Code Quality and Programming Standards

COMSM0086

Dr Simon Lock and Dr Sion Hannuna

Aim of This Lecture

To try to make you into a better programmer !
(not just a "coder")

Achieved by considering code quality at two levels:

1. Low-level material "quality" of your source code
2. Higher-level structural "quality" of the program

Code Quality

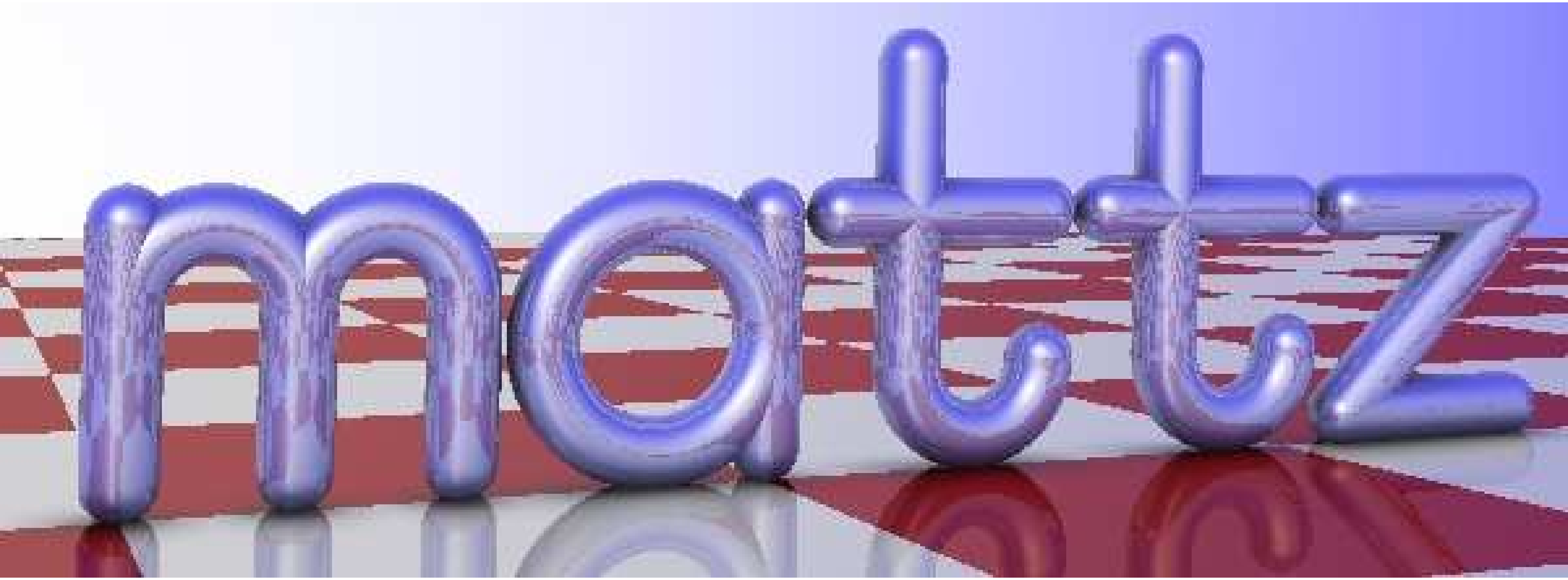
Good code is not just about correct operation
Code may run just fine, but still be badly written !

Key questions to ask yourself about your code are:

- How easy is your code for others to understand ?
- How easy is your code for others to change ?
- Does your code support long-term maintenance ?

As a "coder" you probably don't care about these
As a "programmer" you definitely should !

```
#include <stdio.h>
typedef double f;f H=.5,Y=.66,S=-1,I,y=-111;extern"C"{f cos(f),pow(f
,f),atan2(f,f);}struct v{f x,y,z;v(f a=0,f b=0,f c=0):x(a),y(b),z(c)
{}f operator%(v r){return x*r.x+y*r.y+z*r.z;}v operator+(v r){return
v(x+r.x,y+r.y,z+r.z);}v operator*(f s){return v(x*s,y*s,z*s);}}W(1,1
,1),P,C,M;f U(f a){return a<0?0:a>1?1:a;}v _(v t){return t*pow(t%t,-
H);}f Q(v c){M=P+c*S;f d=M%M;return d<I?C=c,I=d:0;}f D(v p){I=99;P=p
;f l,u,t;v k;for(const char*b="BCJB@bJBHbJCE[FLL_A[FLMCA[CCTT`T";*b;
++b){k.x+=*b/4&15;int o=*b&3,a=*++b&7;k.y=*b/8&7;v d(o%2*a,o/2*a);!o
?l=a/4%2*-3.14,u=a/2%2*3.14,d=p+k*-H,t=atan2(d.y,d.x),t=t<l?l:t>u?u:
t,Q(k*H+v(cos(t),cos(t-1.57))*(a%2*H+1)):Q(k+d*U((p+k*S)%d/(d%d)));}
return M=Q(v(p.x,-.9,p.z))?(int(p.x+64)^int(p.z+64))/8&1?Y:W:v(Y,Y,1
),pow(I,H)-.45;}v R(v o,v d,f z){for(f u=0,l=1,i=0,a=1;u<97;u+=l=D(o
+d*u))if(l<.01){v p=M,n=_(P+C*S),L=_(v(S,1,2));for(o=o+d*u;++i<6;a-=
U(i/3-D(o+n*i*.3))/pow(2,i));p=p*(U(n%L)*H*Y+Y)*a;p=z?p*Y+R(o+n*.1,d
+n*-2*(d%n),z-1)*H*Y:p;u=pow(U(n%_(L+d*S)),40);return p+p*-u+W*u;}z=
d.z*d.z;return v(z,z,1);} int main(){for(puts("P6 600 220 255");++y<
110;)for(f x=-301;P=R(v(-2,4,25),_(_(v(5,0,2))*++x+_(v(-2,73))*-y+v(
301,-59,-735)),2)*255,x<300;putchar(P.z))putchar(P.x),putchar(P.y);}
```


mattz

Method and Variable Naming

Names we choose greatly impact understandability

How readable is code if the variables are a, b, c
(Just think back to the Ray Tracer code !)

Everyone has their own ideas...
    about what makes a good name

Different organisation...
    have their own conventions and standards...

Here are ours !

Variable names should describe the data they hold
Method names should describe action they perform

Anything less than 5 chars is probably too short
Anything greater than 20 chars is getting a bit long

Single words are typically not enough
I favour Verb/Subject names for methods...

Examples of Good Method Names

```
getSurname

setAge

initaliseDataArray

drawNodes

findStringMatches
```

Bad Method Names

```
go

set

calculate

evaluate

enable
```

Accepted "Standard" Terms

Sometimes single words _may_ be acceptable
IF they are standard terms from the domain
OR they are self-evident on their own:

```
run, draw, clone, delete, multiply, connect, filter
```

But why take the risk ?
Does it really hurt to use compound names ?
You can only improve understandability

Method Complexity

'Divide and Conquer' is an oft touted strategy...
Split complex code up into simple sub-procedures
(and sub-sub-procedures)

Avoid massive, hard-to-understand methods
Particularly with complex loop & decision structures
These are very hard to understand (and to change)

Big improvements in understandability can be
achieved by "farming out" code to suitable methods

Simple "Farming Out" Example

Consider a method to check if two numbers are "close"

(e.g. 1 and 2 are close, 1 and 8 are not)

A first attempt might look something like this:

("bad" var names hard to avoid, so code fits on line)

```
int a = int(random(0, 10));
int b = int(random(0, 10));
System.out.println("Numbers are " + a + " and " + b);
if (((a>b)&&((a-b)<2)) || ((a<b)&&((b-a)<2)) || (a==b)) {
  System.out.println("They are close");
}
else System.out.println("They are NOT close");
```

A Simpler, Clearer Solution (?)

```
{
  int firstAge = int(random(0, 10));
  int secondAge = int(random(0, 10));
  System.out.println("Ages are " + firstAge + " and " + secondAge);
  if (differenceBetween(firstAge, secondAge) < 2) {
    System.out.println("They are close");
  }
  else System.out.println("They are NOT close");
}

int differenceBetween(int a, int b)
{
  if (a>b) return a-b;
  else return b-a;
}
```

Minimising Complexity

To minimise complexity, you should try to avoid:

- Very long lines (stretching off side of screen)
- Long methods (stretching off bottom of screen)
- Methods with many parameters (doing too much)
- Deep indentation (many levels of IFs and loops)
- Complex control flow - 'Cyclomatic Complexity'
  (a numerical measure of code complexity)

https://en.wikipedia.org/wiki/Cyclomatic_complexity

Elegance and Replication

Code should be elegant, versatile and minimal

Nice if we can get one method to do the job of 20 !

(Especially if it is a fraction of the size of those 20)

Achieved by "factoring out" common functionality

Placing that common code in an often-called method

This attitude to programming often referred to as

'DRY' (Don't Repeat Yourself)

Some "WET" code

```
public void processCommand(String action, Unit unit)
{
   if(action.equals("add")) {
      System.out.println("ID of student to add ?");
      String id = System.in.readline();
      Student student = cohort.getStudent(id);
      unit.addStudent(student);
   }
   else if(action.equals("remove")) {
      System.out.println("ID of student to remove ?");
      String id = System.in.readline();
      Student student = cohort.getStudent(id);
      unit.removeStudent(student);
   }
}
```

DRYer equivalent

```
System.out.println("ID of student to "+ action +" ?");
String id = System.in.readline();
Student student = cohort.getStudent(id);
if(action.equals("add")) unit.addStudent(student);
if(action.equals("remove")) unit.removeStudent(student);
```

This seems like a trivial improvement to make

But you'd be surprised...
...how much we see code like the WET version

Redundant Code

Whilst we are on the subject of redundant code
What about code that is never actually used at all ?

Happens from time-to-time during evolutionary dev
Trying out some ideas in an experimental method
But never actually linking things in

This is fine, but just be careful not to submit it !
It's easy for checkers to detect this kind of thing ;o)

Higher Level Structural Considerations

Structural Cohesion

Classes should be "cohesive":

"A logical & coherent cluster of data & behaviour"

Aim is to make the purpose of a class very clear

Is

a tumble dryer

that also makes coffee

cohesive ?

Coupling

Classes should also be distinct and independent

You should avoid tight coupling between objects

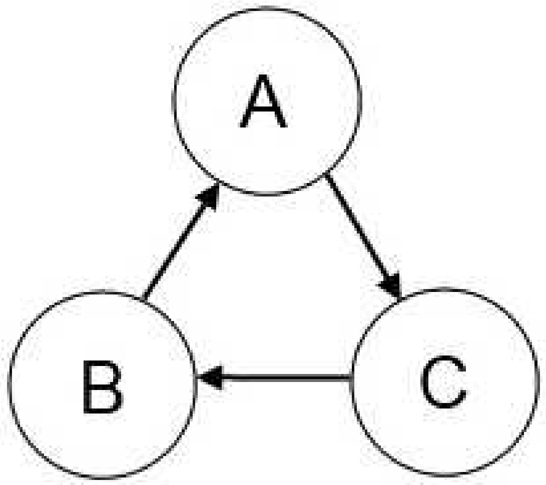
Cyclic Dependency

It is good to have a clear hierarchy of responsibility

Like management structures in an organisation

It is best not to have cyclic loops in these structures

What if your boss was managed by your subordinate !



Problems with Cyclic Dependencies

Responsibility for features is not clearly defined
Maintainer has to cycle round the code searching

Often a sign of arbitrary allocation of responsibility
Developer doesn't have a clear structure in mind
Implements a feature inside inappropriate class

Cyclic loops are also a type of tight coupling
(which we have already talked about)

Questions ?