A Plastic Skeleton

and associated ramblings

Prepare your bingo cards.

encapsulation	antifragile	Hemispheric integration	characteristic scale
OODA	Abstraction	Adaptability	Culture
implicit	Plasticity	Any extended metaphor	Overreaching generalisations
LaTeX	structural		A Plastic Skeleton
monorepo	Pointless semantic nonsense	explicit	adaptive

Warning, opinions inside

This is mostly about 1 to N,
not 0 to 1

What is good code?

Yes really I’m sorry it’s that sort of talk

What is good code?

Yes really I'm sorry it's that sort of talk

- At some time *t*, the code could be considered good if using it brings benefit.

What is good code?

Yes really I'm sorry it's that sort of talk

- At some time t, the code could be considered good if using it brings benefit.
- But the code is - some might say sadly - still around at $t + \Delta t$ and we will want it to be providing benefits then, too!

What is good code?

Yes really I'm sorry it's that sort of talk

- At some time t, the code could be considered good if using it brings benefit.
- But the code is - some might say sadly - still around at $t + \Delta t$ and we will want it to be providing benefits then, too!
- So in fact, the value of the code is the present value of all its future costs and benefits.

What is good code?

Yes really I'm sorry it's that sort of talk

- At some time t, the code could be considered good if using it brings benefit.
- But the code is - some might say sadly - still around at $t + \Delta t$ and we will want it to be providing benefits then, too!
- So in fact, the value of the code is the present value of all its future costs and benefits.
- But - equally applicable to assets generally as it is to code - how on earth can we know the value of some benefit or cost in the future?

Well, maybe we can anticipate future demands and circumstances?

Well, maybe we can anticipate future demands and circumstances?

Well, maybe we can anticipate future demands and circumstances?

YOUR HUBRIS WILL BE PUNISHED

Well if the only way of telling good from bad is to know the future

And I can’t anticipate the future

How can I write good code?

You can make your code easy to **change**

Decoupling, composability, all that good stuff?

Well, yes and no.

Often small code - not painfully squeezed code, just no bigger than is necessary to comfortably do the job - is going to be easier to change than your cleverly designed super-composable code

YAGNI?

YAGNI?

YAGNI?

Yes, but be careful.

Fear Of The Code
Fear Of The Code
I have a constant fear
the code is getting large

Fear Of The Code
Fear Of The Code
I have a paranoia
I might have to code

We should be afraid of code

We should be afraid of code

But not of changing code

Making design decisions that make adding features very small diffs, doing so is back to predicting the future: you don't know what features will be needed, so you don't know what flexibility to include.

Of course, if the cost is low, do it! Some things are predictable & cheap to prepare for.

Don’t be afraid of actually hitting the keyboard a bunch.

The complexity/complicatedness of the code, not number of lines you touch today.

Flexibility vs Adaptability

Building in flexibility ahead of time is optimising for a set of known knowns & known unknowns. You don't treat them all equally (some you ignore entirely). And what about the unknown unknowns? The future is a strange place.

The only way I know to prepare for the unknown is to be able to change, to be adaptable.

Adaptability is more universal than flexibility

This is the key property that makes it better able to handle the future.

Coding is creative

Creativity is mostly listening

Listening is hard when it’s noisy (Boilerplate, extraneous fluff, muddy sound)

Adaptable/plastic features of D

Adaptable/plastic features of D

- The basic building block: `struct`

Adaptable/plastic features of D

- The basic building block: `struct`
- The GC

Adaptable/plastic features of D

- The basic building block: `struct`
- The GC
- Sane metaprogramming

Adaptable/plastic features of D

- The basic building block: `struct`
- The GC
- Sane metaprogramming
- template functions + `static if`

Adaptable/plastic features of D

- The basic building block: `struct`
- The GC
- Sane metaprogramming
- template functions + `static if`

- Code moulds itself statically to other code

Adaptable/plastic features of D

- The basic building block: `struct`
- The GC
- Sane metaprogramming
- template functions `+ static if`

- Code moulds itself statically to other code
- Some untellable combination of features that fit together

Adaptable/plastic features of D

- The basic building block: `struct`
- The GC
- Sane metaprogramming
- template functions + `static if`

- Code moulds itself statically to other code
- Some untellable combination of features that fit together
- Encapsulation of awfulness

Adaptable/plastic features of D

- The basic building block: `struct`
- The GC
- Sane metaprogramming
- template functions + `static if`

- Code moulds itself statically to other code
- Some untellable combination of features that fit together
- Encapsulation of awfulness
- D, at is best, is SaneHackerLang

```
override string toString() @safe const {
    import std.conv : text;
    import std.string : toLower;
    import std.algorithm : map;
    import std.array : join;

    final switch (tag) with (Tag) {
        static foreach (t; [TVoid, TInteger, TNumber, TChar, TBoolean,
                            TAny]) {
            case t:
                return t.text.toLower[1 .. $];
        }

        static foreach (t; [TUni, TOverloads, TStruct, TFun, TVar,
                            TNamed]) {
            mixin(`case t: return `, getterName(t), `.toString;`);
        }
    }
}
```

```
// somePackage/someModule.d
Nullable!int foo(string a, FancyInt b);
FancyInt bar(FancyInt[] r);
// and so on ...

// inside a function that is wrapping code for SIL
handlers.wrapAll!(
    TypeMaps!(
        ParamTypeMaps!((long x) => FancyInt.sillyCtor(x)),
        ReturnTypeMaps!((FancyInt x) => x.getInt)),
    "somePackage.someModule");
```

Flexibility of structs & metaprogramming:
Good because you can customise without (much) compromise or breakage
Bad because you accumulate shims and cleverness

D Culture

Mostly small codebases

Mostly library/language work

When I first started to working professionally with D I went to great effort to break everything up in to packages & separate repos, because that was the default that came from the community

I was wrong, I should have focused on the problem at hand

Scope & monorepos

Scope & monorepos

- The D community is heavily biased towards language & library developers

Scope & monorepos

- The D community is heavily biased towards language & library developers
- This is great, it tends to mean people have an interest in “doing it right” and making really great, usable abstractions

Scope & monorepos

- The D community is heavily biased towards language & library developers
- This is great, it tends to mean people have an interest in “doing it right” and making really great, usable abstractions
- But it also is a heavy bias towards quite “isolated” development, where it assumed that the consumers of your work interact with you by the occasional complaint or bugfix, but mostly are silent

Scope & monorepos

- The D community is heavily biased towards language & library developers
- This is great, it tends to mean people have an interest in “doing it right” and making really great, usable abstractions
- But it also is a heavy bias towards quite “isolated” development, where it assumed that the consumers of your work interact with you by the occasional complaint or bugfix, but mostly are silent
- Building internal software at a larger company setting you have many users, but you have the luxury of actually knowing them! You can probably see all their code and you have some common goals.

Scope & monorepos

- The D community is heavily biased towards language & library developers
- This is great, it tends to mean people have an interest in “doing it right” and making really great, usable abstractions
- But it also is a heavy bias towards quite “isolated” development, where it assumed that the consumers of your work interact with you by the occasional complaint or bugfix, but mostly are silent
- Building internal software at a larger company setting you have many users, but you have the luxury of actually knowing them! You can probably see all their code and you have some common goals.
- Breaking changes vs stable interfaces have a different trade off here

Scope & monorepos

- The D community is heavily biased towards language & library developers
- This is great, it tends to mean people have an interest in “doing it right” and making really great, usable abstractions
- But it also is a heavy bias towards quite “isolated” development, where it assumed that the consumers of your work interact with you by the occasional complaint or bugfix, but mostly are silent
- Building internal software at a larger company setting you have many users, but you have the luxury of actually knowing them! You can probably see all their code and you have some common goals.
- Breaking changes vs stable interfaces have a different trade off here
- Putting everything in one repo can really help with breaking down the clever barriers and just having straightforward code that does something

Pull don’t push

OODA loop of coding:

Observe: read the code, read the diffs, read the user feedback, read other developer's feedback

Orient: understand what the code is doing, how it structured, what about it is lacking for the endpoint you desire

Decide: work out roughly what the next diff will be

Act: write the diff and push it

Come work for us!

jcolvin at symmetryinvestments dot com

https://bit.ly/3Joc4GO

Good coding practice is antifragile