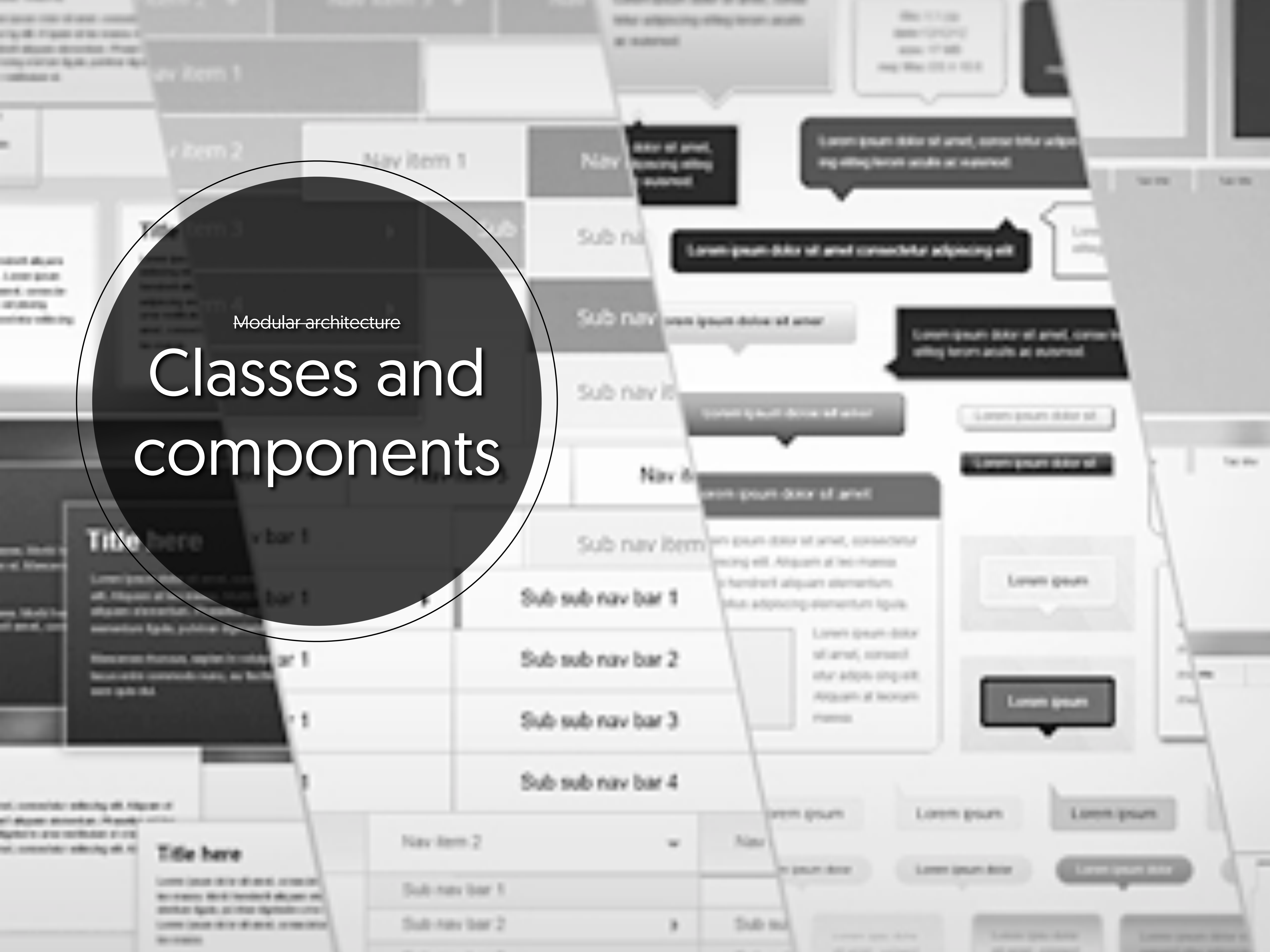
Modular architecture



~~Modular architecture~~

Classes and components

~~Modular architecture classes and~~

Components and modifiers

BLOCK, ELEMENT, MO

~~Modular architecture classes and~~

Components, modifiers and overrides

~~Modular architecture~~
~~classes,~~
~~modifiers~~
~~and overrides~~

Components, patterns and sh*t it's hard to deal with

@cedmax

*Webmaster
before it was cool*

Tech Lead
Condé Nast International

Components, patterns and sh*t

it's hard to deal with

Components, patterns and sh*t

it’s hard to deal with

or… *How I came up with a good use of quotes from Lost in Translation*

Lost in Translation?

Basically

Disclaimer

"This movie is an hour and some odd minutes of my life I will never get back."

JoeB. on Metacritic

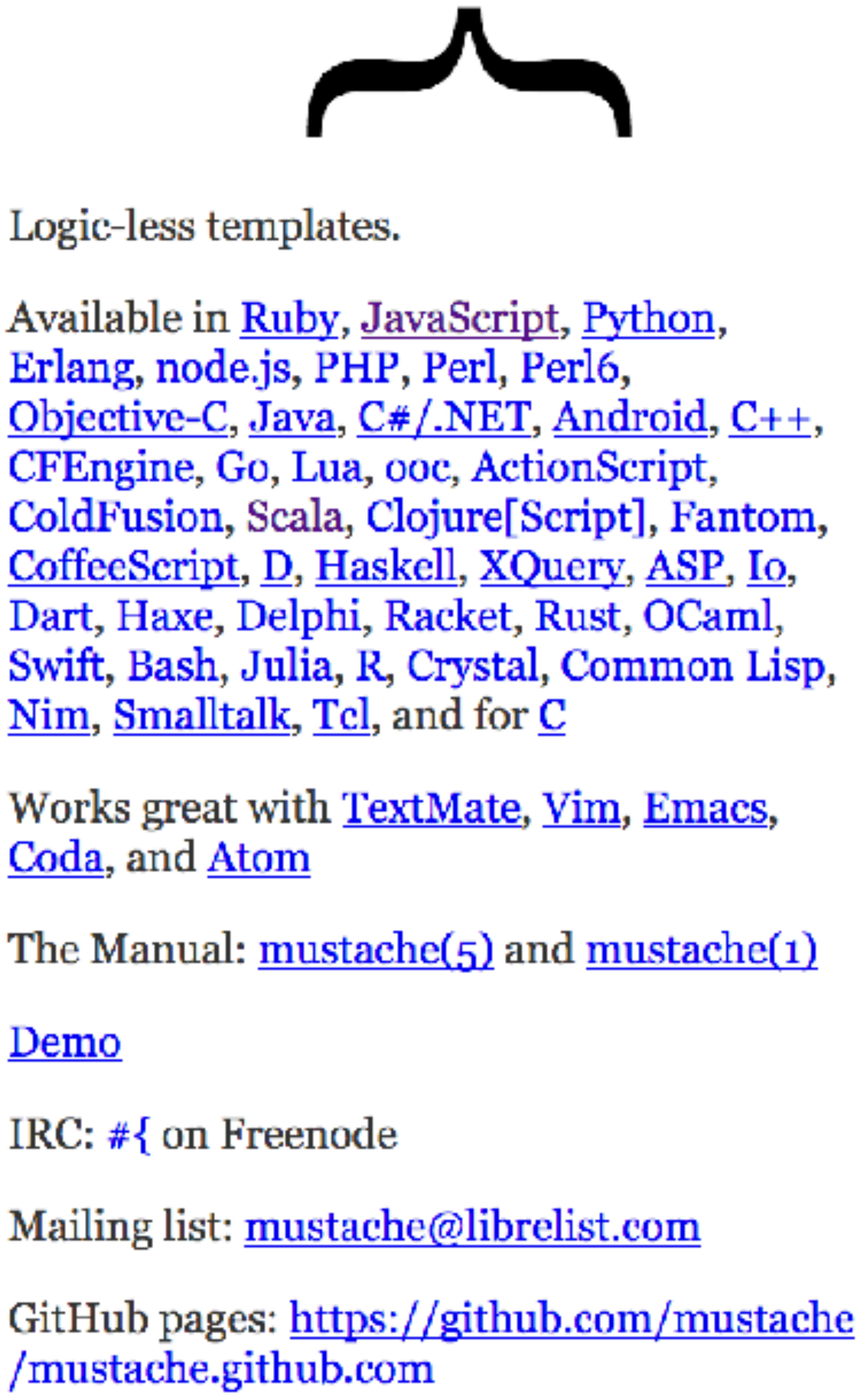
Lost in Translation

"Meaning is complex and often gets lost in translation. Everybody has their own mental model of things"

Alla Kholmatova

Modular design

2013 - 2015



Logic-less templates.

Available in Ruby, JavaScript, Python, Erlang, node.js, PHP, Perl, Perl6, Objective-C, Java, C#/.NET, Android, C++, CFEngine, Go, Lua, ooc, ActionScript, ColdFusion, Scala, Clojure[Script], Fantom, CoffeeScript, D, Haskell, XQuery, ASP, Io, Dart, Haxe, Delphi, Racket, Rust, OCaml, Swift, Bash, Julia, R, Crystal, Common Lisp, Nim, Smalltalk, Tcl, and for C

Works great with TextMate, Vim, Emacs, Coda, and Atom

The Manual: mustache(5) and mustache(1)

Demo

IRC: #{ on Freenode

Mailing list: mustache@librelist.com

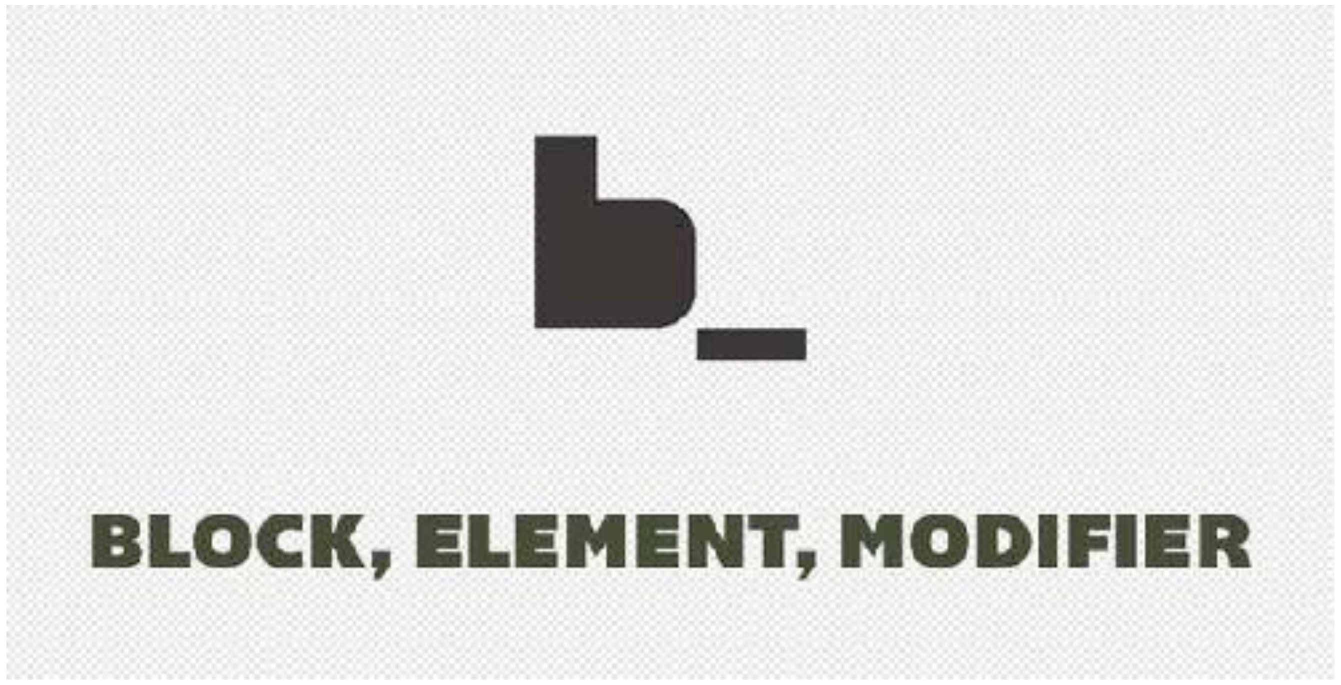
GitHub pages: https://github.com/mustache/mustache.github.com

Akase

a small decoupled, event-driven architecture framework.

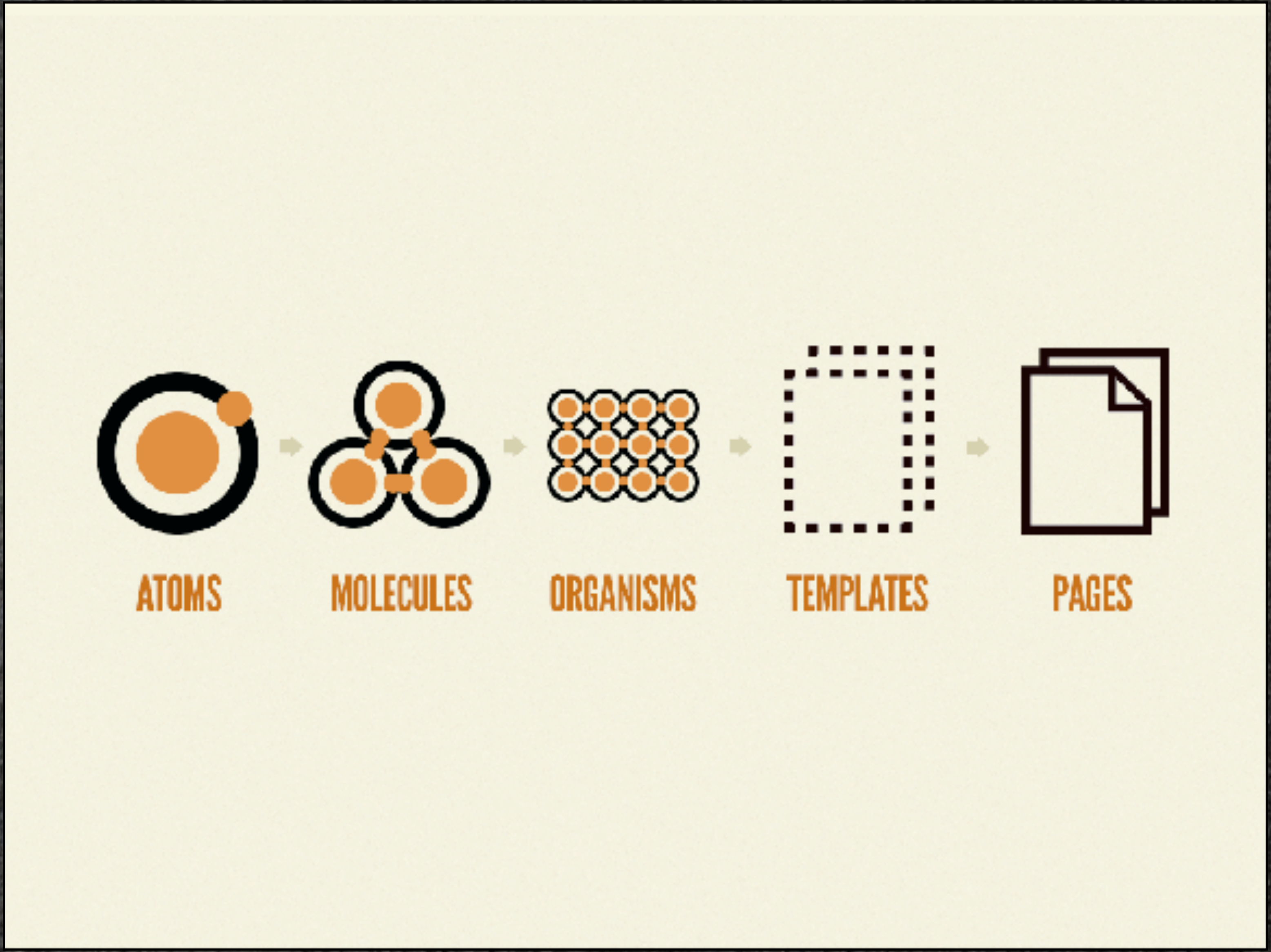
ākāśe

ākāśe (sanskrit for "in the sky"/"to the sky") is a small decoupled, event-driven architecture framework. It is based on Nicholas Zakas Scalable Javascript Application Architecture and RequireJS and AMD.



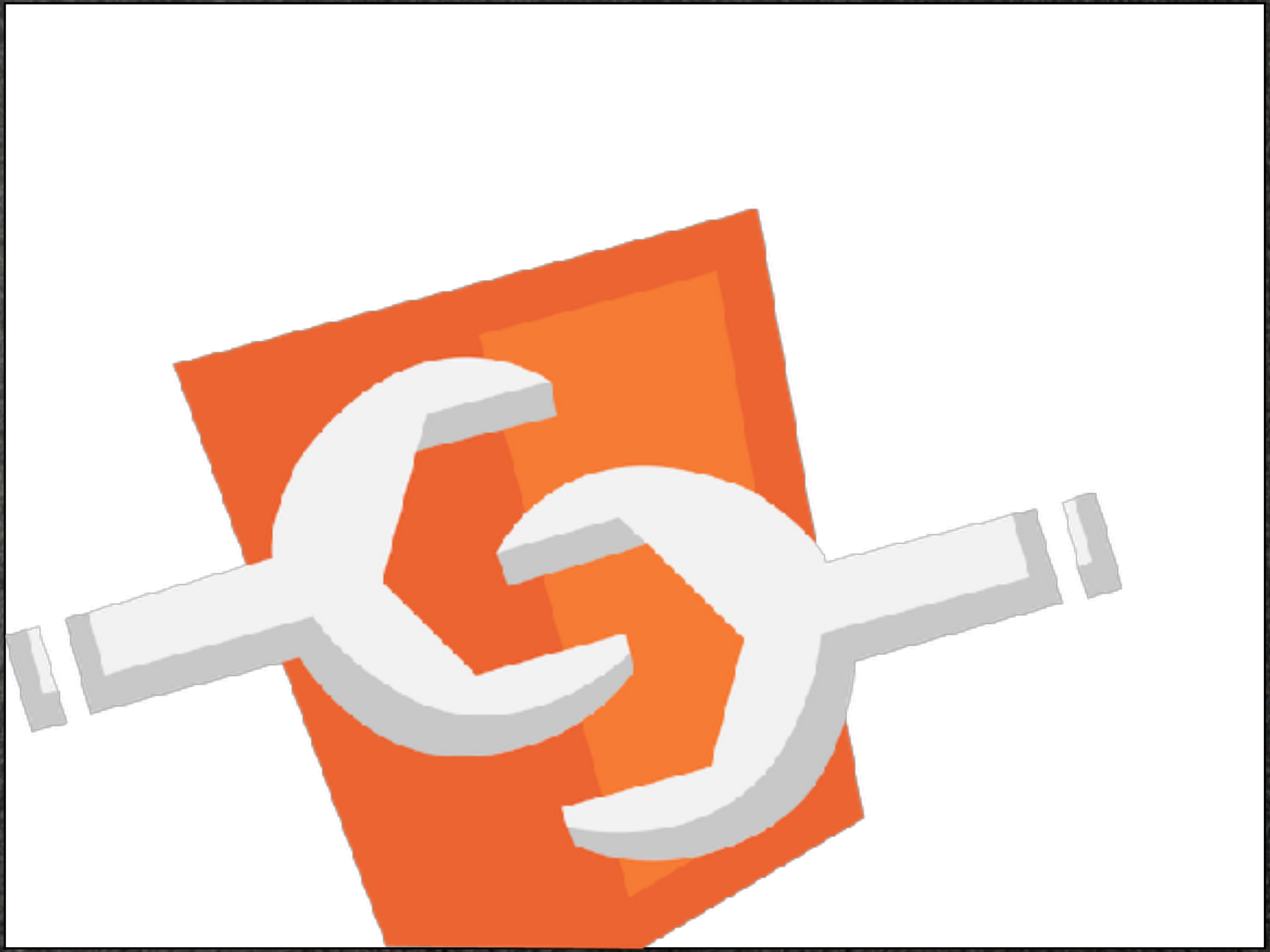
Atomic design

Brad Frost · October 2013



Web components

announced in November 2011



Pattern Library

"Pattern libraries are something I do a lot for client projects. [...] It's a technique I first saw [...] Natalie Downe develop for client projects back in 2009"

Anna Debenham

MISSING SLIDE* ABOUT PATTERN LIBRARIES

* on purpose, I promise

Pattern Library

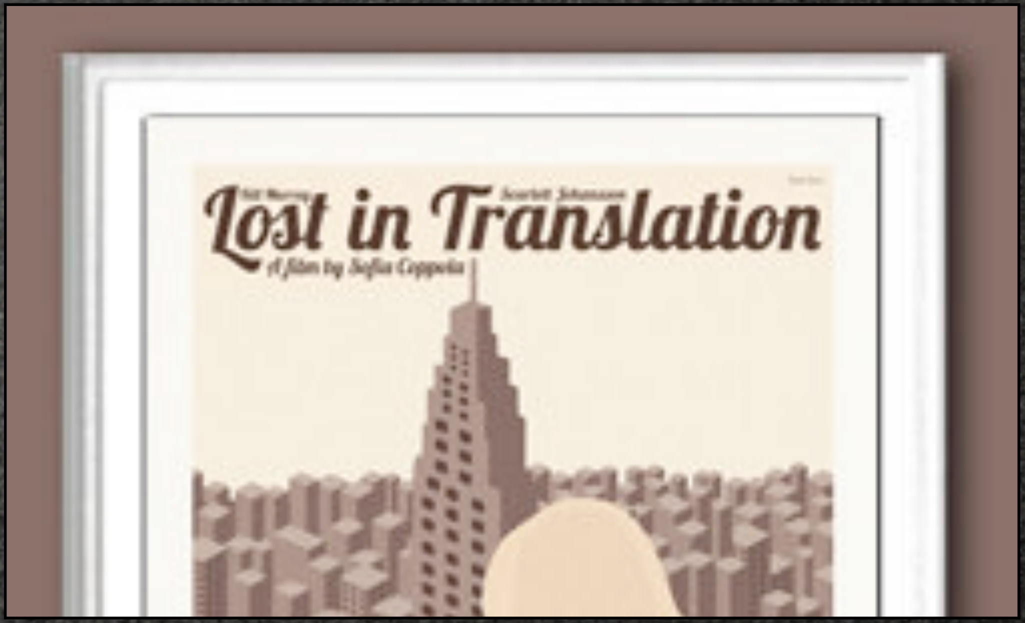
“Pattern libraries are something I do a lot for client projects. [...] It’s a technique I first saw [...] Natalie Downe develop for client projects back in 2009”

Anna Debenham

ReactJS

First release: March 2013

Where are we at, today?



Frame the issue

Basically

It's not that simple

“When you actually try to apply a modular approach to your day to day work, it isn’t really that simple”

Alla Kholmatova · June 2015

The issue

The issue

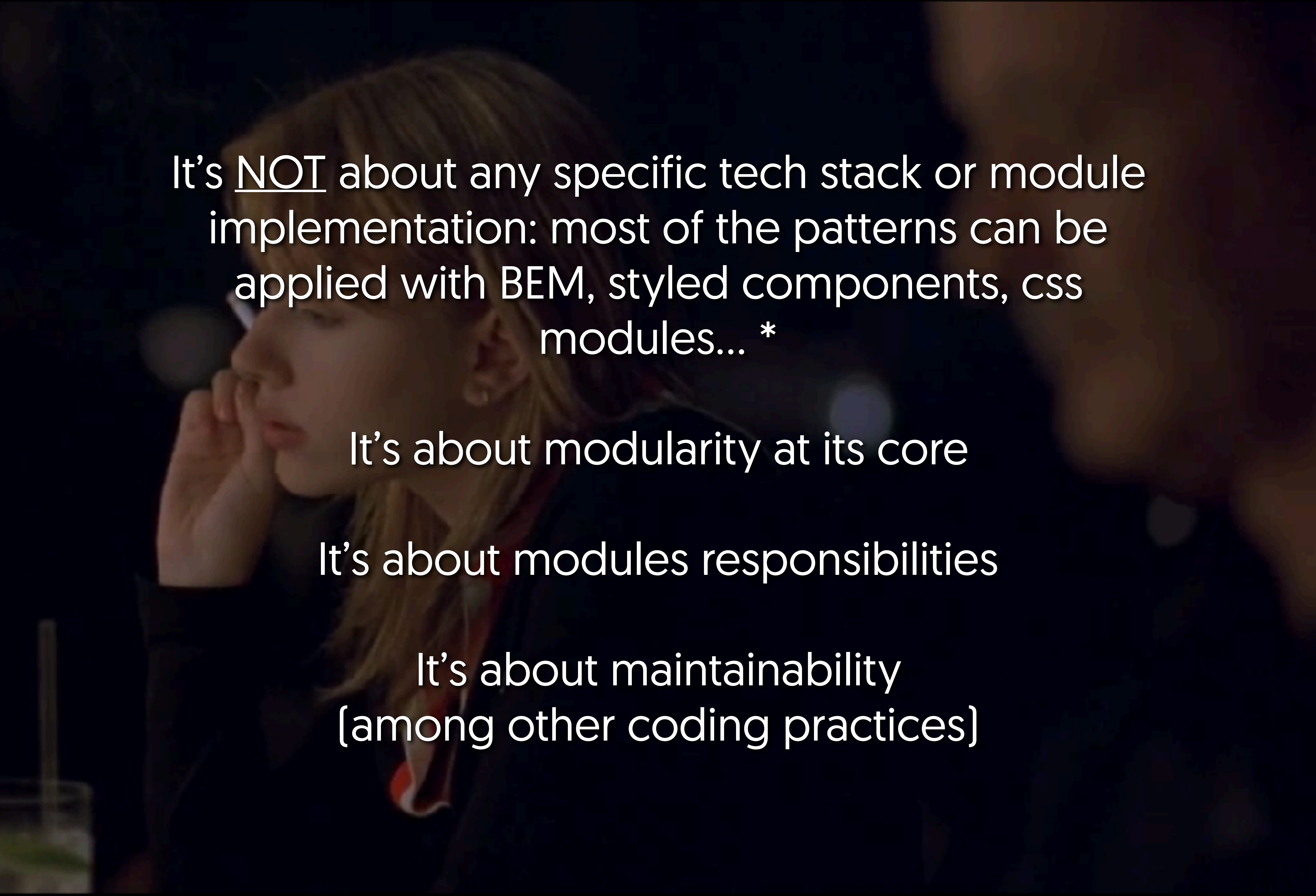
How do we manage our code, to re-use patterns without making them too rigid for the day to day activities?

The issue

~~How do we manage our code, to re-use patterns without making them too rigid for the day to day activities?~~

How do we re-use our patterns in slightly different use cases?

Wish I could sleep



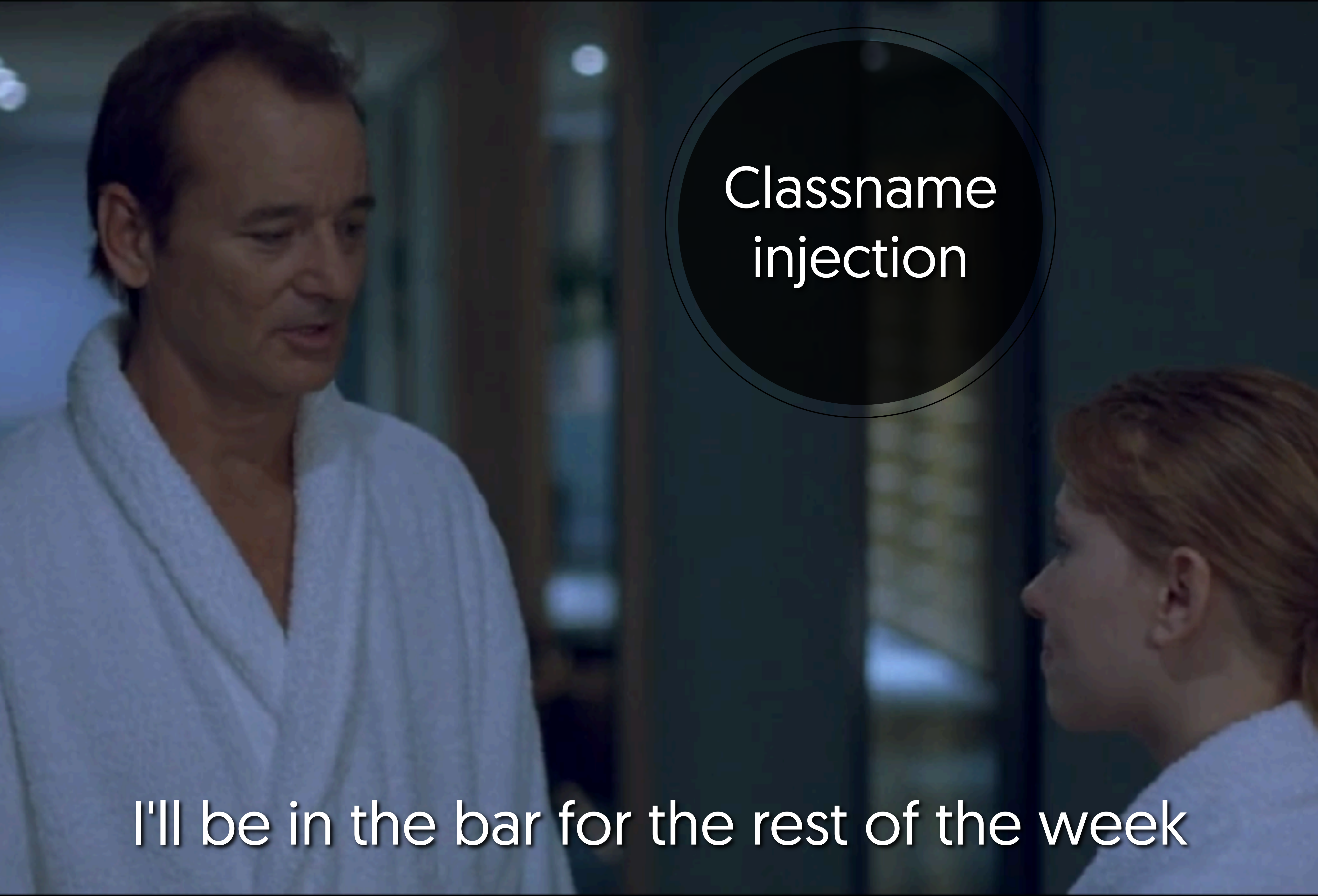
It's NOT about any specific tech stack or module implementation: most of the patterns can be applied with BEM, styled components, css modules… *

It's about modularity at its core

It's about modules responsibilities

It's about maintainability
(among other coding practices)

Classname injection



I'll be in the bar for the rest of the week

```
<IconButton
  className="content-actions__button"
  iconId="close"
/>
```

```
<IconButton
  className="content-actions__button"
  iconId="close"
/>
```

```scss
//_content-actions.scss
.content-actions {
  //[...]
  &__button {
    flex: 1 0 auto;
    padding: 1rem;
    line-height: 1.5;

    &:hover, &:focus {
      background: $grey-1;
    }

    &:active {
      background: $grey-2;
    }
  }
}
```

```scss
//_content-actions.scss
.content-actions {
  //[...]
  &__button {
     flex: 1 0 auto;
     padding: 1rem;
     line-height: 1.5;

     &:hover, &:focus {
       background: $grey-1;
     }

     &:active {
       background: $grey-2;
     }
  }
}
```

```scss
//_content-actions.scss
.content-actions {
  //[...]
  &__button {
    flex: 1 0 auto;
    padding: 1rem;
    line-height: 1.5;

    &:hover, &:focus {
      background: $grey-1;
    }

    &:active {
      background: $grey-2;
    }
  }
}
```

What's the effect on the base button?

```scss
//_content-actions.scss
.content-actions {
  //[...]
  &__button {
    flex: 1 0 auto;
    padding: 1rem;
    line-height: 1.5;

    &:hover, &:focus {
      background: $grey-1;
    }

    &:active {
      background: $grey-2;
    }
  }
}
```

Why is this button different from the pattern library ones?

What works

This is the most flexible way to extend anything.

What really doesn't

1. The default style could be overridden in unexpected ways.

2. We are creating many variants of the original patterns.

Ad hoc modifiers

- You're too tall.
- Anybody ever tell you you may be too small?

```
<Dialog
  className="dialog--user-intent">
  <!-- [...] -->
</Dialog>
```

```
<Dialog
  className="dialog--user-intent">
  <!-- [...] -->
</Dialog>
```

```scss
//_dialog.scss
.dialog {
  //[...]

  &--user-intent {
    width: 43.75rem;
    height: auto;
  }
}
```

```scss
//_dialog.scss
.dialog {
  //[...]

  &--user-intent {
    width: 43.75rem;
    height: auto;
  }
}
```

```scss
//_dialog.scss
.dialog {
  //[...]

  &--wizard {
   width: 43.75rem;
   height: 35rem;
  }

  &--game-intent {
    width: 43.75rem;
    height: auto;
  }

  &--save-results {
    width: 23.75rem;
    height: auto;
  }
}
```

How many variants do we have to account for?

What works

This practice allows for flexibility, giving a reasonable control and keeping all the variants in proximity.

What really doesn't

1. The generic component style have knowledge of specific implementations.

2. The file size might be effected by unused code.

3. It doesn't scale

Specialised patterns

```
<Dialog
  className="dialog--prompt">
  <!-- [...] -->
</Dialog>
```

```
<Dialog
  className="dialog--prompt">
  <!-- [...] -->
</Dialog>
```

```scss
//_dialog.scss
.dialog {
  //[...]

  &--prompt {
    display: block;
    overflow: hidden;
    max-width: map-get($dialog-prompt, max-width);
    height: auto;
    margin: map-get($dialog-prompt, margin);
    padding: 2rem 0 0;
    border-radius: 3px;
  }
}
```

```scss
//_dialog.scss
.dialog {
  //[...]

  &--prompt {
    display: block;
    overflow: hidden;
    max-width: map-get($dialog-prompt, max-width);
    height: auto;
    margin: map-get($dialog-prompt, margin);
    padding: 2rem 0 0;
    border-radius: 3px;
  }
}
```

The semantic value of the modifiers is different from the ad-hoc ones.

What works

The patterns are at the centre: no special cases, but pre-defined flavours of the basic components.

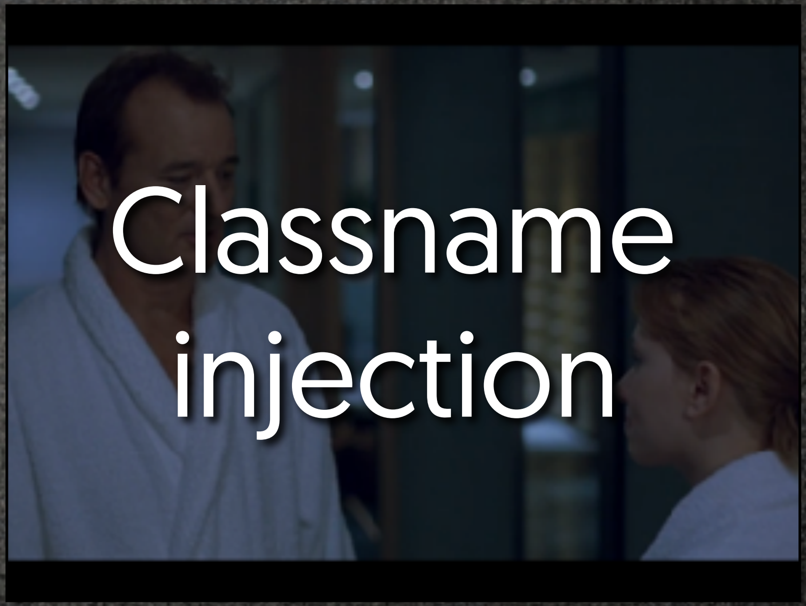
What really doesn't

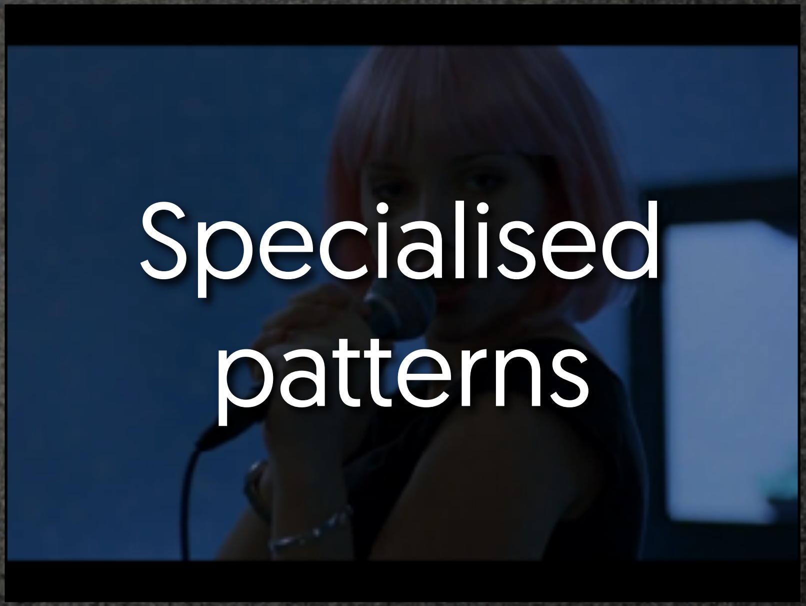
1. It might drive to preemptive abstraction
2. It does account for a finite number of use cases

```
<Dialog
  className="dialog--prompt">
  <!-- [...] -->
</Dialog>
```

```
<Dialog
  type="prompt" />
```

```
<DialogPrompt />
```



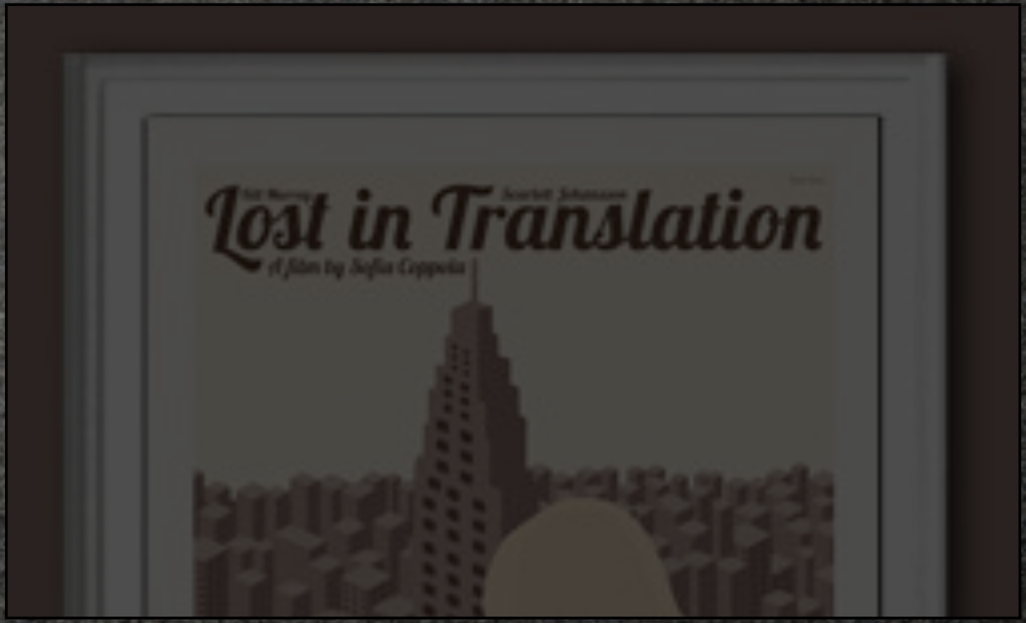


A no go: it defies the point of having a pattern library

A code smell, it's an hack and it should be treated like one

The best approach, even though sometimes

I still wish I could sleep



I'm stuck

Basically

It's not that simple

“It isn’t really that simple”

Alla Kholmatova · June 2015

The issue

How do we re-use our patterns in slightly different use cases?

What am I trying to solve?

Arrangement within parent components

```
<div
  className="game-intent__dialog">
    <Dialog>
      <!-- [...] -->
    </Dialog>
</div>
```

```
<div
  className="game-intent__dialog">
    <Dialog>
      <!-- [...] -->
    </Dialog>
</div>
```

```scss
//_dialog.scss
.dialog {
  width: 100%;
  height: 100%;

  //[...]
}

//_game-intent.scss
.game-intent {
  //[...]

  &__dialog {
    width: 43.75rem;
    height: auto;
  }
}
```

```scss
//_dialog.scss
.dialog {
  width: 100%;
  height: 100%;

  //[...]
}

//_game-intent.scss
.game-intent {
  //[...]

  &__dialog {
    width: 43.75rem;
    height: auto;
  }
}
```

Each component has its own responsibility

What works

This practices defines responsibilities in a neat way and it enables for specific implementations without invalidating patterns.

```
<Dialog
  className="custom-class">
  <!-- [...] -->
</Dialog>

<div
  className="custom-class">
    <Dialog>
      <!-- [...] -->
    </Dialog>
</div>
```

What really doesn't

Potentially you might need a wrapper HTML element that could have been avoided.

Space in relation to other components

```
<Dialog
  className="space-max inner-space-min">
  <!-- [...] -->
</Dialog>
```

```
<Dialog
  className="space-max inner-space-min">
  <!-- [...] -->
</Dialog>
```

What works

It reduces the need to come up with new class names and it moves the conversation regarding component relationships back to the pattern library.

What really doesn't

1. The positional classes might get stale if not codified properly in the pattern lib.
2. The flexibility of the helper classes is limited
3. Do you like atomic css? https://acss.io/

```
<Dialog
  className="M(defSpace) P(defSpace)">
  <!-- [...] -->
</Dialog>
```

"Open" components

```scss
//_question-content-block.scss
.question-content-block {
  //[...]
  &__icon-button {
    //[...]

    .icon {
      width: $content-block-icon-large-size;
      height: $content-block-icon-large-size;
    }
}
```

```
//_question-content-block.scss
.question-content-block {
  //[...]
  &__icon-button {
    //[...]

    .icon {
      width: $content-block-icon-large-size;
      height: $content-block-icon-large-size;
    }
}
```

```scss
//_question-content-block.scss
.question-content-block {
  //[...]
  &__icon-button {
    //[...]

    @include icon-size($content-block-icon-medium-size);
  }
}

//_icon.scss
@mixin icon-size($size) {
  .icon {
    width: $size;
    height: $size;
  }
}
```

```scss
//_question-content-block.scss
.question-content-block {
  //[...]
  &__icon-button {
    //[...]

    @include icon-size($content-block-icon-medium-size);
  }
}

//_icon.scss
@mixin icon-size($size) {
  .icon {
    width: $size;
    height: $size;
  }
}
```

```scss
//_question-content-block.scss
.question-content-block {
  //[...]
  &__icon-button {
    //[...]

    @include icon-size($content-block-icon-medium-size);
  }
}

//_icon.scss
@mixin icon-size($size) {
  .icon {
    width: $size;
    height: $size;
  }
}
```

The responsibility of being flexible it back to the component itself

```
<Icon size={32} />
```

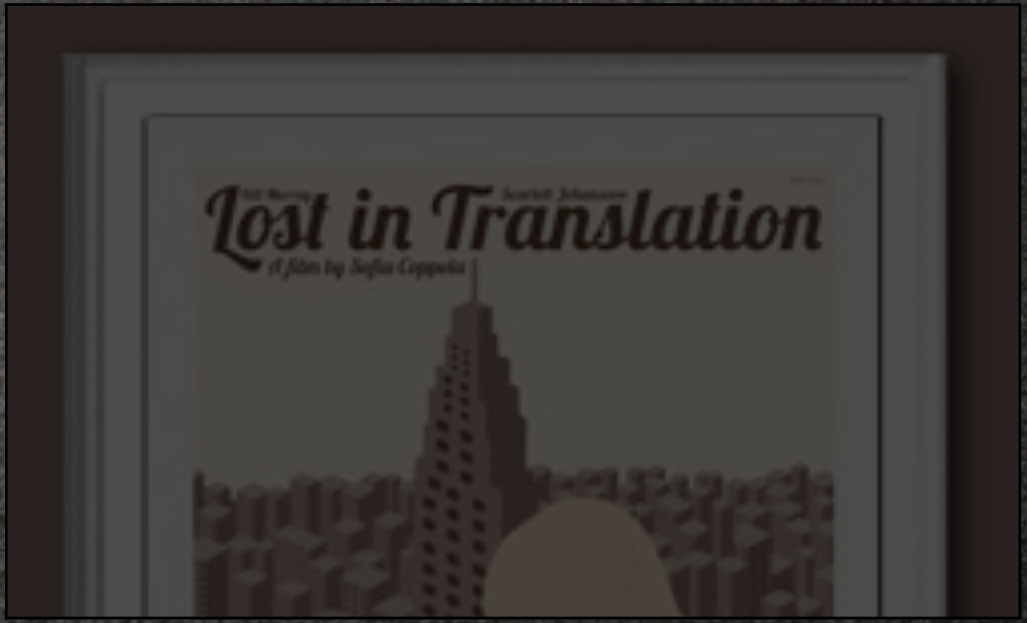
What works

1. Every base component can be as flexible as it defines itself to be.

2. Developers always have control on what they expose.

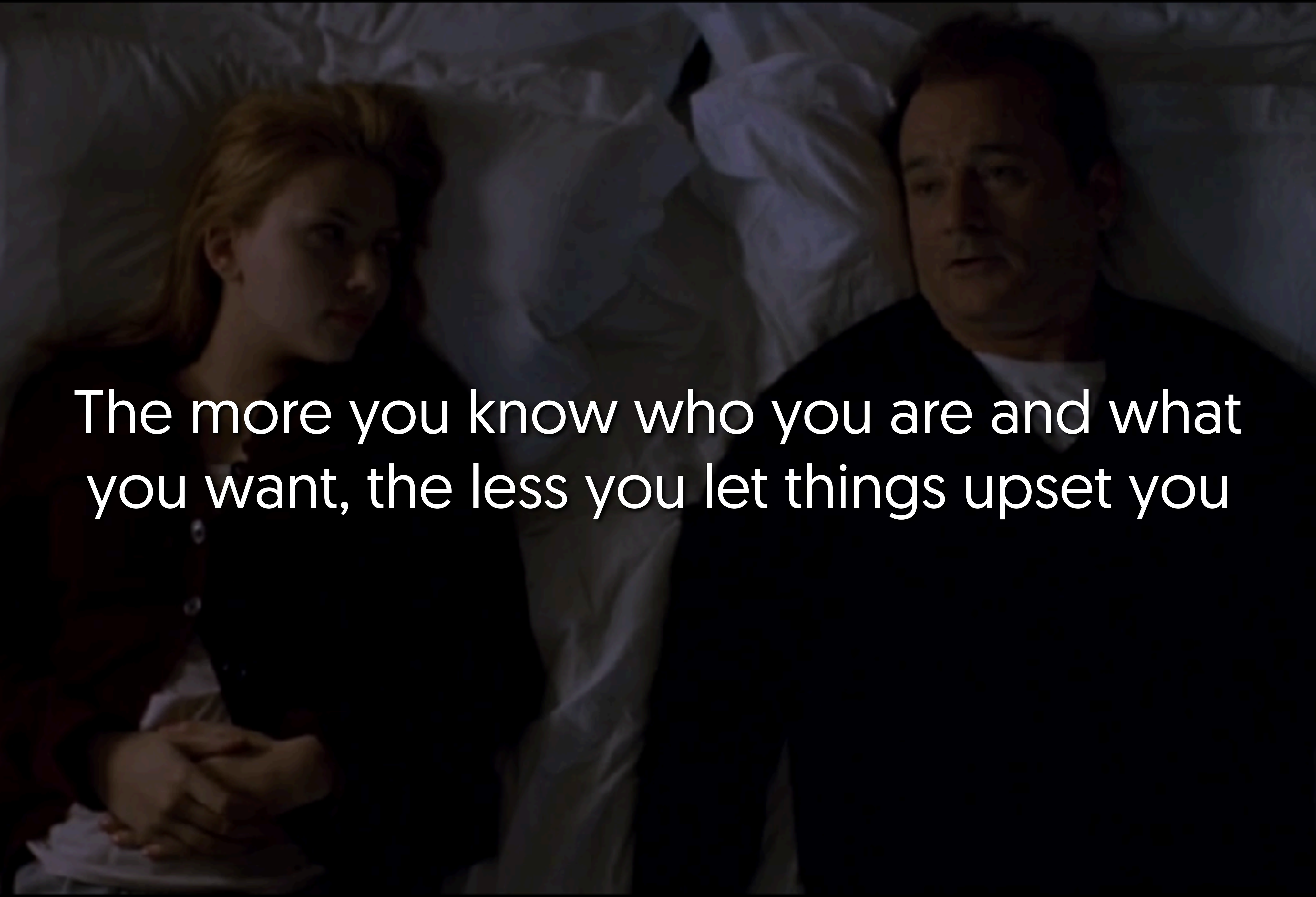
What really doesn't

1. This technique involves more complexity in thinking about the components
2. It's a slippery slope
3. How does an "open" component fit in the patterns?

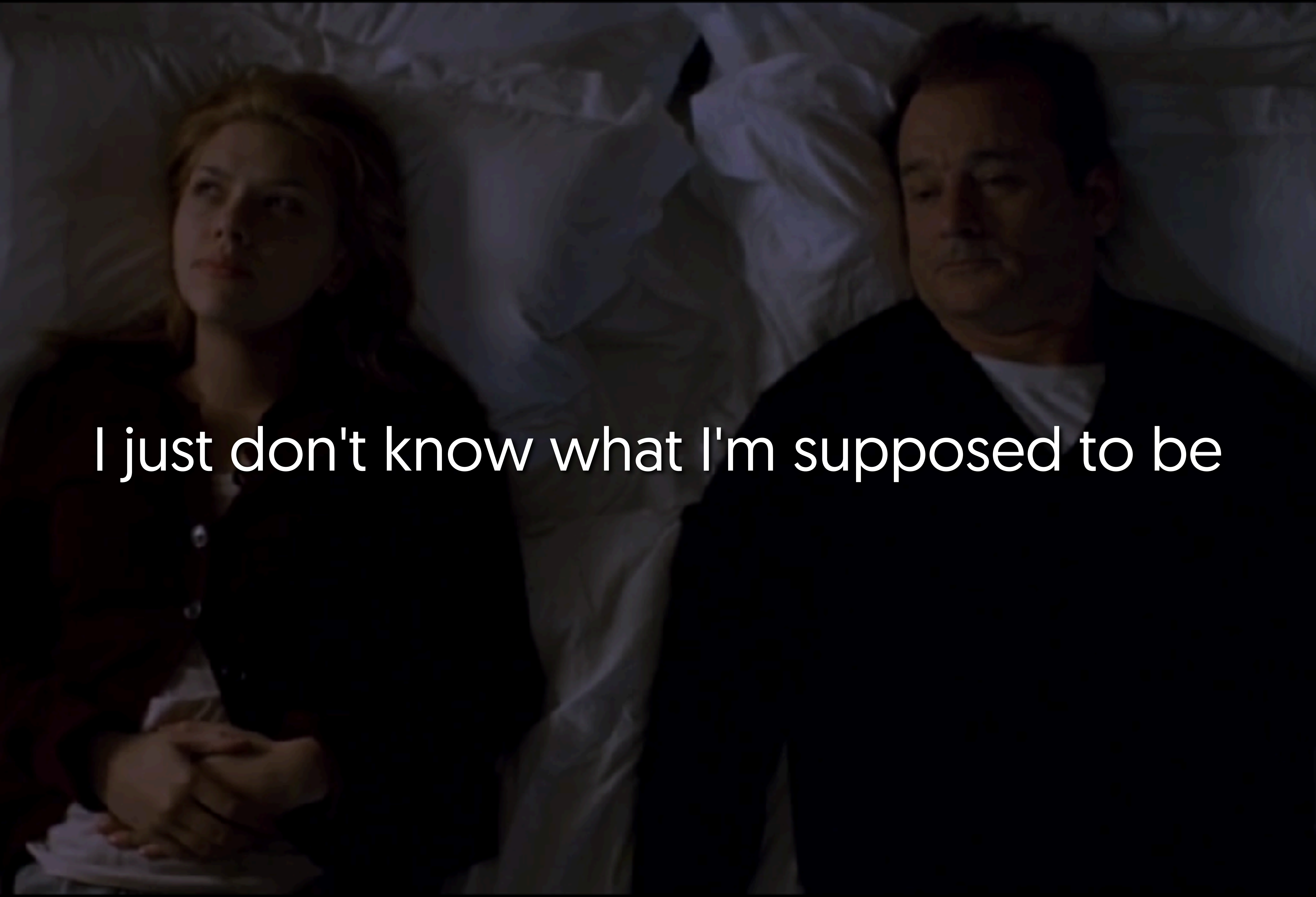


Does it get easier?

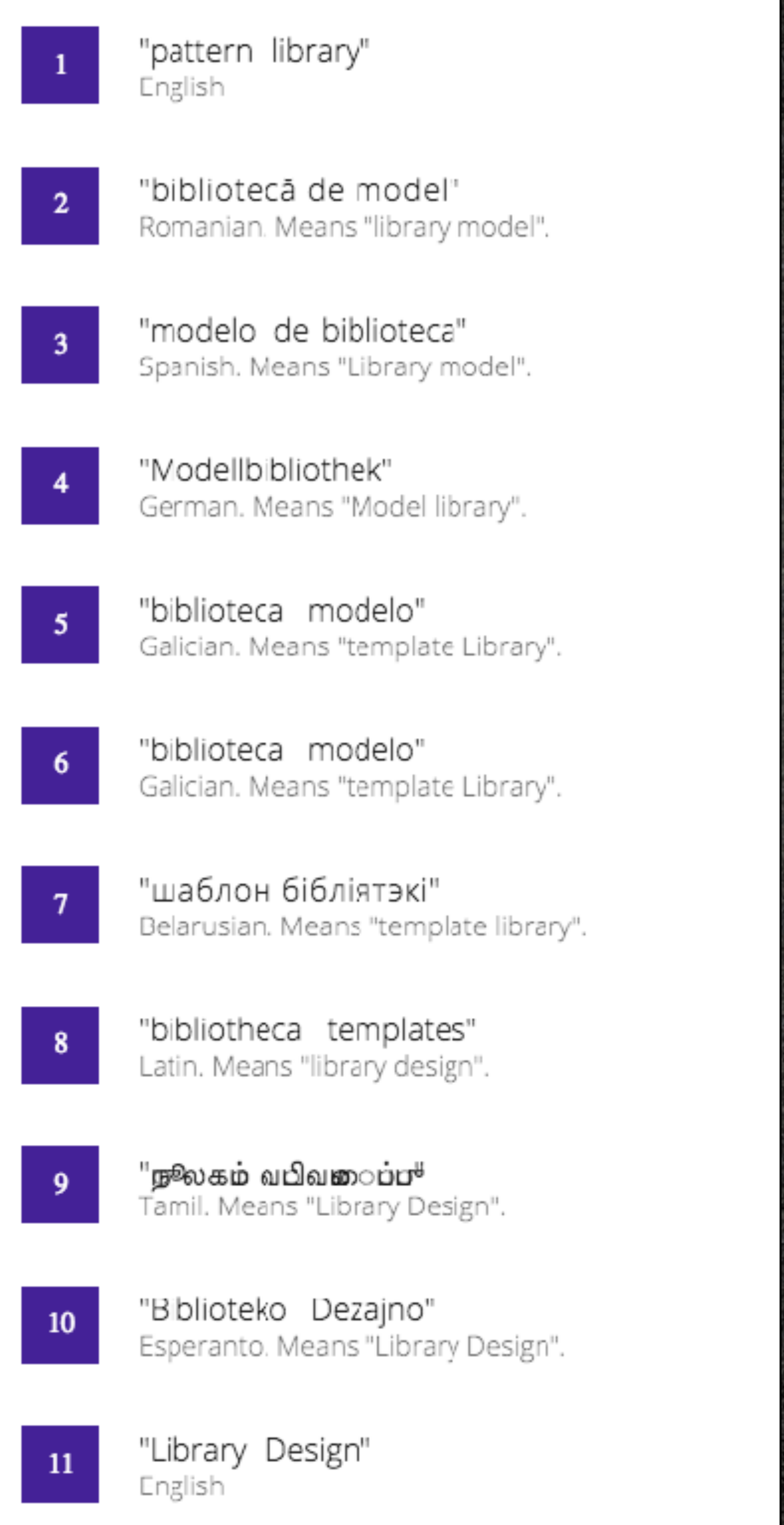
Basically



The more you know who you are and what you want, the less you let things upset you



I just don't know what I'm supposed to be



1	"pattern library"	English
2	"bibliotecă de model"	Romanian. Means "library model".
3	"modelo de biblioteca"	Spanish. Means "Library model".
4	"Modellbibliothek"	German. Means "Model library".
5	"biblioteca modelo"	Galician. Means "template Library".
6	"biblioteca modelo"	Galician. Means "template Library".
7	"шаблон бібліятэкі"	Belarusian. Means "template library".
8	"bibliotheca templates"	Latin. Means "library design".
9	"நூலகம் வடிவமைப்பு"	Tamil. Means "Library Design".
10	"Biblioteko Dezajno"	Esperanto. Means "Library Design".
11	"Library Design"	English

"A common language is a first step towards communication across cultural boundaries."

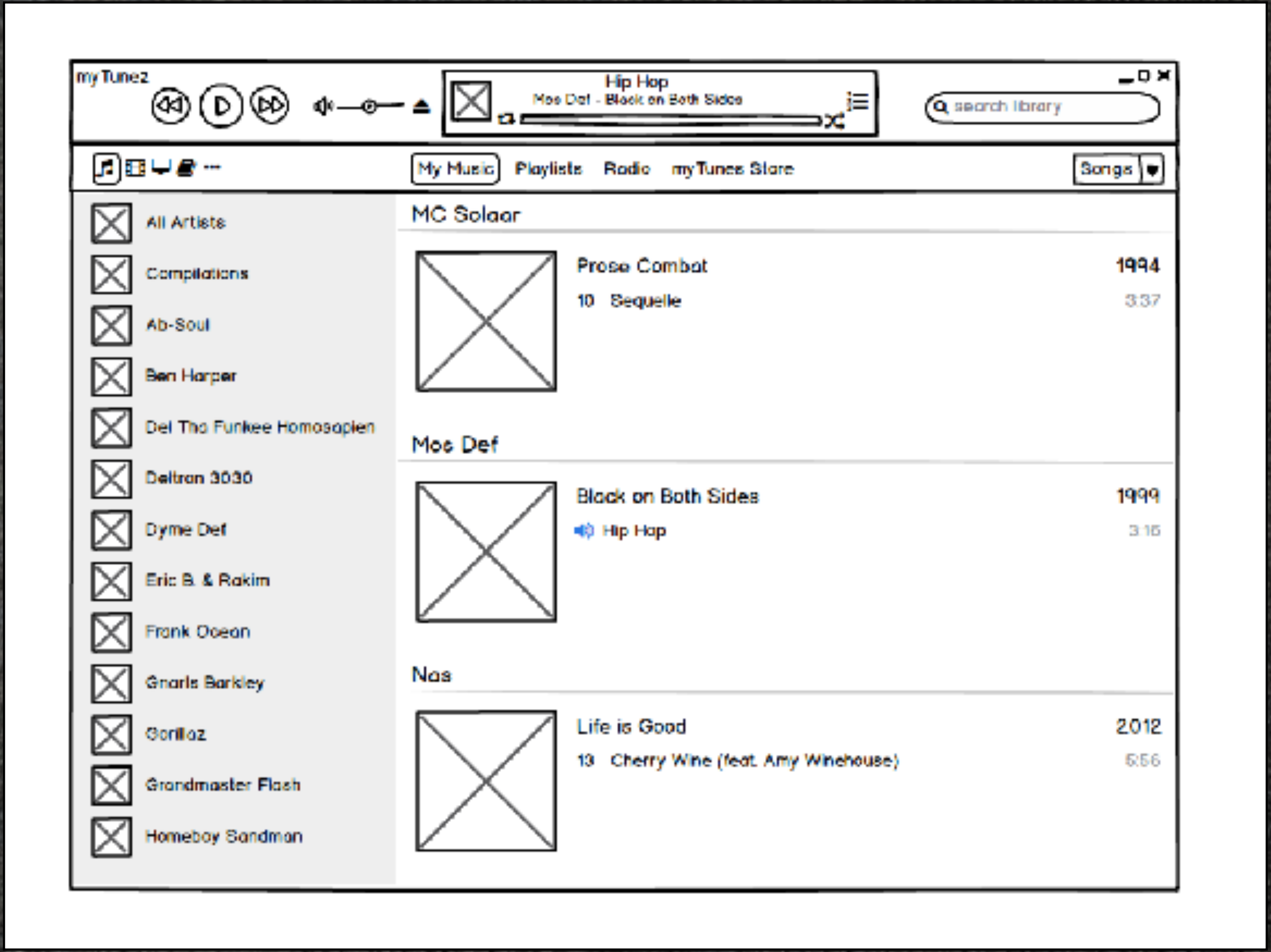
Ethan Zuckerman

The issue

How to understand - and convey - the meaning of an exception in our patterns?

Learn what the pattern your are building is supposed to be

Get involved early



All Artists
Compilations
Ab-Soul
Ben Harper
Del Tha Funkee Homosapien
Deltron 3030
Dyme Def
Eric B. & Rakim
Frank Ocean
Gnarls Barkley
Gorillaz
Grandmaster Flash
Homeboy Sandman

MC Solaar

Prose Combat 1994
10 Sequelle 3:37

Mos Def

Black on Both Sides 1999
Hip Hop 3:15

Nas

Life is Good 2012
13 Cherry Wine (feat. Amy Winehouse) 5:56

Talk to people

and remember that...

You are not hopeless

marco@fromthefront.it
http://cedmax.com
@cedmax

つづく