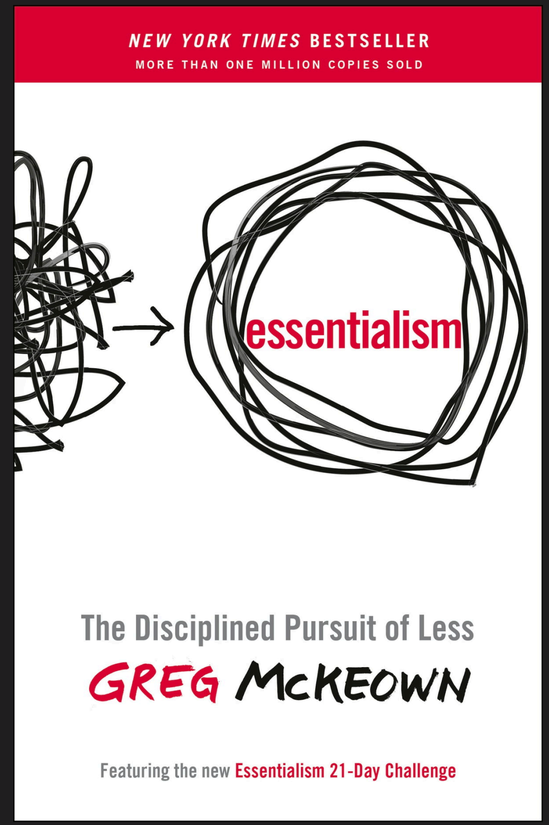
The
SOFTWARE
ESSENTIALIST

The 4 Pillars of Scalable Software: Architecture, Design, Testing & Strategy

essentialist.dev

Essentialism Produces10,000x Developers

> “According to the power law theory, certain efforts actually produce exponentially more results than others. For example, as Nathan Myhrvold, the former chief technology officer for Microsoft, has said (and then confirmed to me in person), “The top software developers are more productive than average software developers not by a factor of 10X or 100X or even 1,000X but by 10,000X.” It may be an exaggeration, but it still makes the point that certain efforts produce exponentially better results than others.”
>
> — Greg McKeown



We are much closer to mastering what actually matters

✅ *Problem Decomposition is the Main Skill*

✅ *Systems Thinking is for Problem Decomposition*

✅ *The Purpose of Software: Help 3 Roles Achieve Their Goals*

✅ *Your Responsibility = Key Developer Use Cases (ie: Feature Work; add, change, remove, discover, test, understand)*

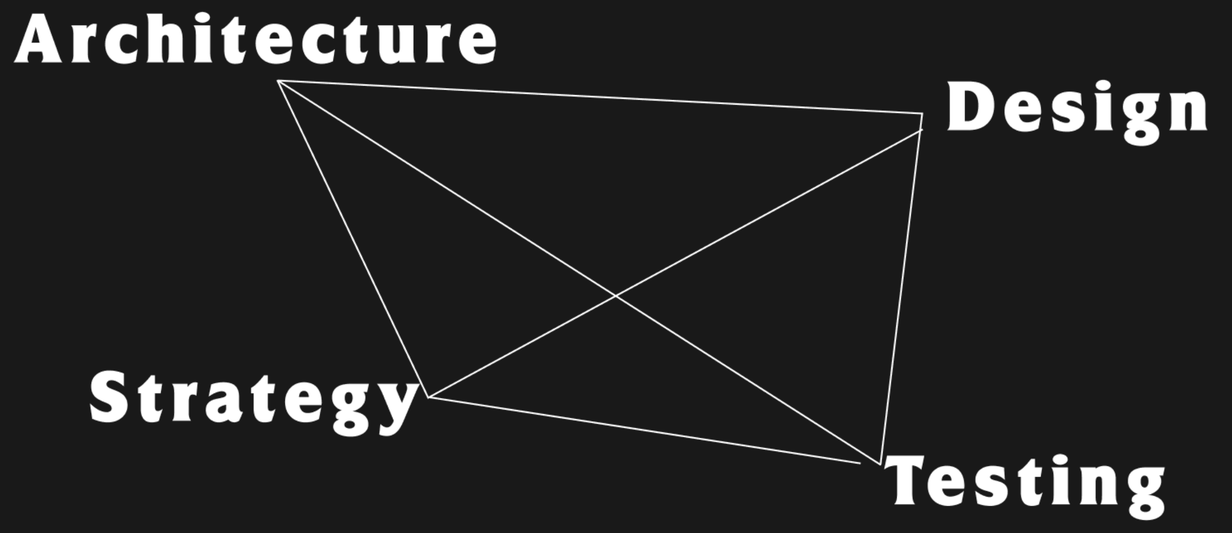
✅ *Scalability = Keeping 3 Roles In Benefit*

... How do we achieve scalability?

What we'll cover

- *The 4 pillars of scalable software and why it's hard to learn them*
- *What we're going to do instead in order to learn these*
- *The 80-20 (the most important parts of each)*

The 4 Pillars



These are the main buckets for everything

Clean Code → *Design (mostly)*

Stubbing/mocking → *Testing (mostly)*

Hexagonal Arch → *Arch (mostly)*

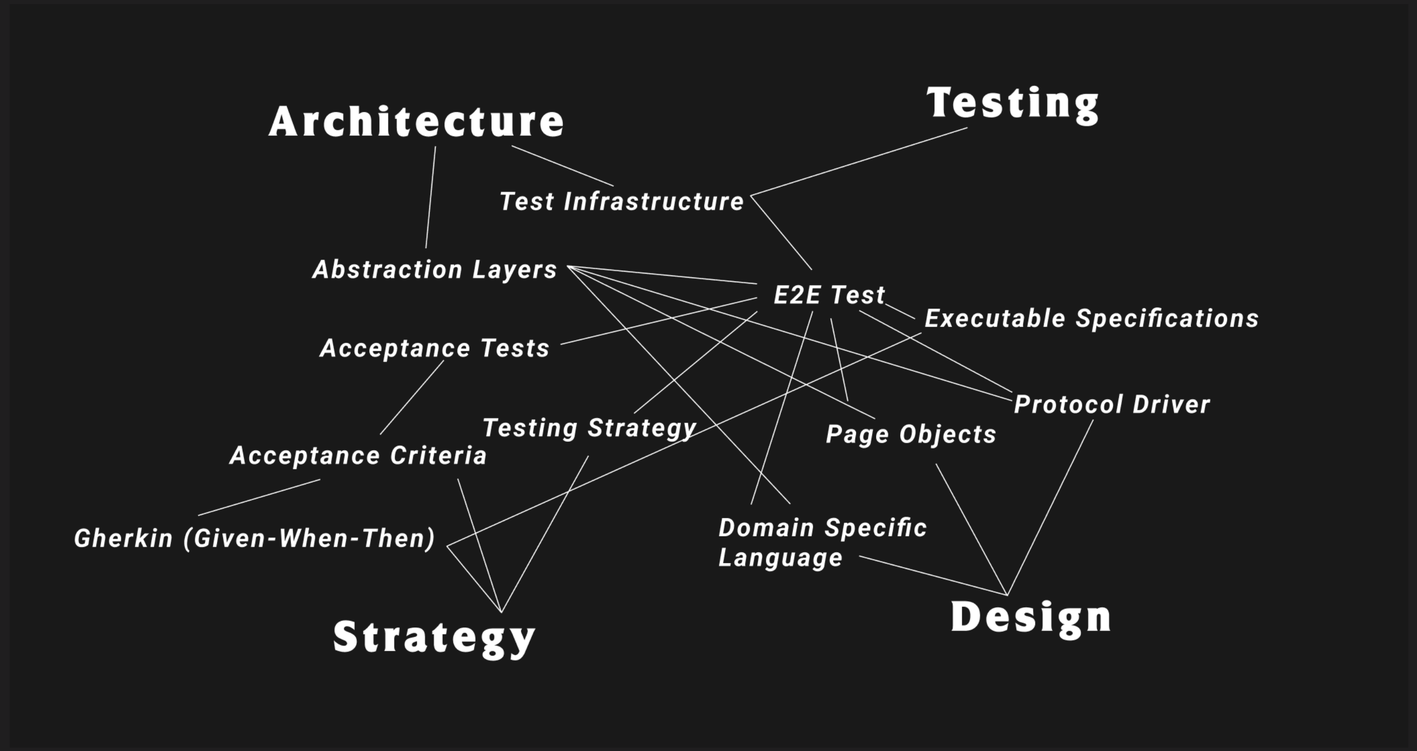
Impact Mapping → *Strategy (mostly)*

The problem?

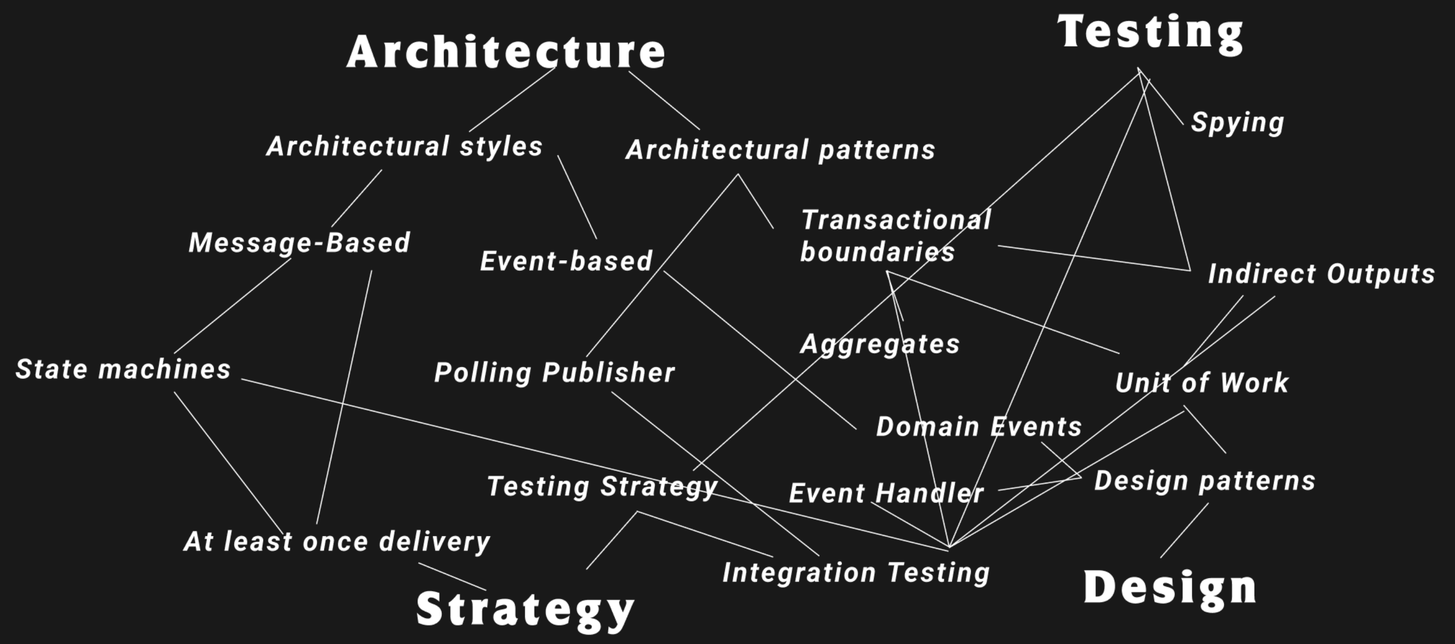
They're all inextricably linked together, which makes it extremely hard to learn "just testing" or "just design" or "just architecture".

You need to learn pieces of each at the same time.

Example: Proper E2E Testing

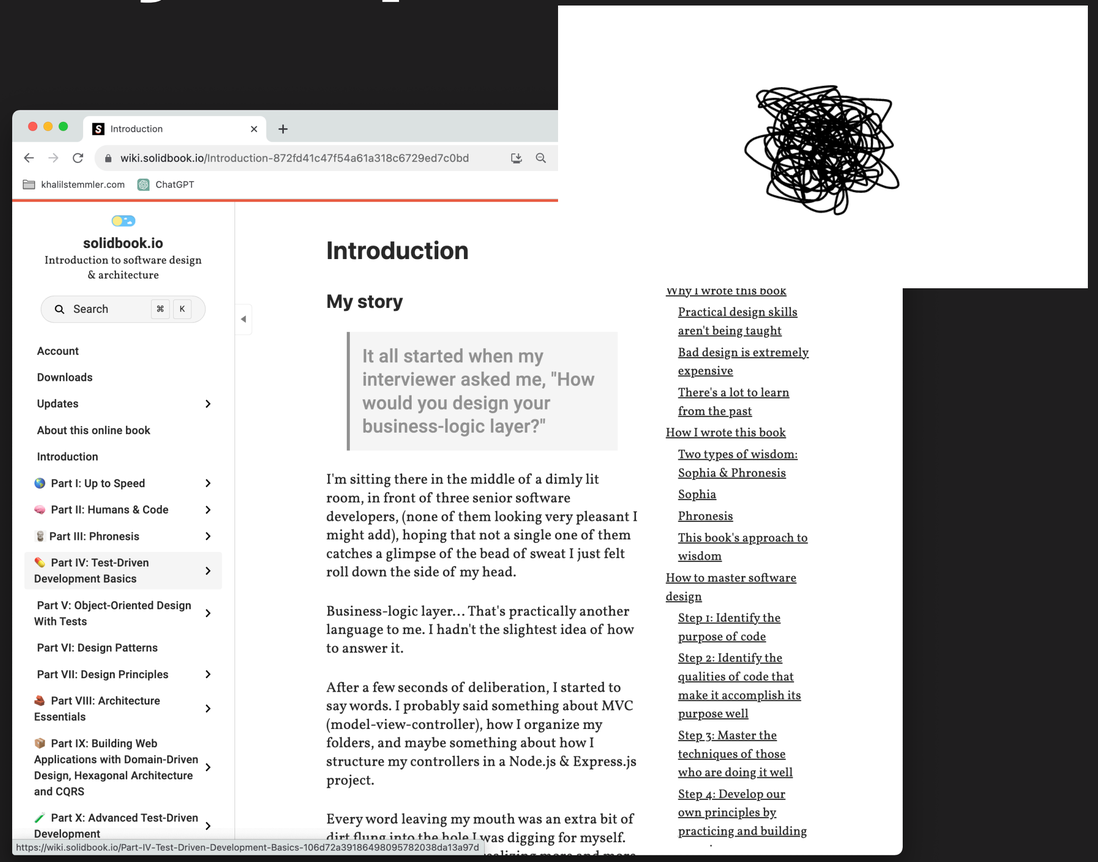


Example: Decoupling Operations using an Event-Based Architecture



Complexity explosion

Gregor Hohpe's bookshelf

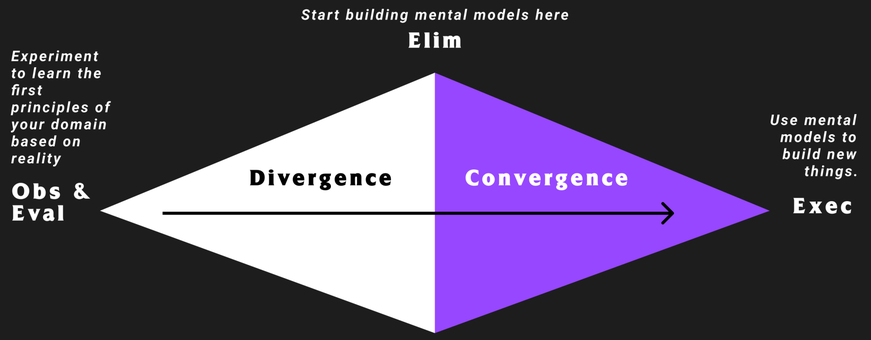


Solidbook wiki - started out as a handbook, turned into a wiki

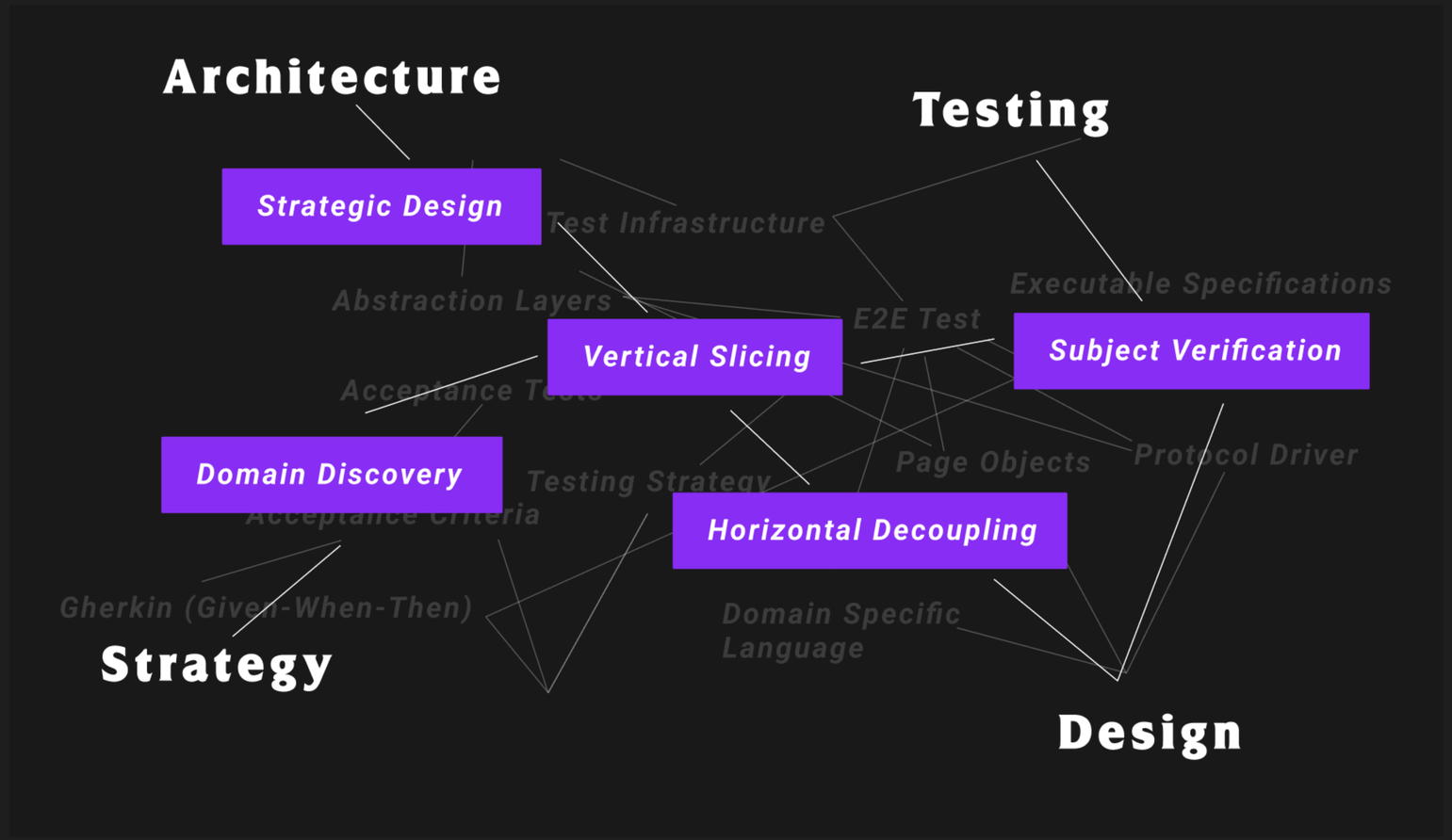
Common scene; another stack of books on my table to digest

essentialist.dev

Mastery takes time, it's an untangling, clarification & installation process

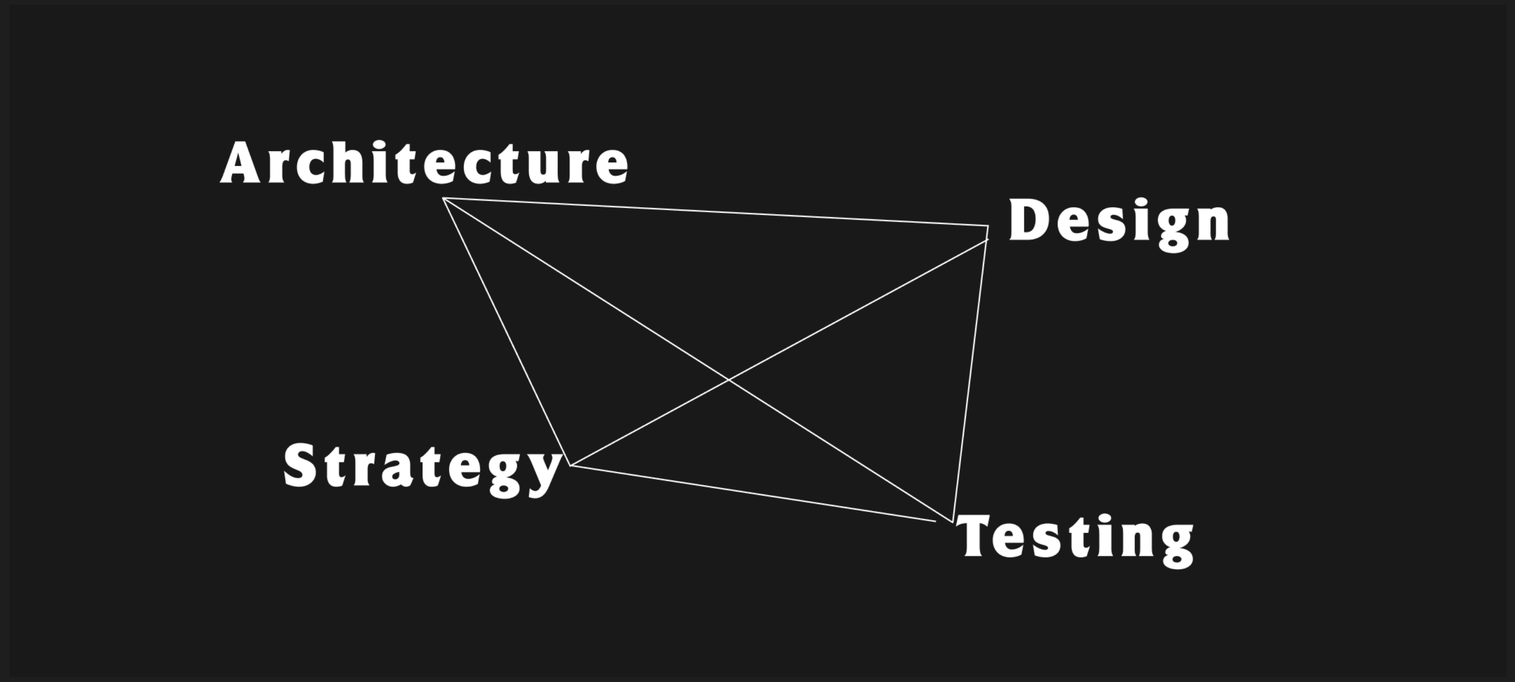


How we'll detangle: 12 Principles



We'll continually expand & pick up more of these

But first, let’s look at the most important parts of each



Architecture

“Will the structure support the long term goals of the 3 roles?”

- *Architectural Styles*
- *Architectural Patterns*
- *Architectural Components*
- *Frameworks & libraries*
- *Boundaries (customer & developer)*
- *Folder/project structure*

Design 🎨

“Can other devs easily discover & understand how to perform their key developer use cases against the codebase?”

Design

Discoverability & understanding

Design patterns

Naming

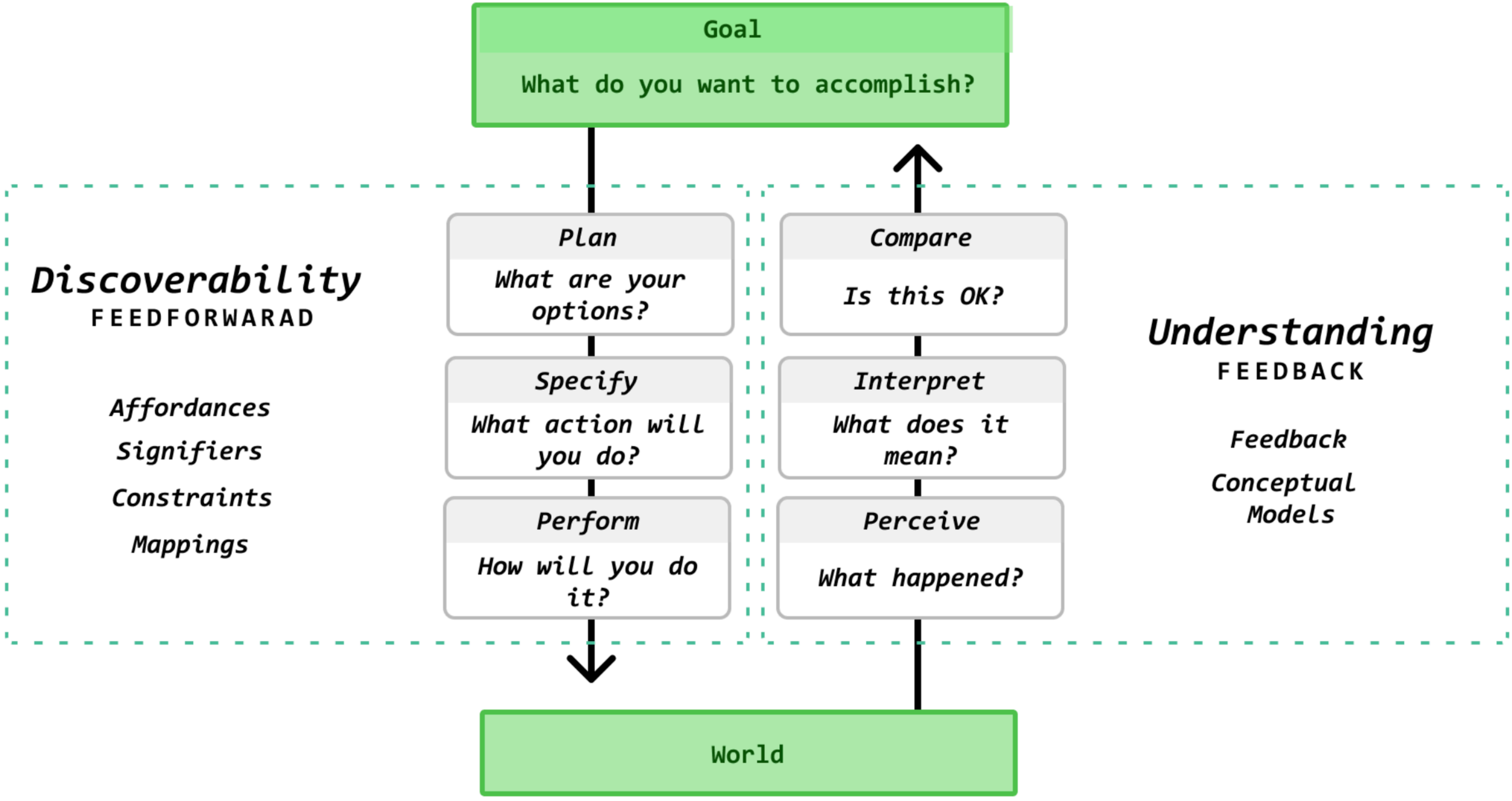
Consistency

Constraints

Style

Mental models

> "Two of the most important characteristics of good design are ***discoverability*** and ***understanding***." — Don Norman, author of "The Design of Everyday Things"


Goal
What do you want to accomplish?
Plan
What are your options?
Specify
What action will you do?
Perform
How will you do it?
Compare
Is this OK?
Interpret
What does it mean?
Perceive
What happened?
World
Discoverability
FEEDFORWARAD
Affordances
Signifiers
Constraints
Mappings
Understanding
FEEDBACK
Feedback
Conceptual Models

Optional reading:

Check out Humans & Code chapter in solidbook

You can download an old copy in the Welcome section of the course.

Testing 

“Can I define what the system does? Can I be made aware of if it still does this successfully?”

- *Testing strategies*
- *Types of tests*
- *Schools of test-driven development*
- *Coupling & decoupling*
- *CI/CD & quality*

Strategy 

“How can we make most effective use out of the contending & assistive elements we currently have?” (Time, money, current codebase, developers, market, competition)

Vision & value

Planning

Architecture, Design, Testing

What we covered

Now let’s learn about the 12 essential principles