# Incoherent Rant about Code

Aras Panckevičius
Unity

Unity Bootcamp II, 2010 Nov-Dec

This is pretty much on random code related things with no structure. Expect lots of topic jumps for no reason at all!

## Intro

- Focus on native engine code

I mostly work on native, runtime engine code and hence focus on that

- 15 years of coding
- I know the letters by now
- Don’t know much more

In 15 years of coding I’ve learned something. Mostly the 7 bit ASCII set. And a couple of other things.

- 10 years back
- Loved C++, Boost etc.
- Viewed Design Patterns as essential book

Design Patterns
Elements of Reusable Object-Oriented Software

I used to interview folks for programming positions and ask them about design patterns with a serious face. You don’t know everything about Visitor, no hire sir!

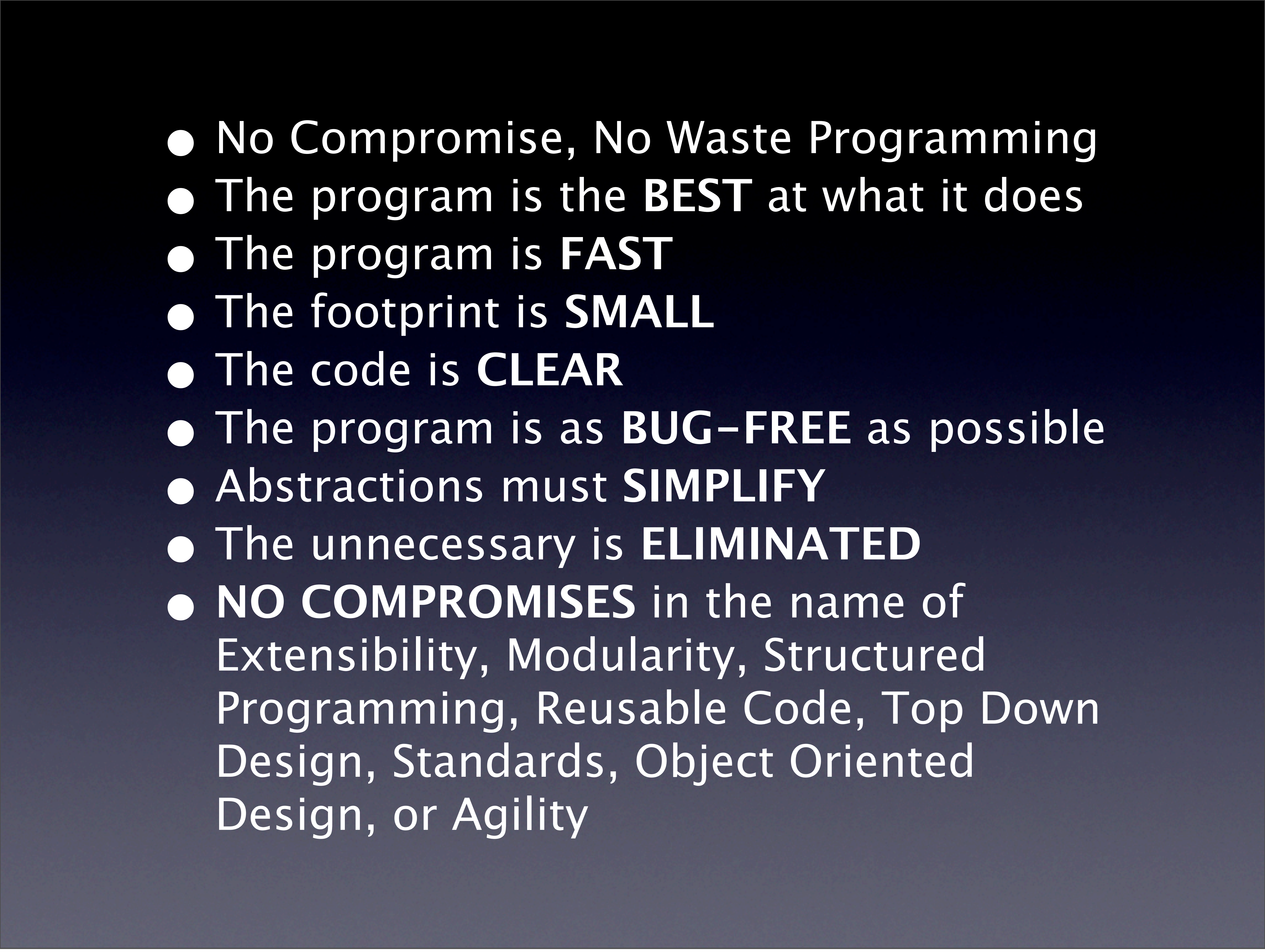
- Now
- Some things have changed
- Mostly, lost some hair

Some of my views have changed. Some will change in next 10 years.

## Some Inspiring Things

- Futurist Programming by Nick Porcino

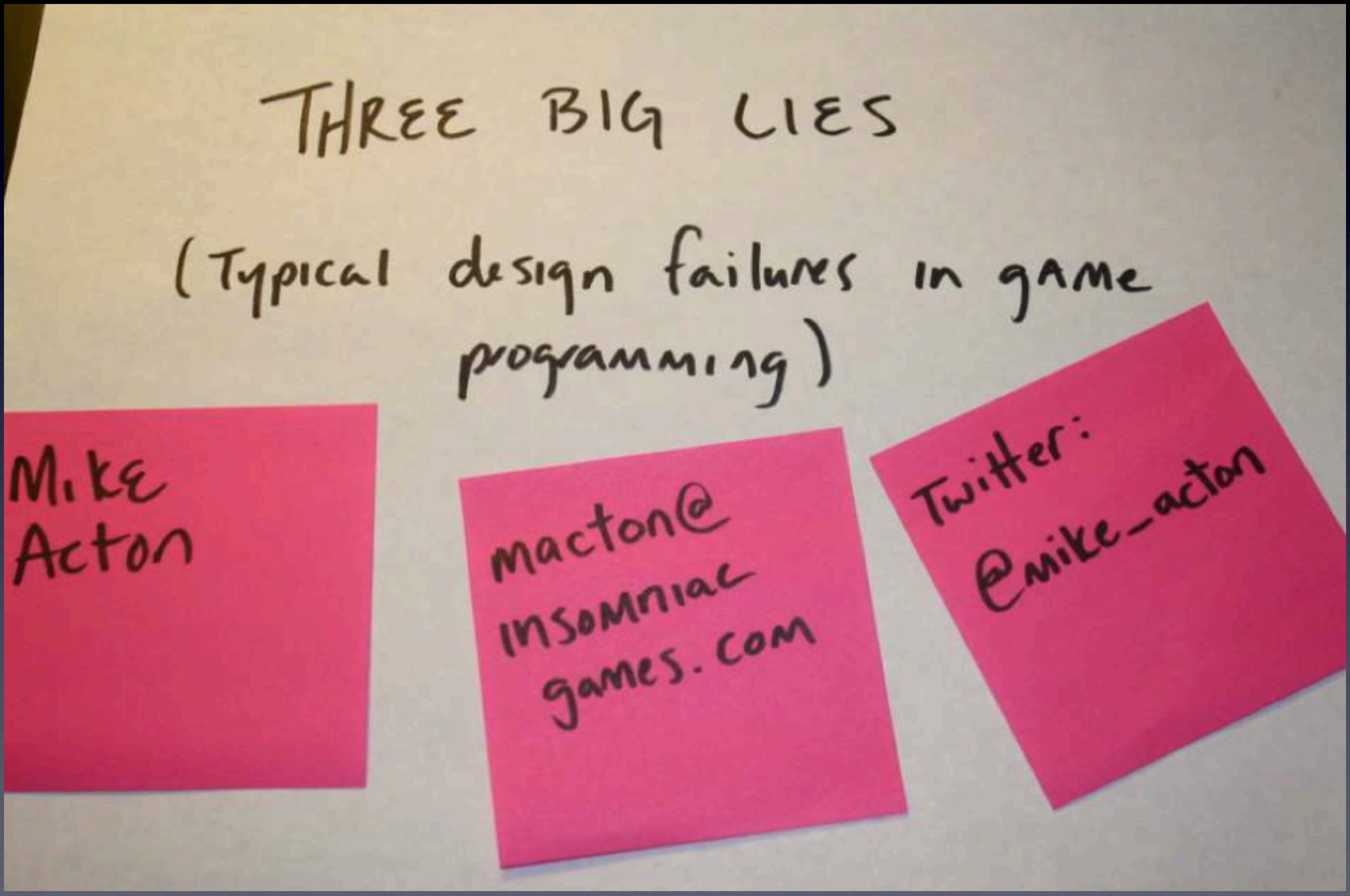
http://meshula.net/wordpress/?p=168



- No Compromise, No Waste Programming
- The program is the **BEST** at what it does
- The program is **FAST**
- The footprint is **SMALL**
- The code is **CLEAR**
- The program is as **BUG-FREE** as possible
- Abstractions must **SIMPLIFY**
- The unnecessary is **ELIMINATED**
- **NO COMPROMISES** in the name of Extensibility, Modularity, Structured Programming, Reusable Code, Top Down Design, Standards, Object Oriented Design, or Agility

This is awesome. I like the abstractions part & the no compromises part. The list of buzzwords there can be extended nowadays, even to include fancy things like Data Oriented Design - nothing should be done in the name of anything. Only for a real actual reason.

- Three Big Lies by Mike Acton



THREE BIG LIES

(Typical design failures in game programming)

Mike Acton

macton@insomniacgames.com

Twitter: @mike_acton

http://www.insomniacgames.com/research_dev/articles/2010/1522262

## Three Big Lies

- Software is a platform
- Code designed around the model of the world
- Code is more important than data

## We’re different

It’s awesome how different we are

- A: dependency inversion card +2!
- B: something something something
- A: nirvana fallacy card +5!
- B: <gives up>

This is a recent long discussion on Unity’s devs mailing list, summarized.

- A: This sucks for these reasons
- B: This sucks for those reasons
- And the reasons don’t overlap

This is several hours of live discussion in the recent Unity dev-leads meeting

- And that is fine!

It’s ok to agree to disagree

## Random Stuff

## All Code Sucks

- Look at your own code of 4 years ago
- If it looks ok, I've got bad news for you
- Throw away vs. Refactor
- DOWIT

All code sucks, by definition. Features & behavior is what you ship; code is a legacy that lingers around and gets in the way. Less code is almost always better.
If I look at my old code, it always sucks. When in the future I'll look at my current code, I hope I'll think it sucks.
Maybe in the future we'll look at our brand new shiny data-oriented-design code and laugh at it, thinking DOD in 2010 is like design patterns in 1996. Who knows.
Since all code sucks, it can either be changed or thrown away; and which one to choose depends on a lot of things. I think it's important to do either of those and not let the sucky code rot. It will be sucky in the future, and that's fine; but make it better for today.

## Data Oriented

- Data in, data out
- Code transforms the data
- Discussion/rountable one of these days

Topic jump: all the cool kids are using data oriented thingy now, and we'll have a discussion/roundrable on that tomorrow or one of these days. Everyone invited!

## Optimization

Topic jump again! Little section on optimization

- Optimization is the root of all evil

This guy, Donald, is quoted with this a lot

- Slap with a large trout!

My reaction to this

- We should forget about **small** efficiencies, say about 97% of the time: **premature** optimization is the root of all evil"
- Context!

Supposedly full quote above, emphasis mine. Also, “small efficiencies” was something like “20 cycles”, and not big performance characteristics in general.

Also, everything depends on context. Performance can be important or can be not. It can also be critical, especially in our field.

- Optimized
- **Optimizable**
- There's a difference!

There's a big difference between optimized and optimizable. Not optimized right now is fine, as long as it's not some monstrosity that can't ever be optimized because it's messed up. Alas, that's not true for lots of our code; we have lots of places that are hard to optimize even if we want to...

## Future Proof

Another random thing!

- Prediction is very difficult, especially about the future.

A guy called Niels said this

- Think ahead
- Enough to not put yourself into corner
- Simplest thing that works
- **Change** in the future

It’s good to think ahead... but not too much!
You can’t predict the future; the situation can and will change. And will change in unexpected ways.
When conditions will change, you will change the code or design – it’s really that simple.

The above does not quite apply to some situations. For example, Unity’s scripting API – when anything is added, it’s almost for “forever”. So you have to think quite a lot when designing it.

## False “future proof”

- Get/Set accessors
- “If data type changes, won’t have to change code”
- Bullshit!

A silly example of false “this is future proof”: when someone says that Get/Set accessor functions are future proof because if underlying variable type changes, or goes away completely (i.e. is computed), the client code won’t have to change.
This is stupid! If something in the data changes, there was a reason for it. Better review all the users of that code; they might need to be changed accordingly.
What I often do when changing something: I don’t use some refactoring tools in the IDE to change stuff. I just rename it and see all the compile errors. That forces me to go through all places and review them.

## Good Code
## Bad Code

Favorite topic: good vs. bad code

## Good Code

- Works
- Solves the right problem
- Understandable / Hackable

Good code has these qualities IMO

## Not Necessarily

- Test Coverage
  - Can be important for “works” and “hackable”
- Design Idioms
  - OOD/OOP, Design Patterns, DOD
  - ...every other one, really
- Curly braces on “right” lines

Good code does not necessarily have test coverage, any particular design idiom behind it, or curly braces on the correct lines.

Tests can be important to ensure other good qualities of code itself of course.
Some design idiom or though behind the code can be important to ensure it works, and within required performance/whatever needs.

## Works

- Duh!

Code needs to work!

## Solves the Right Problem

- Coders like to drift
  - see Architecture Astronauts
- Reality check
- Ask someone

A lot of coders like to drift away from the problem; inventing problems that may not exist. Or only might exist in some hazy future (i.e. never).
And then you end up with some architecture monstrosities that create more problems than they solve.
Reality check helps! Just talk to someone else and discuss what you're about to do.

## Understandable

- Makes sense
  - Requiring domain specific knowledge is ok!
- Code flow is clear
- **Data** flow is clear!

Good code is understandable. It’s perfectly fine to require domain specific knowledge! I don’t expect a database guy to go into SIMD skinning code and understand it; and vice versa.
But for someone who knows the domain, getting into some code should be reasonably easy.
Code flow and more importantly, data flow should be clear enough.
Do we have enough of that in our own code?

## Not Necessarily

- Short functions/classes/...
  - short = more scattered around
- Model the world
  - software is not the world

Understandable code does not necessarily mean “short functions/classes/whatever” or “descriptive variable names”. Shorter code blocks means the whole code is scattered around more, so you have to look at more places to really understand it. Likewise, shorter variable names (within reason) lets you read code faster. “mat” or even “m” is perfectly fine for a matrix.

Understandable code does not necessarily model the world. Most of interesting and hard problems in software do not even have a real world equivalent!

## Hackable

- Does it feel good to work on it?
- Can you accomplish anything?

Understandable code is very often easily hackable code. Does it feel good to work in it, or are you deep in some misery? Can you accomplish your task within reasonable time, or are you hindered by some monstrosities in the code?

## Hackable

- My own recent example:
- HLSL2GLSL: horrible, painful
- Mesa GLSL: awesome, smooth

Recent example from my own memory. I had to work with two external libraries not written by us; to translate/optimize shaders for mobile platforms. Both of them are quite similar, in that they involve parsing shaders, constructing internal representations of them and then doing something on that.

One of them was HLSL2GLSL. Each time I have to work with it, it's a pain. It gets the job done, i.e. it works; and it solves the right problem ("HLSL to GLSL!"), but the code is a crap. Seems like someone had a textbook idea what OOP is and applied that everywhere without any thinking. Perhaps another reason was multiple different code owners; looks like the project started from "reference GLSL parser" from 3DLabs; preprocessor was made by NVIDIA, and then actual translation part done by ATI. What good can come out of that?!

On the other hand, Mesa's GLSL compiler is a pleasure to work with. No bullshit, simple code. No fancy ideologies, just code that does what it's supposed to do.

## Hackable

- How many places of our codebase is a pleasure to work in?
  - See what I did there?
- ...but we own the code. We can make it better!

Trick question: how many places in Unity’s codebase are a pleasure to work with? Yeah.
But the good thing is, we own that code. You don’t have to buy a source license to change it; just check it out and improve it. Dowit! Naow!

## Flame?

And the flaming continued...