REACT ADVANCED
LONDON

July 3, 2025

Design Systems, AI, and the Art of **Separation of Concern**

John Adib

Figma - 3rd floor, 9 Devonshire Square, London EC2M 4YF

Hosted by

Figma

AI doesn’t care about your architecture

AI cares about your prompt.

You are the architect

AI is your assistant (for now 😅)

AI moves fast

You decide where it goes.

John Adib

MrAdib.com

A design system isn’t just a Figma file

It’s a shared language.

A design system is a contract

Between designers, developers, product managers — and now, AI.

Look. Feel. Behave.

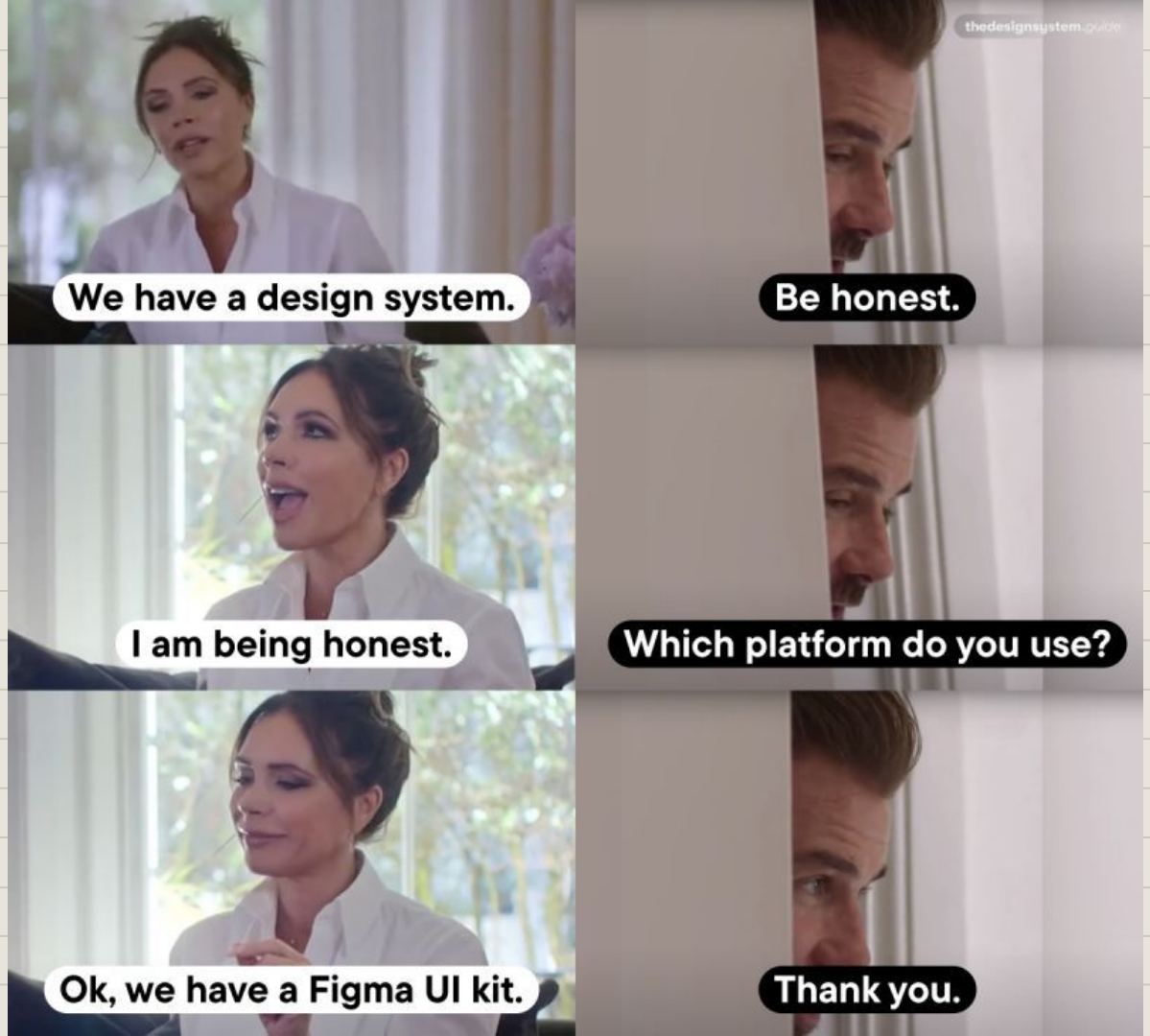
A real design system defines all three.

What about user flow and journey?

The design system gives us the bricks, not the blueprint.

So… what is a design system?

A set of reusable rules and components


We have a design system.
Be honest.
I am being honest.
Which platform do you use?
Ok, we have a Figma UI kit.
Thank you.

Look - Visual design

Colors

Typography

Spacing & layout

Icons

Component structure

Feel - Interaction

Hover, focus, press states

Motion/animation

Micro-interactions

Component feedback

Behave - Function

Disabled, loading, empty state, error states

Validation logic

State transitions

Accessibility

Component logic


DESIGN SYSTEM IS DONE!
FALSE. A DESIGN SYSTEM IS NEVER "DONE".

When the system is unclear...

Developers start making it up as they go. And honestly, they have to.

Challenge

Who owns the design system?

1. Designers build it.
2. Developers implement it.
3. Product teams rely on it.

Everyone is responsible.

Respect the system, or it breaks.


CREATING A DESIGN SYSTEM
001
HOW IT STARTED
HOW IT'S GOING

When the system breaks...

Inconsistency creeps in. Chaos follows.


THIS IS FINE.

Separation of Concern

Organize your system so each part handles a specific concern

Single Responsibility Principle

A file should have only one reason to change, one job to do.


WENGER

Tip #1

Separate Logic, UI, and Data

❌ All-in-One (Don't Do This)

```
1  export function LogViewer() {
2    // 📦 Fetch logs
3    const [logs, setLogs] = useState([]);
4    useEffect(() => {
5      fetch('/api/logs')
6        .then(res => res.json())
7        .then(setLogs);
8    }, []);
9    // 🧠 Format & highlight
10   const formatTime = (ts: string) => new Date(ts).toLocaleTimeString();
11   const getColor = (level: string) =>
12     level === 'error' ? 'red' : level === 'warn' ? 'orange' : 'black';
13   // 🎨 UI
14   return (
15     <ul>
16       {logs.map(log => (
17         <li style={{ color: getColor(log.level) }}>
18           [{formatTime(log.timestamp)}] {log.message}
19         </li>
20       ))}
21     </ul>
22   );
23 }
```

❌ All-in-One (Don't Do This)

```
1  export function LogViewer() {
2    // 📦 Fetch logs
3    const [logs, setLogs] = useState([]);
4    useEffect(() => {
5      fetch('/api/logs')
6        .then(res => res.json())
7        .then(setLogs);
8    }, []);
9    // 🧠 Format & highlight
10   const formatTime = (ts: string) => new Date(ts).toLocaleTime(log.
11   const getColor = (level: string) =>
12     level === 'error' ? 'red' : level === 'warn' ? 'orange' : 'black';
13   // 🎨 UI
14   return (
15     <ul>
16       {logs.map(log => (
17         <li style={{ color: getColor(log.level) }}>
18           [{formatTime(log.timestamp)}] {log.message}
19         </li>
20       ))}
21     </ul>
22   );
23 }
```

✅ Separation of Concern

```
└── logs/
    ├── LogViewer.tsx        # Composition
    ├── LogList.tsx          # UI
    ├── useLogs.ts           # Data
    ├── formatTime.ts        # Logic
    ├── getColor.ts          # Logic
    ├── types.ts             # Types
```

```typescript
// types.ts

export type LogLevel = 'info' | 'warn' | 'error';

export interface ILogItem {
  id: string;
  message: string;
  timestamp: string;
  level: LogLevel;
}

```

```ts
// useLogs.ts

import { useEffect, useState } from 'react';
import { ILogItem } from './types';

interface IUseLogs {
  logs: ILogItem[];
  isLoading: boolean;
  logsError: Error | null;
}

// Handles fetching log data, error handling, and loading state
export function useLogs(): IUseLogs {
  const [logs, setLogs] = useState<ILogItem[]>([]);
  const [isLoading, setIsLoading] = useState(true);
  const [logsError, setLogsError] = useState<Error | null>(null);

  useEffect(() => {
    const fetchLogs = async () => {
      try {
        const res = await fetch('/api/logs');
        const data = await res.json();
        setLogs(data);
      } catch (err: unknown) {
        setLogsError(err);
      } finally {
        setIsLoading(false);
      }
    };
    fetchLogs();
  }, []);
  return { logs, isLoading, logsError };
}
```

```ts
// formatTime.ts

import { format, isValid, parseISO } from 'date-fns';

// Safely formats a timestamp. Returns fallback if invalid.
export function formatTime(ts: string): string {
  const date = parseISO(ts);
  return isValid(date) ? format(date, 'HH:mm:ss') : 'Invalid date';
}
```

```ts
// getColor.ts

import { LogLevel } from './types';

// Returns a color code based on the log level
export function getColor(level: LogLevel): string {
  switch (level) {
    case 'error': return 'red';
    case 'warn': return 'orange';
    default: return 'black';
  }
}
```

```tsx
// LogList.tsx

import { formatTime } from './formatTime';
import { getColor } from './getColor';
import { ILogItem } from './types';

interface Props {
  logs: ILogItem[];
}

// Pure UI component: displays a list of logs
export function LogList({ logs }: Props) {
  return (
    <ul>
      {logs.map(log => (
        <li key={log.id} style={{ color: getColor(log.level) }}>
          [{formatTime(log.timestamp)}] {log.message}
        </li>
      ))}
    </ul>
  );
}

```

```tsx
// LogSkeleton.tsx

// Simple skeleton UI for loading state
export function LogSkeleton() {
  return (
    <ul>
      {[1, 2, 3].map(i => (
        <li key={i} style={{ opacity: 0.5 }}>
          [--:--:--] Loading log message...
        </li>
      ))}
    </ul>
  );
}
```

```tsx
// LogEmpty.tsx

import { Player } from '@lottiefiles/react-lottie-player';
import animationData from './animations/empty.json';

export function LogEmpty() {
  return (
    <div>
      <Player autoplay loop src={animationData}/>
      <p>No logs available</p>
    </div>
  );
}
```

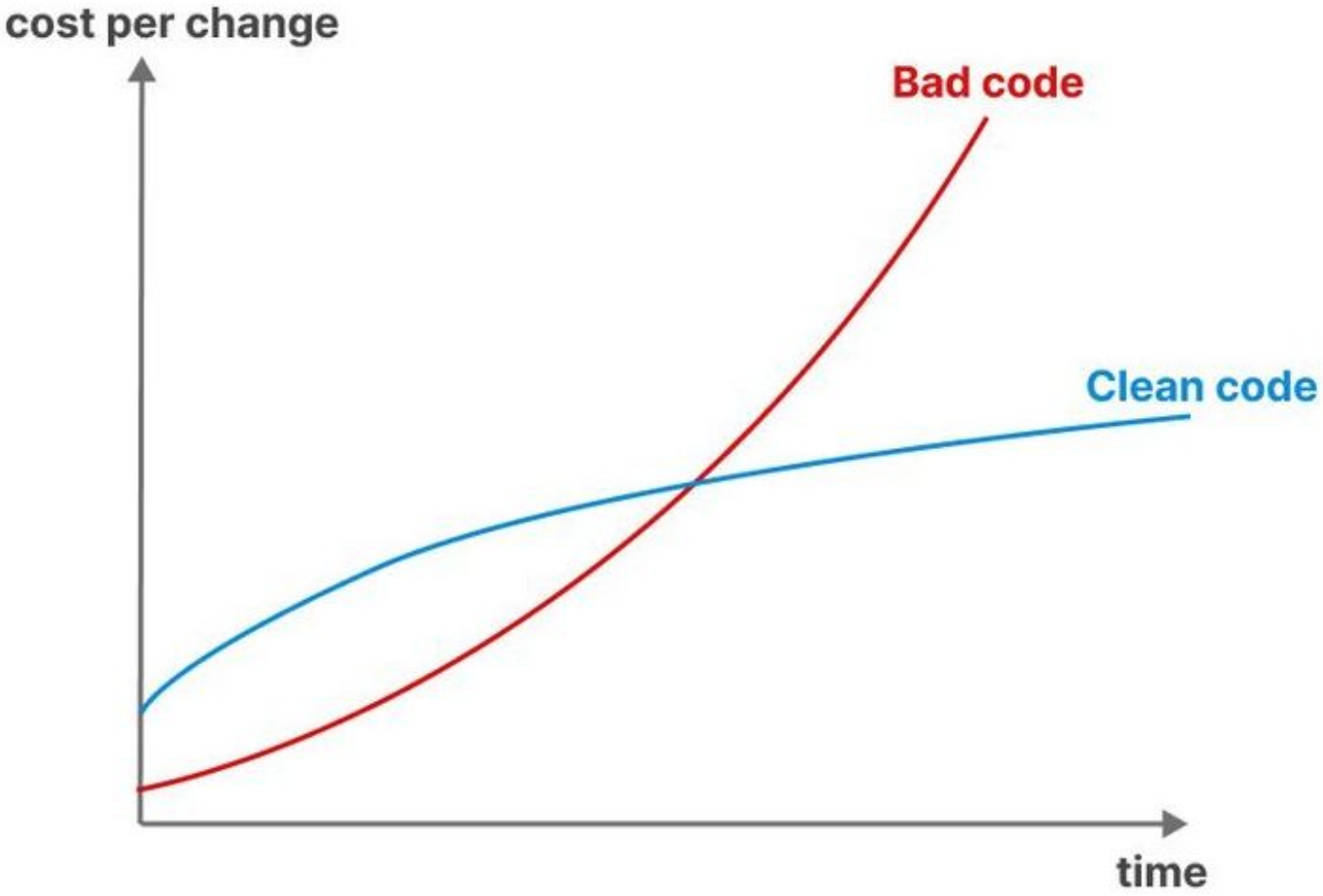
```tsx
// LogViewer.tsx

import { useLogs } from './useLogs';
import { LogList } from './LogList';
import { LogSkeleton } from './LogSkeleton';
import { LogError } from './LogError';
import { LogEmpty } from './LogEmpty';

export default function LogViewer() {
  const { logs, isLoading, logsError } = useLogs();

  if (isLoading) return <LogSkeleton />;
  if (logsError) return <LogError error={logsError} />;
  if (logs.length === 0) return <LogEmpty />;

  return <LogList logs={logs} />;
}
```


cost per change
Bad code
Clean code
time

Tip #2

Encapsulate Feature Logic in a Folder Module

Use `index.tsx` to make your feature folder the module

✅ Treat Folders as Encapsulated Modules

```
// Before:
import LogViewer from '@/components/logs/LogViewer';

// After:
import LogViewer from '@/components/logs';

```

Tip #3

Use Design Tokens to Centralize UI Decisions

Tip #4

Make Your Design System the Source of Truth

Tip #5

Let Your Linter Enforce Boundaries.

Tip #6

Use Descriptive, Consistent Naming Across Design and Code

Tip #7

Teach Your AI Tools Your Architecture

7 Tips to Rule Them All

1. Separate Logic, UI, and Data
2. Encapsulate Feature Logic in a Folder Module
3. Use Design Tokens to Centralize UI Decisions
4. Make Your Design System the Source of Truth
5. Let Your Linter Enforce Boundaries.
6. Use Descriptive, Consistent Naming Across Design and Code
7. Teach Your AI Tools Your Architecture

AI doesn’t care about your architecture

AI cares about your prompt.

THANK YOU!

SCAN ME

MrAdib.com