

Waterbed

Code

DISCLAIMERS

Light animations and transitions

No sounds

No flashing content



ANDREI PFEIFFER

Timișoara / RO



Code Designer



Organizer

rethink
revisit
revolve





5/6 Oct 2023


Timișoara/RO

revojs.ro










Some systems contain a minimum amount of complexity.

*Attempting to "**push down**" the complexity in one place will invariably cause it to "**pop up**" elsewhere.*

- Larry Wall, The Waterbed Theory

Law of Conservation of Complexity

TESLER'S LAW



Every application has an inherent amount of complexity that cannot be removed or hidden.

Instead, it must be dealt with, either in product development or in user interaction.

- Lawrence Gordon Tesler (mid-1980s)

UNNECESSARY

COMPLEXITY

UNNECESSARY **!=** **INHERENT**

COMPLEXITY

STATE

DEPENDENCIES

CONTROL FLOW STATEMENTS

USER INPUT

DEPENDENCIES

REQUIREMENTS

STATE

CONTROL FLOW STATEMENTS

USER INPUT


Useless
Box

SchubertBox.com

ON

OFF






Every application has an inherent amount of complexity that cannot be removed or hidden.

Instead, it must be dealt with, either in product development or in user interaction.

- Lawrence Gordon Tesler (mid-1980s)



Every application has an inherent amount of complexity that cannot be removed or hidden.

Instead, it must be dealt with, either in product development or in user interaction.

- Lawrence Gordon Tesler (mid-1980s)



Code

File

Edit

Selection

View

Go

Run

Terminal

Window

Help



Undo

⌘ Z

Redo

⇧ ⌘ Z

Cut

⌘ X

Copy

⌘ C

Paste

⌘ V

Find

⌘ F

Replace

⌥ ⌘ F

Find in Files

⇧ ⌘ F

Replace in Files

⇧ ⌘ H

Toggle Line Comment

⌘ /

Toggle Block Comment

⌥ ⇧ A

Emmet: Expand Abbreviation

→


Start Dictation...



Every application has an inherent amount of complexity that cannot be removed or hidden.

Instead, it must be dealt with, either in product development or in user interaction.

- Lawrence Gordon Tesler (mid-1980s)



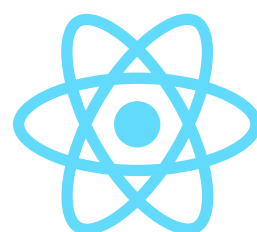
Every **(reusable) code** has an inherent amount of complexity that cannot be removed or hidden.

Instead, it must be dealt with, either in **the implementation** or by **its consumers**.

- Andrei Pfeiffer (May 2023, JSHeroes)

<Item />

Content



.....

CONSUMER / USAGE

Content

`<Item body="Content" />`


```
interface Item {  
  body: string;  
}
```

AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content

```
<Item body="Content" />
```

```
interface Item {  
  body: string;  
}
```

AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content

>

<**Item** body="Content" />

New design requirement

```
interface Item {  
  body: string;  
  arrow?: boolean;  
}
```

Add optional prop

AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content



<Item body="Content" arrow />

```
interface Item {
  body: string;
  icon?: "arrow" | "check";
}
```

		AUTHOR / IMPLEMENTATION
	
		CONSUMER / USAGE
Content	>	<Item body="Content" icon="arrow" />
Enabled	✓	<Item body="Enabled" icon="check" />

```
interface Item {
  body: string;
  icon?: "arrow" | "check";
  count?: number;
}
```

		AUTHOR / IMPLEMENTATION
	
		CONSUMER / USAGE
Content	>	<Item body="Content" icon="arrow" />
Enabled	✓	<Item body="Enabled" icon="check" />
Amount	5	<Item body="Amount" count={5} />


```
interface Item {  
  body: string;  
  icon?: "arrow" | "check";  
  count?: number;  
}
```

Complex implementation
Easy to use, but low control

AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content



```
<Item body="Content" icon="arrow" />
```

Enabled



```
<Item body="Enabled" icon="check" />
```

Amount

5

```
<Item body="Amount" count={5} />
```

```
interface Item {
  body: string;
  extra?: React.ReactNode;
}
```

Inversion of control

AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content	>
Enabled	✓
Amount	5

```
<Item body="Content" extra={<Icon name="arrow" style={} />} />
```

```
<Item body="Enabled" extra={<Icon name="check" style={} />} />
```

```
<Item body="Amount" extra={<Badge val={5} color={BLUE} />} />
```

// complex implementation

```
interface Item {  
  body: string;  
  icon?: "arrow" | "check";  
  count?: number;  
}
```

 *easy to use, low control*

 *great for reusability*

```
<Item body="Amount" count={5} />
```

// simple implementation

```
interface Item {  
  body: string;  
  extra?: React.ReactNode;  
}
```

 *high effort & control*

 *great for customisation*

```
<Item body="Amount" extra={  
  <Badge val={5} color={BLUE} />  
} />
```

// complex implementation

```
interface Item {  
  body: string;  
  icon?: "arrow" | "check";  
  count?: number;  
}
```

📈 *easy to use, low control*

👍 *great for reusability*

```
<Item body="Amount" count={5} />
```

Interface merge

// simple implementation

```
interface Item {  
  body: string;  
  extra?: React.ReactNode;  
}
```

📈 *high effort & control*

👍 *great for customisation*

```
<Item body="Amount" extra={  
  <Badge val={5} color={BLUE} />  
} />
```

```
interface Item {  
  body: string;  
  icon?: "arrow" | "check";  
  count?: number;  
  extra?: React.ReactNode;  
}
```

```
<Item body="Amount" count={5} />
```

```
<Item body="Amount" extra={  
  <Badge val={5} color={BLUE} />  
} />
```

```
interface Item {  
  body: string;  
  icon?: "arrow" | "check";  
  count?: number;  
  extra?: React.ReactNode;  
}
```

```
<Item body="Amount" count={5} />
```

```
<Item body="Amount" extra={  
  <Badge val={5} color={BLUE} />  
} />
```



Interface segregation

```
interface Item {  
  body: string;  
  extra?: React.ReactNode;  
}
```

```
interface IconItem {  
  body: string;  
  icon?: "arrow" | "check";  
}
```

```
interface CountItem {  
  body: string;  
  count?: number;  
}
```

```
<IconItem body="Content" icon="arrow" />
```

```
<CountItem body="Amount" count={5} />
```



```
interface Item {  
  body: string;  
  extra?: React.ReactNode;  
}
```

```
interface IconItem {  
  body: string;  
  icon?: "arrow" | "check";  
}
```

```
interface CountItem {  
  body: string;  
  count?: number;  
}
```

// segregated interfaces

👍 *ease of reusability*

👍 *customisation control*

👍 *easier to maintain*

```
interface Item {  
  body: string;  
  extra?: React.ReactNode;  
}
```

```
interface IconItem {  
  body: string;  
  icon?: "arrow" | "check";  
}
```

```
interface CountItem {  
  body: string;  
  count?: number;  
}
```

// segregated interfaces

👍 *ease of reusability*

👍 *customisation control*

👍 *easier to maintain*

But, has complexity
actually changed?

Content	>
Enabled	✓
Amount	5

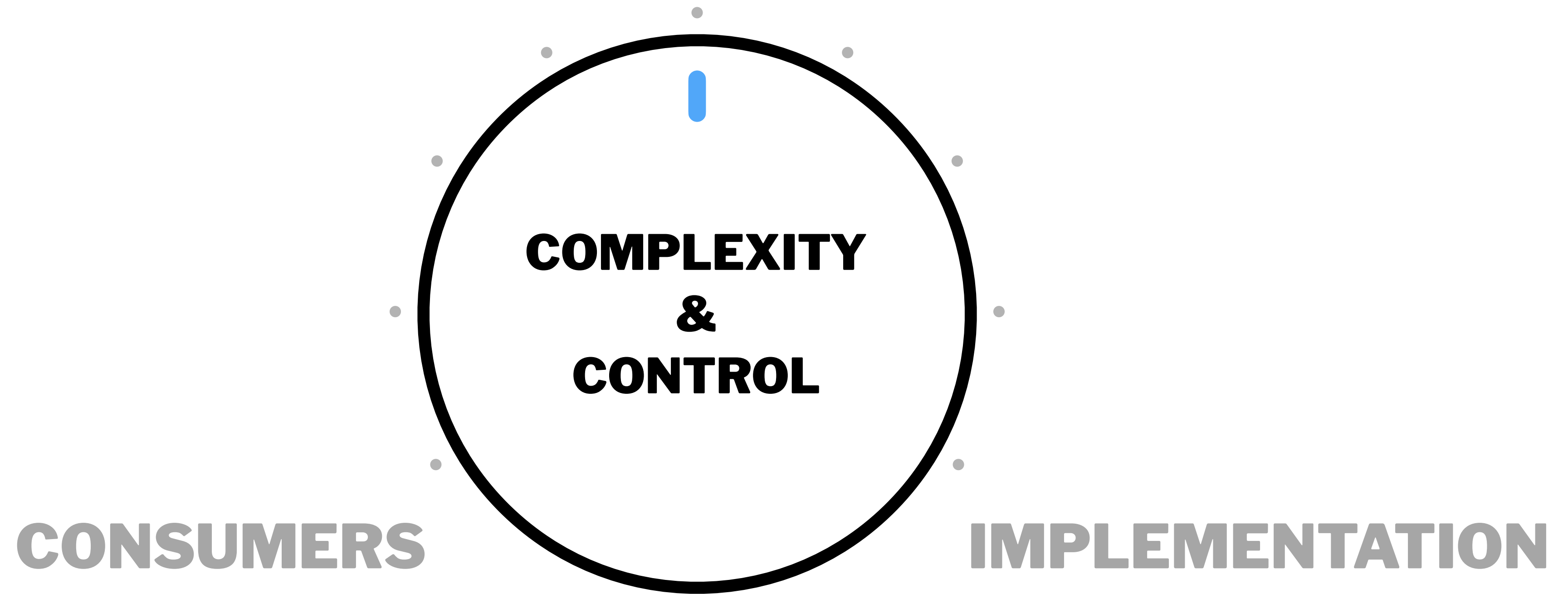
Content	>
Enabled	✓
Almost	ⓘ

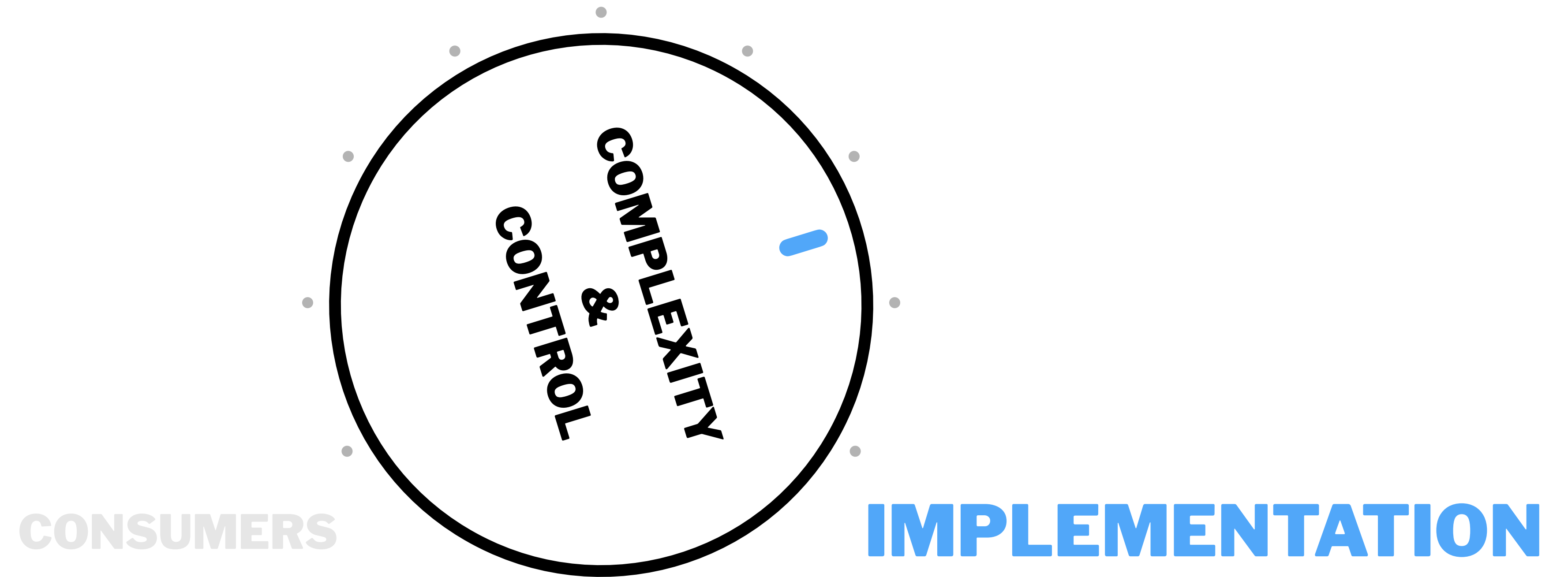
TO CONCLUDE

COMPLEXITY IS
INHERENT

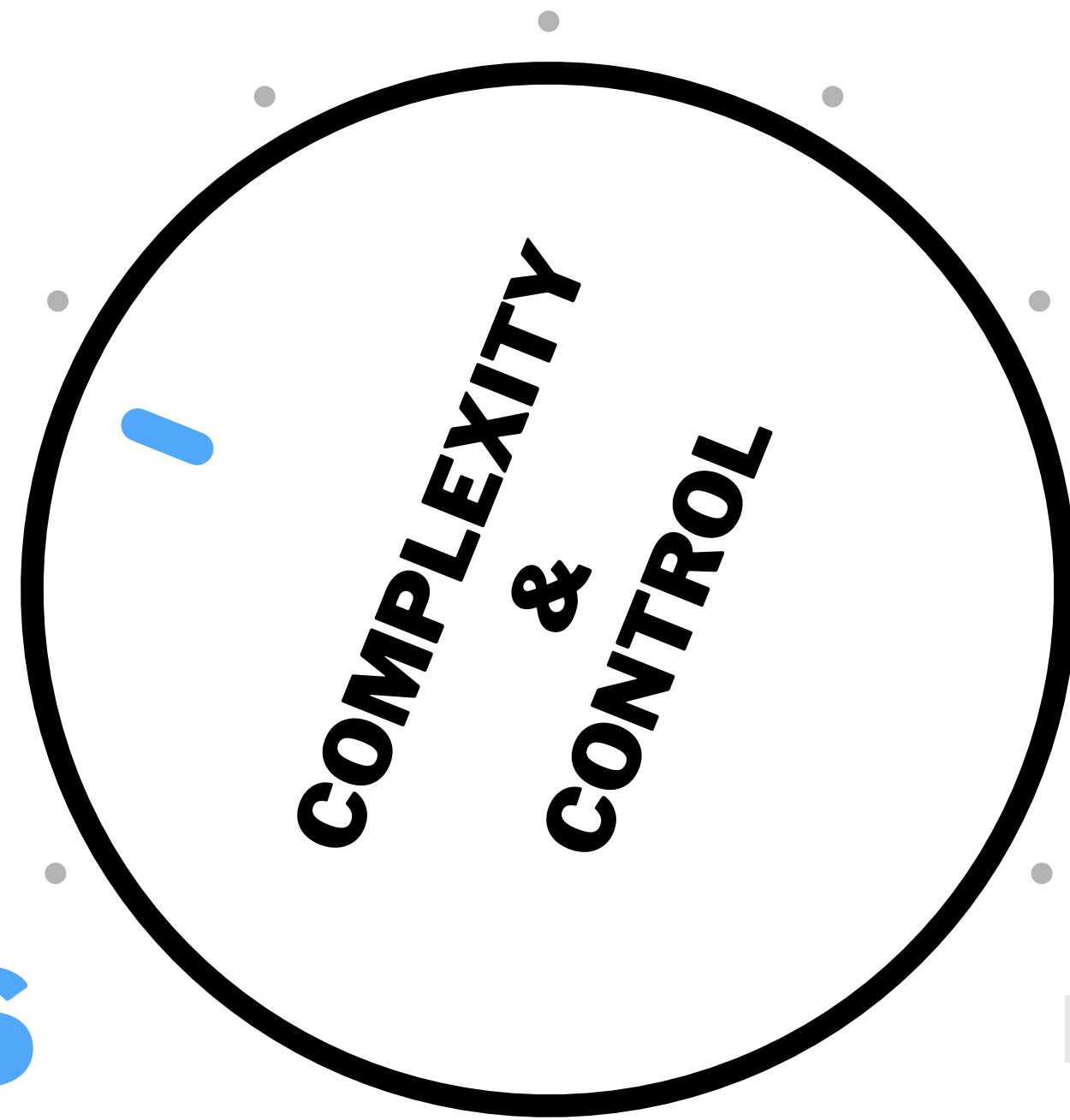
COMPLEXITY === CONTROL

WE ARE BOTH
CONSUMERS
AND AUTHORS





CONSUMERS



IMPLEMENTATION

THANK YOU



andreipfeiffer.dev