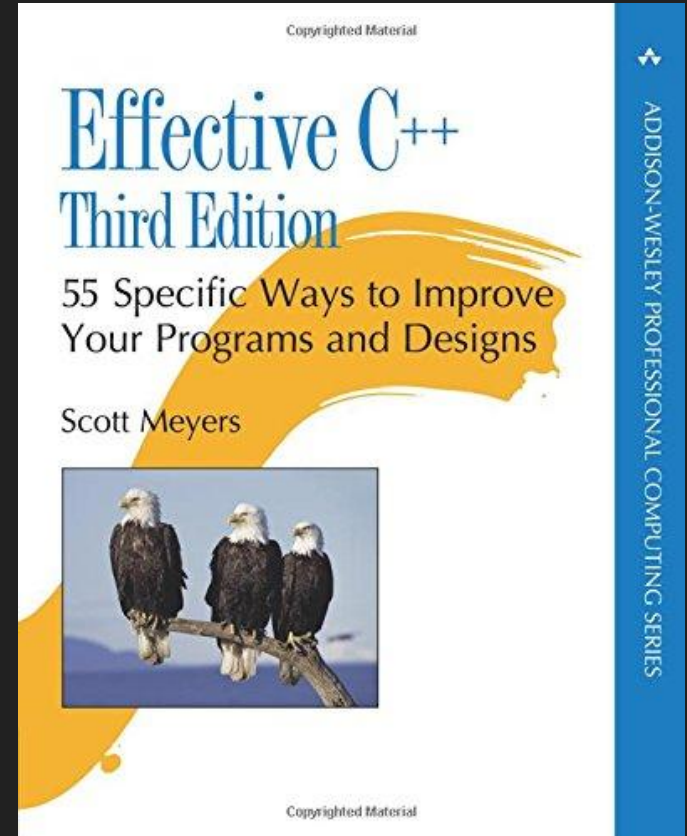
Game engine architecture

For those of you who love tech and low level coding.
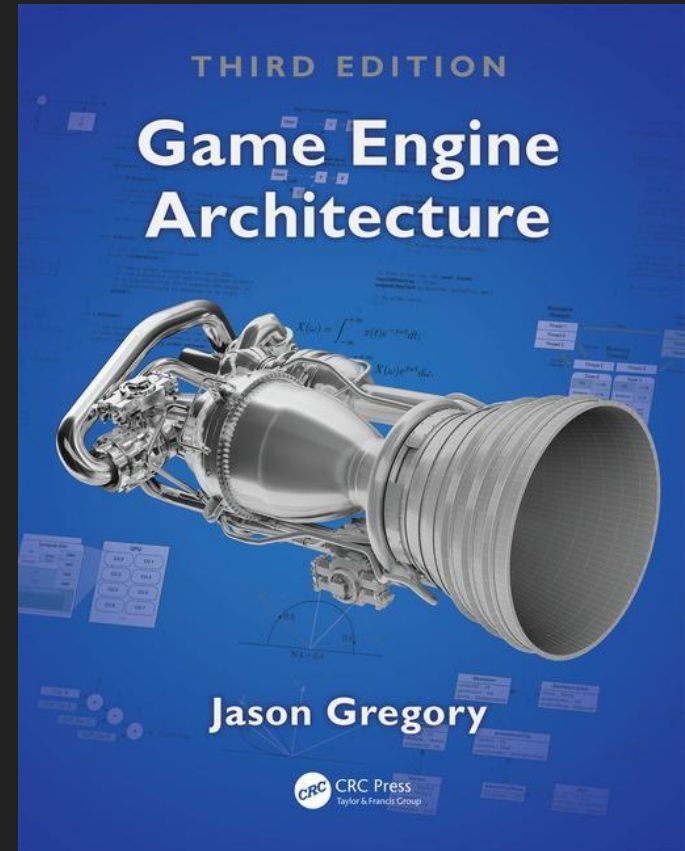
Code style

What does "good code" mean?

1st: Easy to use. Most of the time you'll be working with others. Don't make your code a pain to read and maintain.

2nd: Clear header file
    - A single class has a single purpose
    - A single function does a single job
    - No need to read the .cpp. Header should be enough

3rd: Avoid excessive commenting

4th: Avoid spaghetti code. A readable code is better than an optimized code.

Clear header

Here we have a super simple class.

Can you tell me what this class is?

What it does?

What every function is meant to do?

```cpp
class Shape
{
public:
    void Draw();
    STypes Type();

    void Pos(float a, float b, float c);
    void Pos(const vec3& p);
    void Rotation(float a, const vec3 &u);
    void Scale(float x, float y, float z);
    vec3 Pos();
    mat3x3 Rotation();
    vec3 Scale();
private:
    virtual bool PrivateDraw();

public:
    Color color;
    bool axis, wire;
private:
    STypes type;
    mat4x4 trans;
    PhysBody3D PhysicalBody;

public:
    Shape();
    Shape(const Shape&);
    virtual ~Shape();
};
```

```cpp
class Primitive
{
public:
    Primitive();
    Primitive(const Primitive&);
    virtual ~Primitive();

    void            Render() const;
    PrimitiveShape  GetShape() const;

    void    SetPos(float x, float y, float z);
    void    SetPos(const vec3& pos);
    vec3    GetPos() const;

    void    SetRotation(float angle, const vec3 &u);
    mat3x3  GetRotation() const;

    void    SetScale(float x, float y, float z);
    vec3    GetScale() const;

    Color   meshColor;
    bool    drawAxis;
    bool    drawWireframe;

protected:
    virtual void InnerRender() const;
private:
    PrimitiveShape type;
    mat4x4 transform;
    PhysBody3D body;
};
```

```cpp
class Shape
{
public:
    void Draw();
    STypes Type();

    void Pos(float a, float b, float c);
    void Pos(const vec3& p);
    void Rotation(float a, const vec3 &u);
    void Scale(float x, float y, float z);
    vec3 Pos();
    mat3x3 Rotation();
    vec3 Scale();
private:
    virtual bool PrivateDraw();

public:
    Color color;
    bool axis, wire;
private:
    STypes type;
    mat4x4 trans;
    PhysBody3D PhysicalBody;

public:
    Shape();
    Shape(const Shape&);
    virtual ~Shape();
};
```

The legend

```cpp
class Character
{
public:
    //Constructor, functions...
    void Heal(int amount);
    void Damage(int amount);
    bool IsInvulnerableState() const;
    //More stuff
};
```

Single responsibility

A single class should have a single responsibility.

Render renders. Scene has objects. Player manages inputs.

If you can't describe in a quick sentence what your class does, something might be wrong. Start separating responsibilities.

Same goes for functions.

Learn how to comment

If you've written good code, you won't need comments.

No one wants to read your failed novel attempt.

```cpp
class Primitive
{
public:
    Primitive();
    Primitive(const Primitive&);
    //Destructor for the primitive
    virtual ~Primitive();

    //Draw the object into the world using OpenGL code
    void        Render() const;
    //Get an enum describing the shape of the primitive
    PrimitiveShape GetShape() const;

    //Set position in world coordinates using three float point numbers
    void    SetPos(float x, float y, float z);
    //Set position in world coordinates using a vec3
    void    SetPos(const vec3& pos);
    //Get the position in world coordinates
    vec3    GetPos() const;
    //Add a rotation to the current rotation of the body. Angle defines the amount
    //of rotation and "u" defines the axis around which the rotation takes place
    void    SetRotation(float angle, const vec3 &u);
    //Get the rotation in world coordinates in form of a three by three matrix
    mat3x3  GetRotation() const;
    //Set a new scale for the object in world coordinates
    void    SetScale(float x, float y, float z);
    //Oh god please stop i can't take it anymore
    vec3    GetScale() const;
```

Learn how to comment

```cpp
class MyClass
{
public:
    void AStrangeFunction();
};
```

```cpp
//Here we can explain what the function does
//and even use multiple lines for it
void MyClass::AStrangeFunction()
{
    //Do some black magic in here no one needs to understand
}
```

```cpp
MyClass instance;
instance.
```

AStrangeFunction

public : void MyClass::AStrangeFunction()
Here we can explain what the function does
and even use multiple lines for it
File: Primitive.cpp

When you write over 1000 lines of code
and it works perfectly on the first run


```
993   print("Hello World")
994   print("Hello World")
995   print("Hello World")
996   print("Hello World")
997   print("Hello World")
998   print("Hello World")
999   print("Hello World")
1000  print("Hello World")
```

Encapsulate your code

Design your code like you design your game:
Remember **KISS**:

<div align="center">

Keep **I**t **S**tupidly **S**imple

</div>

If another team member uses your functions wrong… it's your fault, not theirs.

If you don't want them to touch something, make it private.
Trust is nice. But coding isn't about trust.

If something can go wrong, it will.

Encapsulate your code

What could go wrong in this class?

How can it be used incorrectly?

```cpp
class Primitive
{
public:
    Primitive();
    Primitive(const Primitive&);
    virtual ~Primitive();

    void        Render() const;
    PrimitiveShape GetShape() const;

    void    SetPos(float x, float y, float z);
    void    SetPos(const vec3& pos);
    vec3    GetPos() const;

    void    SetRotation(float angle, const vec3 &u);
    mat3x3  GetRotation() const;

    void    SetScale(float x, float y, float z);
    vec3    GetScale() const;

    Color   meshColor;
    bool    drawAxis;
    bool    drawWireframe;

private:
    virtual void InnerRender() const;

    PrimitiveShape type;
    mat4x4 transform;
    PhysBody3D body;
};
```

Enum class instead of enum

```cpp
enum PrimitiveShape
{
    Primitive_Cube,
    Primitive_Sphere,
    Primitive_Cylinder
};



void Function()
{
    PrimitiveShape shape = Primitive_Cube;
}
```

```cpp
enum class PrimitiveShape
{
    Cube,
    Sphere,
    Cylinder
};



void Function()
{
    PrimitiveShape shape = PrimitiveShape::Cube;
}
```

Avoid pointers as much as you can

They're powerful. They can be optimal. They allow us to do all sorts of cool tricks.

They're the best part of C++.

But they're also the worst.

They're fragile. Most crashes come from them. Easily corruptible. Can be **incredibly slow if used incorrectly**.

ejem As far as i've seen, most of you are doing just that *ejem*

Caché miss

Every pointer is a potential caché miss.

No fancy graphics, blackboard time.
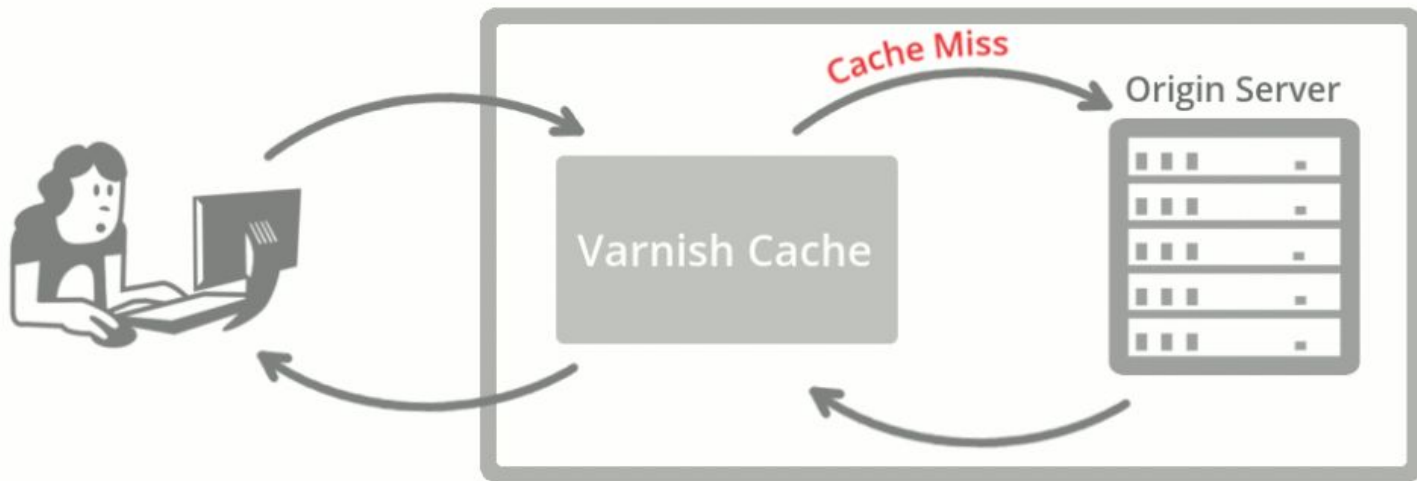
How to code a class

Start from the obvious:
- The header

Single responsibility.
As little public information as possible.
Remember KISS.

What happens behind the scenes is up to you.
What's public it's a problem for everyone.

C++ curiosities

The compiler is your friend

Which one is easier to understand? Which one is the quickest?

A.
```
Func(n + (vec.x / vec.y) * vec.z);
```

B.
```
const int var = n + (vec.x / vec.y) * vec.z;
Func(var);
```

C.
```
int var = n + (vec.x / vec.y) * vec.z;
Func(var);
```

Compiler optimizations

"var" won't occupy memory until "Func()" is reached

"anotherVar" won't exist once you compile

```cpp
void Foo(int n, const vec3& vec)
{
    int var = n + (vec.x / vec.y) * vec.z;
    int anotherVar = 10;
    Func(var);
}
```

Forward declarations vs Includes

Forward declaration	Include
A "promise" that it exists	Will give access to the full class
Usable by pointers and references	That's it
Doesn't need to include any files	They're slower
Reduces compilation time	Increased compiler times
Helps with encapsulation	Increased noise and less encapsulation
Preferred whenever possible	Required when accessing the class or creating it in memory

Constructor / Destructor order and syntax

Stop constructing your data in your header

Initialize EVERYTHING

Construction order:

1. Base class constructor

2. Variables in the order they're declared in the header

```cpp
Primitive::Primitive()
    : color(White)
    , drawAxis(false)
    , drawWired(false)
    , transform(IdentityMatrix)
    , body()
    , type(PrimitiveTypes::Point)
{}
```

```cpp
class Primitive
{
public:
    Primitive();

    //Functions and stuff

    Color color;
    bool drawAxis;
    bool drawWired;
private:
    mat4x4 transform;
    PhysBody3D body;
    PrimitiveTypes type;
};
```

Pointers and references

```cpp
int a = 0;
int b = 1;

int* pointer = &a;
pointer = &b;
*pointer = 4;

const int* constPointer = &a;
constPointer = &b;
*constPointer = 4; //Error! int is const

int* const pointerConst = &a;
pointerConst = &b; //Error! Pointer is const
*pointerConst = 4;

const int* const constPointerConst = &a;
constPointerConst = &b; //Error!
*constPointerConst = 4; //Both are const
```

Casts

C-style cast: (float)myInt;

New style casts:

Static cast: std::static_cast<float>(myInt);
Compile-time cast

Dynamic cast: std::dynamic_cast<float>(myInt);
Execution-time cast

Explicit cast, pointer cast...

Know your keywords!

Virtual   Overrideable functions

Override   Will throw an error if it isn't overriding a function

Final   A class that cannot be inherited from

Const   Constant function/class

Inline   Injects code instead of referring to the function in memory

Static   Unlinked from any instances

Extern   Extreme forward declaration

Virtual classes

Pure virtual functions:
virtual void Update() = 0;

Virtual functions are slower!

If a class has a single virtual function…

Virtual ~destructor() is required!

Passing arguments through const reference

Why?

Bit-wise operators

When memory is tight, or an array is unnecessary

&	|	~	^	>>	<<
AND	OR	NOT	XOR	Right Shift	Left Shift

Compressed "if" statements

Avoid them, they're hard to read…
But you need to know they exist

```
const int n = 8;
bool biggerThanTen = (n > 10 ? true : false);
std::string AsAString = (n > 10 ? "Bigger than 10" : "Smaller than 10");
```

Std data containers

std data containers

You'll end up using them. They're faster and better than anything we can do.

HOWEVER!

You need to know how they work inside.
That's why p2DynArray and p2List are useful.

std data containers

std::vector   The one you'll use 90% of the time. Unless you have a very good reason to use another one, stick with a vector. Contiguous in memory, caché friendly, incredibly fast and efficient.

std::list   Linked list. Useful if you need to move big data more than you need to iterate.

std::set   Vector that doesn't allow duplicates.

std::map Access and order elements using a key. Quick search, since it uses a binary tree search method. Keys cannot be repeated.

std::queue   A queue. First in, first out.

std::stackA stack. First in, last out.

Smart pointers

They're great. Just… don't use them. Yet.

```cpp
std::shared_ptr<Module> MyFirstSmartPointer = std::make_shared<ModulePhysics3D>();

std::shared_ptr<Module> AnotherSmartPointer = MyFirstSmartPointer;

std::weak_ptr<Module> AWeakPointer = MyFirstSmartPointer;
```

Function pointers

They're cool, but don't use them unless you really need to.

```cpp
void SimpleFunction() {};
bool ComplexFunction(int n) { return n > 15; }

void Foo(int n, const vec3& vec)
{
    void(*FuncPtr1)() = SimpleFunction;
    bool(*FuncPtr2)(int) = ComplexFunction;

    FuncPtr1();
    FuncPtr2(15);
}
```

Function pointers

However… member functions are an issue

```cpp
bool ModulePhysics3D::Init()
{
    update_status(*PointerToUpdate1)(float) = Update;
    update_status(*PointerToUpdate2)(float) = this->Update;

    std::function<update_status(float)> FancyPointer = Update;


    return true;
}
```

Function pointers

Are they intimidating yet?

```cpp
bool ModulePhysics3D::Init()
{
    std::function<update_status(float)> FancyPointer = [this](float dt) { return this->Update(dt); };

    return true;
}
```

Lambas

```cpp
class Character
{
    std::string name;
    int hp;
    //MoreStuff
};

void UpdateMyCoolGameUpdate()
{
    //Imagine we have a list of 100 characters
    std::vector<Character> ListOfCharacters;

    //We want to sort all the "characters" in a list,
    //ordering them by the remaining HP
    ListOfCharacters.sort();

}
```

Lambas

```cpp
class Character
{
public:
    std::string name;
    int hp;
    //MoreStuff
};

void UpdateMyCoolGameUpdate()
{
    //Imagine we have a list of 100 characters
    std::vector<Character> ListOfCharacters;

    std::sort(ListOfCharacters.begin(), ListOfCharacters.end(),
        [](const Character& a, const Character& b)
        { return a.hp > b.hp; }
    );

}
```

Lambas

Be sure you know what you're doing, or your program will slow down A LOT

```
[=]() { /*A simple function... that may kill your program*/ };
[&]() { /*A bit better, but still not great*/ };
```

The end…?

Wow, that was long and tough. I really don't think we'll get to see this even.
But if we do, most important things to remember…

- Clear code. Clearer headers.
- Make classes that cannot be used wrongly
- Reduce includes whenever possible
- Use the correct data container (Vector 90% of the time)

The simpler and smaller the code is, the better it is. Overcomplicated code is the death of any project.