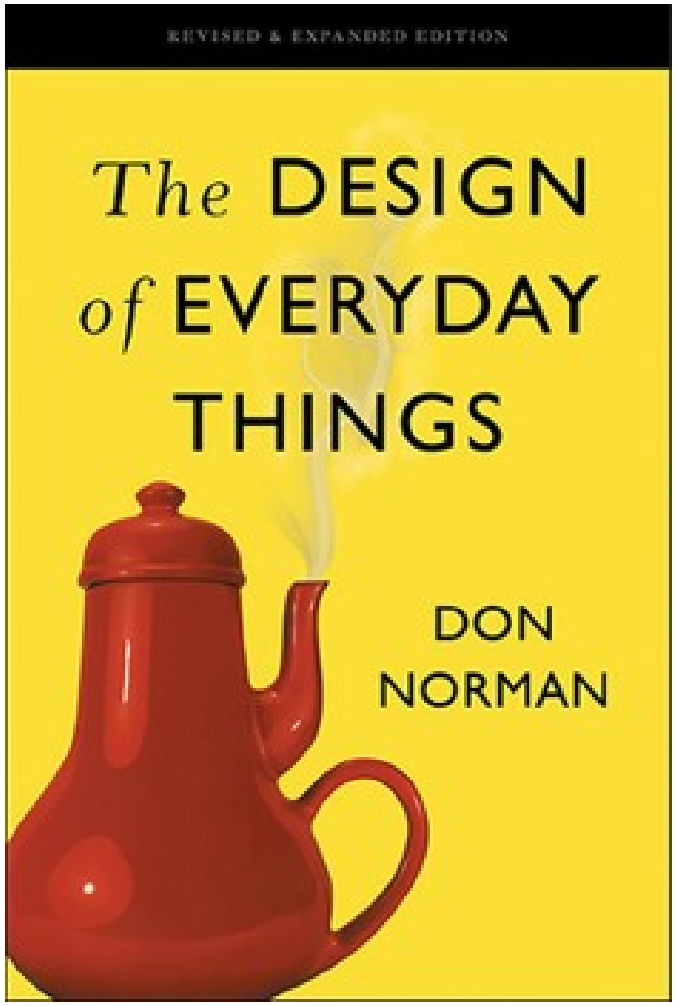
SWE 4743: Object-Oriented Design

Jeff Adkisson

Monday, Wednesday at 6:30 PM
Architecture, 175

“Good design is actually a lot harder to notice than poor design, in part because good designs fit our needs so well that the design is invisible.

Don Norman
The Design of Everyday Things




PULL
No dogs
except guide dogs
PULL

Code makes things work. Design makes things *last*.

Two identical features implemented:

- Version A: shipped fast, hard to change
- Version B: shipped thoughtfully, adapts for years

Most software cost is **not writing code**—it's *changing* it.

Design choices compound over time, for better or worse.

Good design lets *average developers* produce *excellent systems*.

Why do so many successful systems still get rewritten?

```
Month 1 - Works! We shipped so fast!
Vibe coding ROCKS!

Month 6 - Still okay, slightly
difficult to maintain…

Year 2 - Pain, every ”enhancement”
breaks something, so buggy…

Year 5 ——— Rewrite discussion, so
brittle nothing can be added, ugh…
```

Design debt doesn’t hurt immediately—*that’s why it’s dangerous*.

- Early success often masks structural problems.
- OO design exists to manage **long-lived complexity**, not toy programs.

Design Is About Change, Not Features

Initial requirements don’t kill systems - **changing requirements do**.

Object oriented design techniques exist to *localize change*.

Single responsibility, open-closed principle, dependency inversion all answer the same question:

“Where should this change go?”

Your most important reader is *future developers*, not the compiler.

- Code is **read far more** than it is written.
- Good design communicates intent, boundaries, and invariants.
- Classes and interfaces are *sentences*, not just containers.

Design is how you talk to
people you'll never meet.

--- or if you stick around long enough ---

Design is how *future you*
can avoid hating *past you*.

Good design *limits* what you can do… on purpose!

- Encapsulation is about *preventing misuse*.
- Constraints reduce cognitive load.

Well-designed systems make **wrong code hard to write**.

The Tragedy of “Just One More If”

•Most design failures start small and seem reasonable.

•**Today**:

- if (type == A) { ... }
- else if (type == B) { ... }
- else if (type == C) { ... }

•**Soon**:

- *18 more conditions, cases, tweaks, requirements, etc.*

- Design erosion happens incrementally.
- OO design provides *escape hatches* before conditionals explode.
- Patterns often emerge *after* pain—this course helps you see them earlier.

Design distinguishes *programmers* from *software engineers*.

- Industry rewards people who can manage complexity.
- Design skill is what scales your impact.
- Languages and frameworks change—design principles do not.
- Compare:
  “Can you implement this?”
  “Can you evolve this safely?”

•Job title progression
- Junior →
- Senior →
- Architect

Abstraction Is Choosing What Not to Care About

Without Abstraction

Every detail is visible everywhere

↓

More things to think about

↓

Harder to change safely

With Abstraction

Important ideas are *visible and* unimportant details are *hidden*

↓

Fewer things to think about

↓

Changes stay localized

ABSTRACTION

Korean War Veterans Memorial, 1995
Frank Gaylord
Washington, DC

ABSTRACTION

The Walking Man, 1960
Alberto Giacometti
Museum of Modern Art (MoMA), New York

ABSTRACTION

Unique Forms of Continuity in Space (1913)
Umberto Boccioni
Museum of Modern Art (MoMA), New York

Abstraction Is Choosing What Not to Care About

Without Abstraction

Every detail is visible everywhere

↓

More things to think about

↓

Harder to change safely

With Abstraction

Important ideas are *visible*

Unimportant details are *hidden*

↓

Fewer things to think about

↓

Changes stay localized

What you *do not* see is the point

- Every abstraction hides or eliminates irrelevant information.
- In other words, good design discards the *right* details.
- Bad design discards the wrong ones—or none at all.

Abstraction is selective blindness.

Abstraction Comparison: The Linux File System API

The Actual Linux API…

- `int fd = open("data.txt", O_RDONLY);`
- `read(fd, buffer, 4096);`
- `write(fd, buffer, 4096);`
- `close(fd);`

Hypothetical Bad Abstraction

- `int disk = openDisk(0);`
- `int block = allocateBlock(disk);`
- `writeBlock(disk, block, buffer);`
- `flushCache(disk);`

What it abstracts

- Persistent storage
- Sequential & random access
- Durability
- Naming (paths)

What it hides

- Disk blocks
- SSD vs HDD
- Caching
- Journaling
- RAID
- Network storage
- Failure recovery

What it exposes

- Block sizes
- Disk layout
- Caching policy
- Storage hardware
- Write ordering

What it forces

- Every program to manage storage details
- Widespread changes when hardware changes
- High cognitive load
- Fragile code

Abstraction Comparison: The Linux File System API

The Actual Linux API…

- `int fd = open("data.txt", O_RDONLY);`
- `read(fd, buffer, 4096);`
- `write(fd, buffer, 4096);`
- `close(fd);`

What it abstracts

- Persistent storage
- Sequential & random access
- Durability
- Naming (paths)

What it hides

- Disk blocks
- SSD vs HDD
- Caching
- Journaling
- RAID
- Network storage
- Failure recovery

Good abstractions let you reason about *what* you want to accomplish… not *how* it is achieved.

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

Martin Fowler