

Designing Software Driven by the Domain



Dino Esposito

@despos | <http://software2cents.wordpress.com>

Mirror the Business Domain



Key Points

DDD and Legacy
Code

Task-based design

User Experience
Architecture Briefs

Legacy Code

Code that works

Code you don't
like to have
around

Code you can
hardly get rid of

Not necessarily badly and poorly written code!

Common Aspects of Legacy Code

Has an established and implicit model

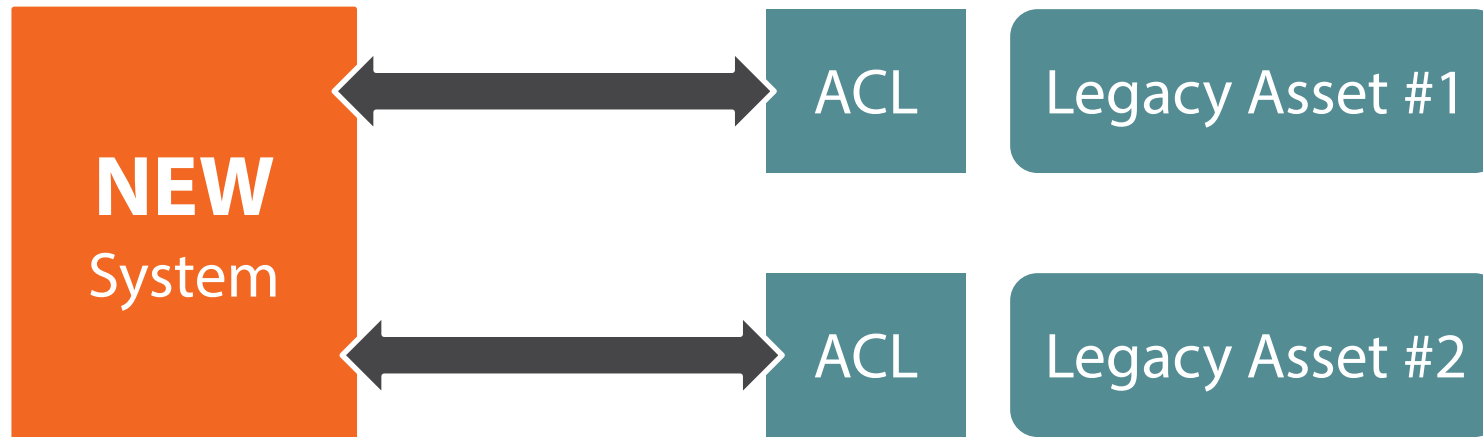
Doesn't typically have a public programmable API

Written to have some work done, not to be reused

Written years ago according to practices now obsolete

Legacy Code as a Service

- Rewrite from scratch with all the abstractions you need
- While rewriting consider, incorporating existing assets as services
- Put legacy assets behind a façade and connect it to the core application



LEGACY CODE

Not all assets are equal

- Some can be reused as services, some not
- Some are just old and obsolete

Before integrating

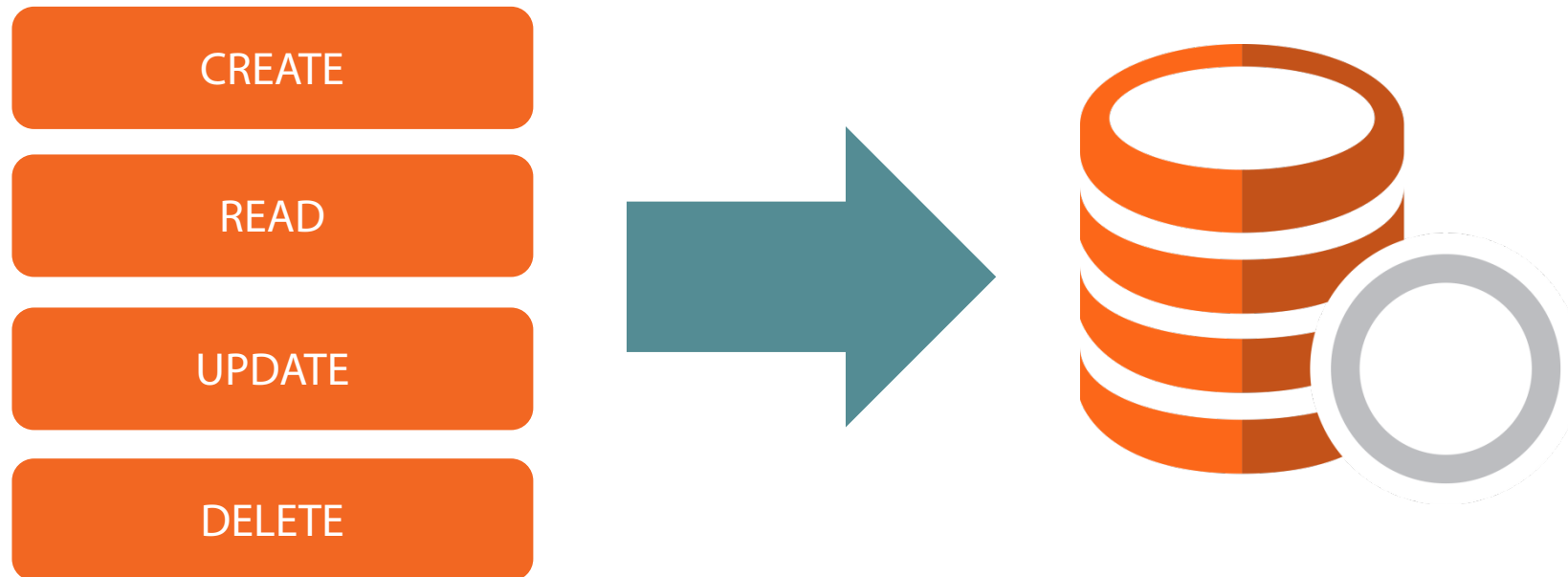
- Evaluate costs of not rewriting from scratch
- Evaluate costs of integration

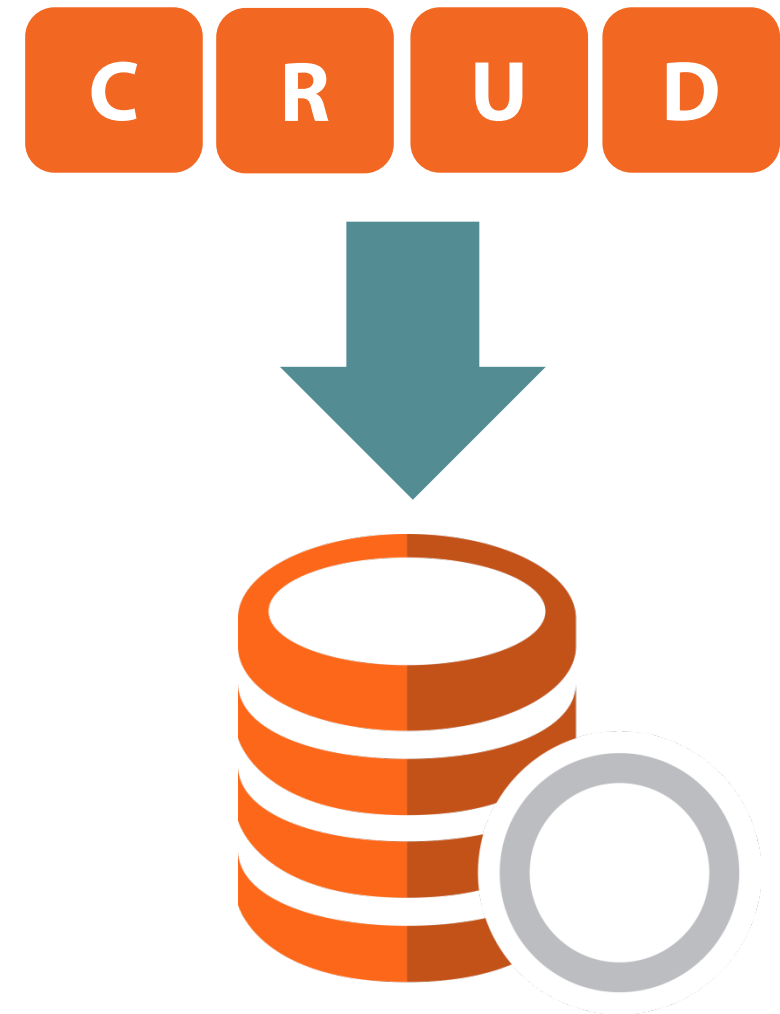
If it can become a service

- Just let it go
- Focus on other things to do!

to some extent

All systems are CRUD systems.





What's **CRUD** today?

Database-centric

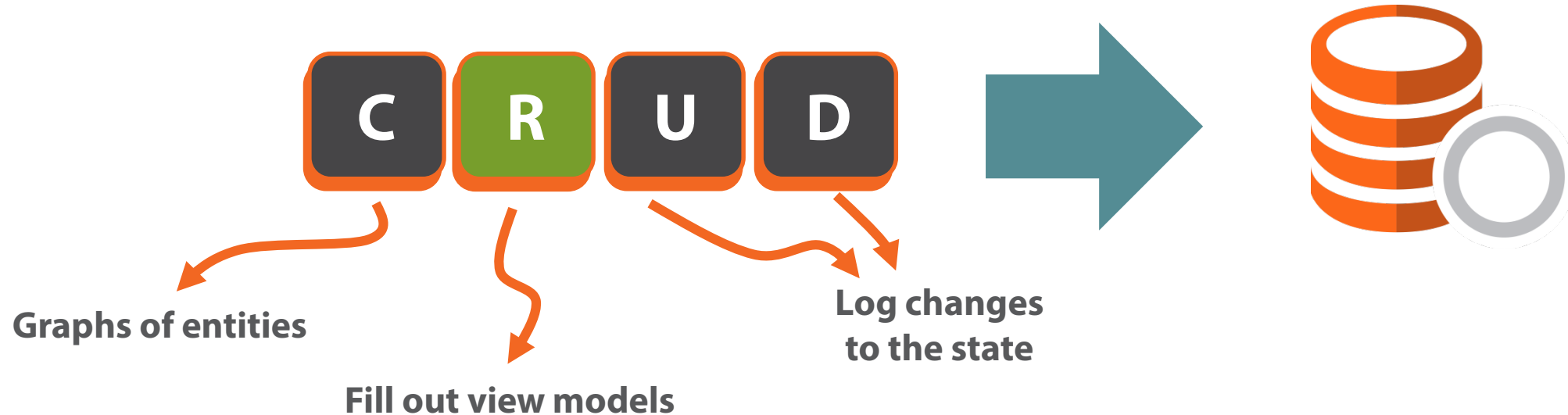
Persistence model is just one model to think about


Not tracking actions

Context of change must be tracked. An update can't simply update overriding the current state

Limited features

Plenty of business logic, concurrency issues, interconnected entities





Way too many great ideas have been first sketched
out on the paper napkins of some cafeterias.

What You **See** Is What You **Get**

What users perceive of each application is what they **see** and what they **get** while interacting with it.



The diagram features a horizontal line that divides the page. Above the line is a paragraph of text. Below the line are two terms, 'User Interface' and 'User Experience'. An orange arrow originates from the word 'see' in the text above and points down to 'User Interface'. Another orange arrow originates from the word 'get' in the text above and points down to 'User Experience'.

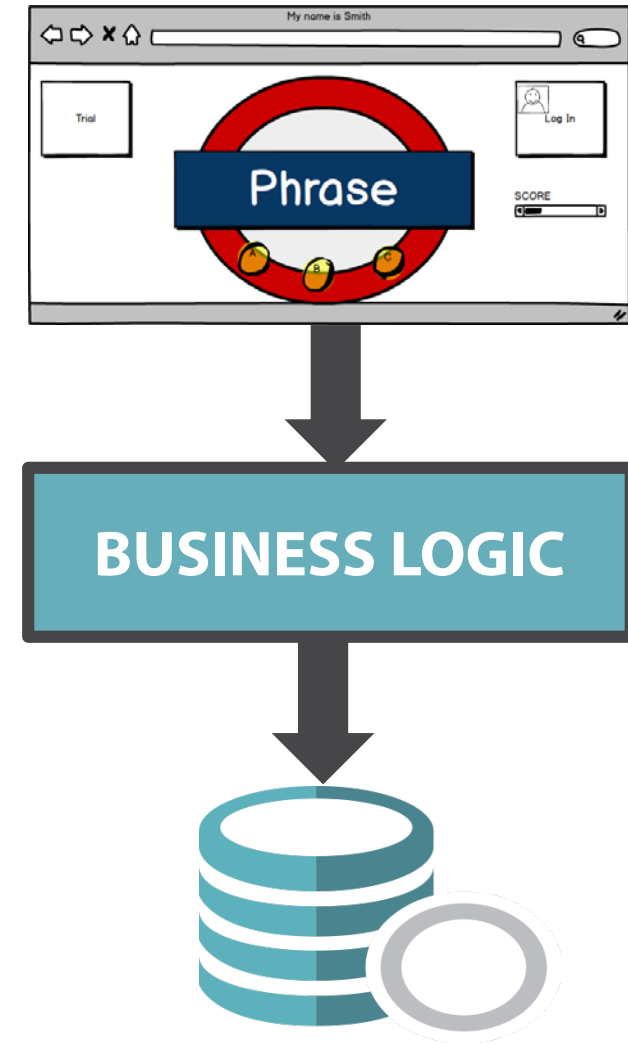
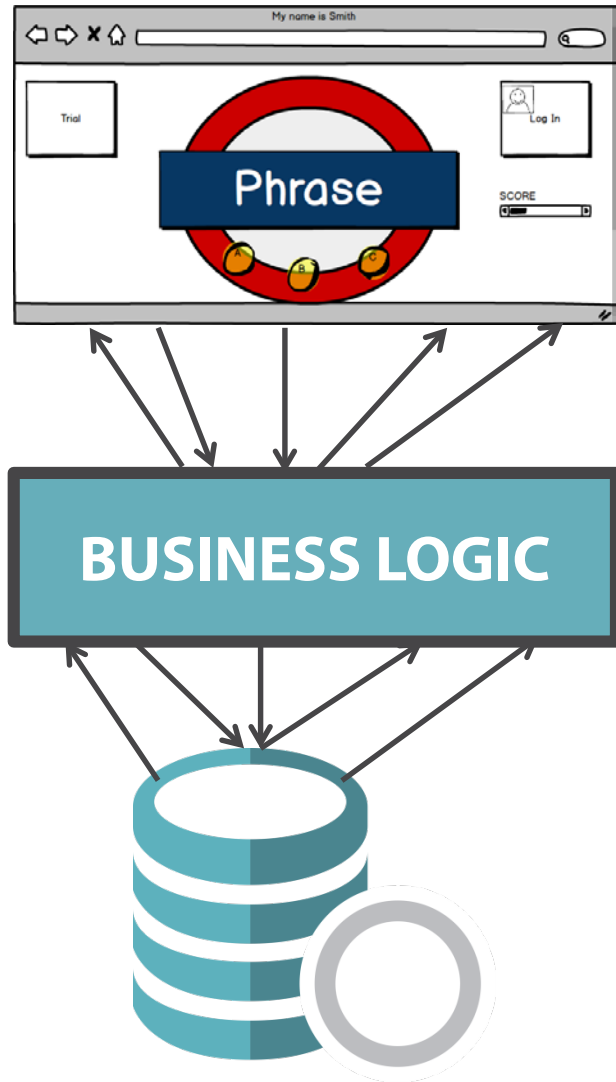
User Interface

User Experience

User Experience

The **experience** that users go through when they **interact** with the application.

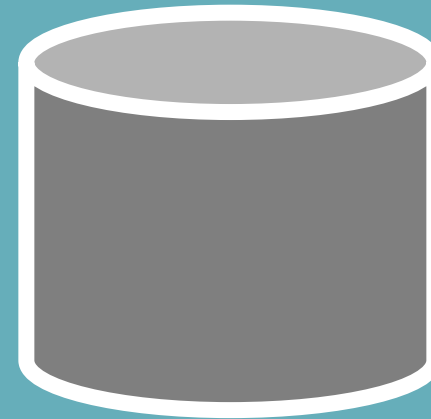
The Fine Art of Architecting Software



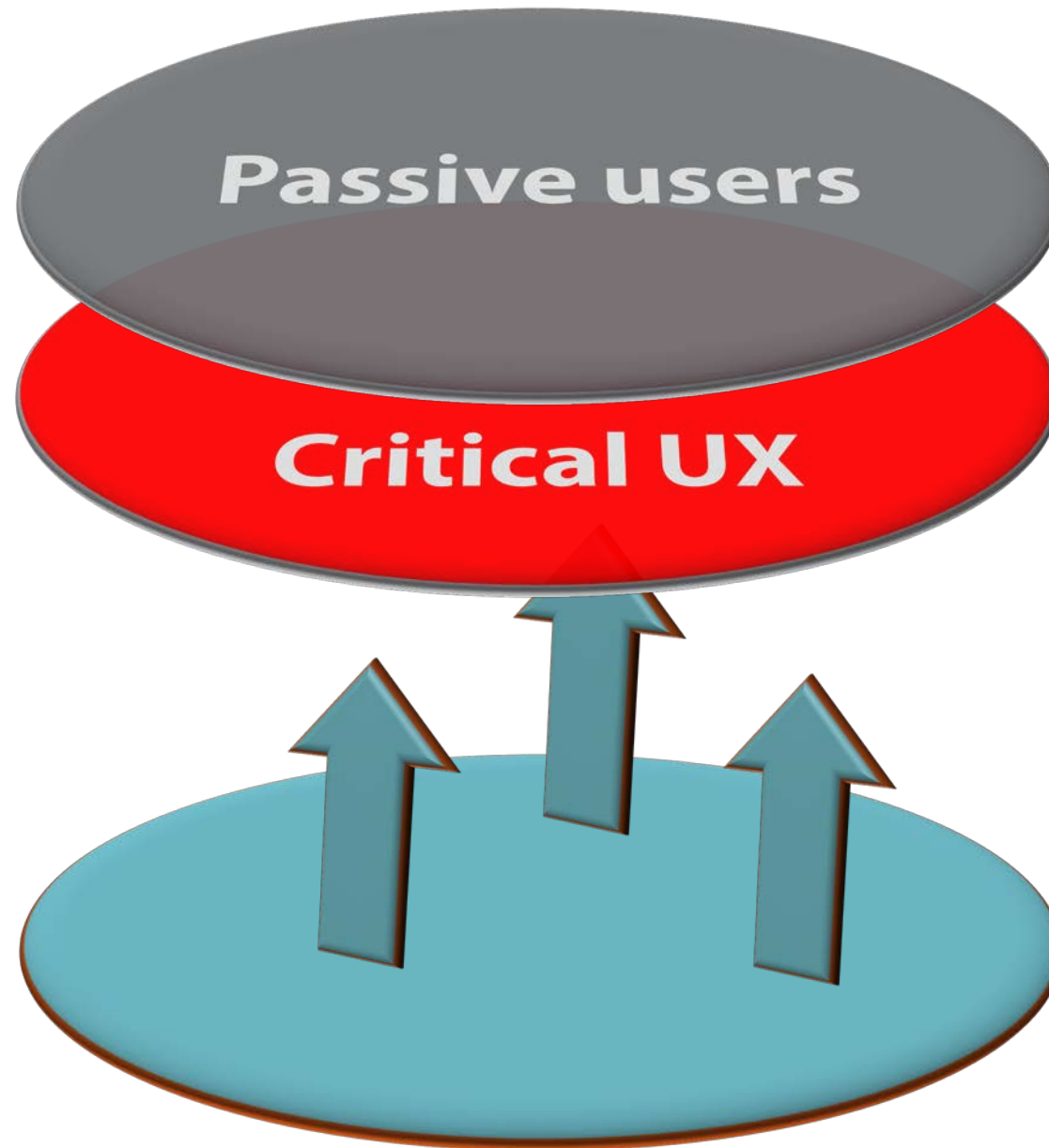
Bottom up



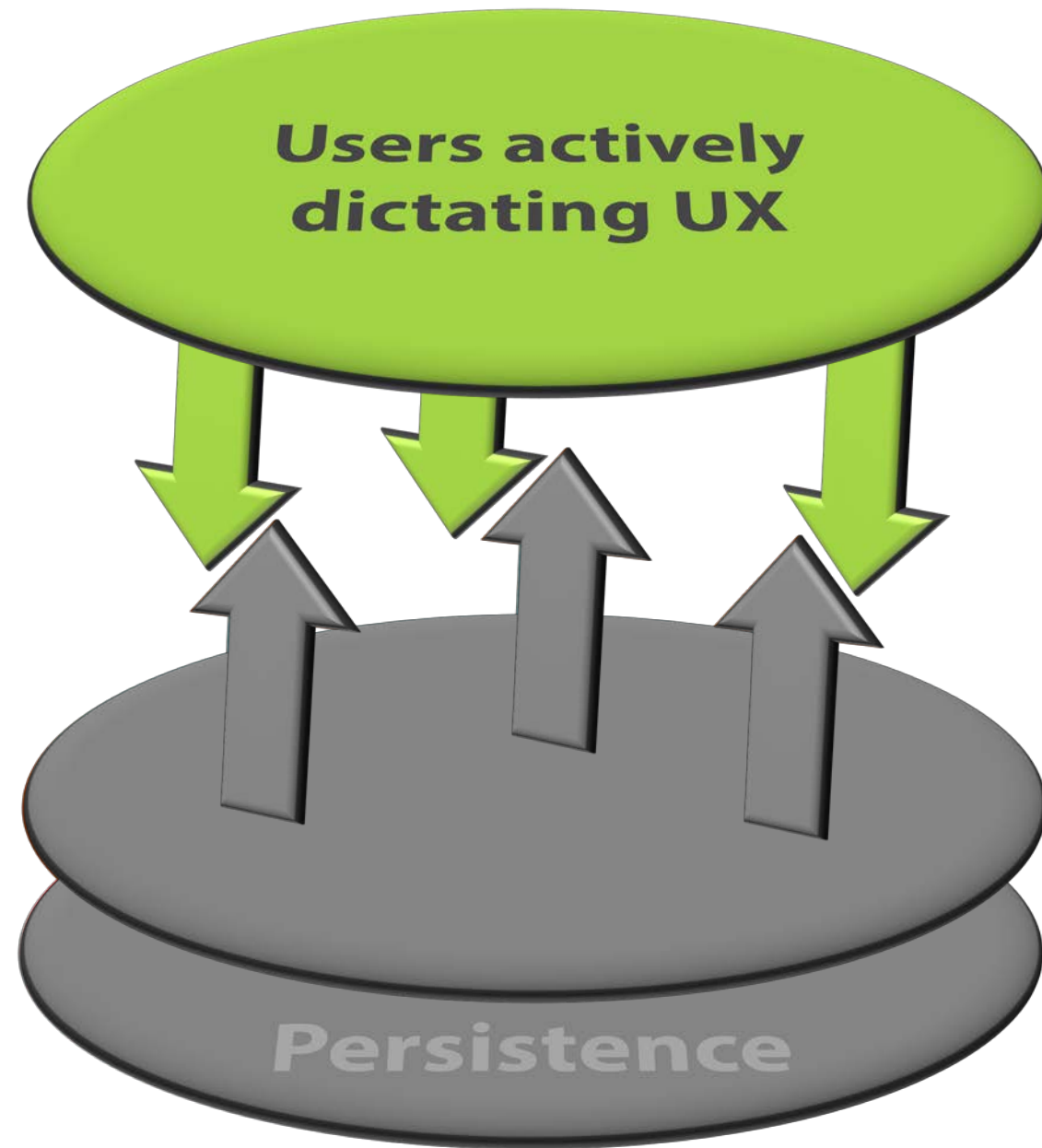
Top-down



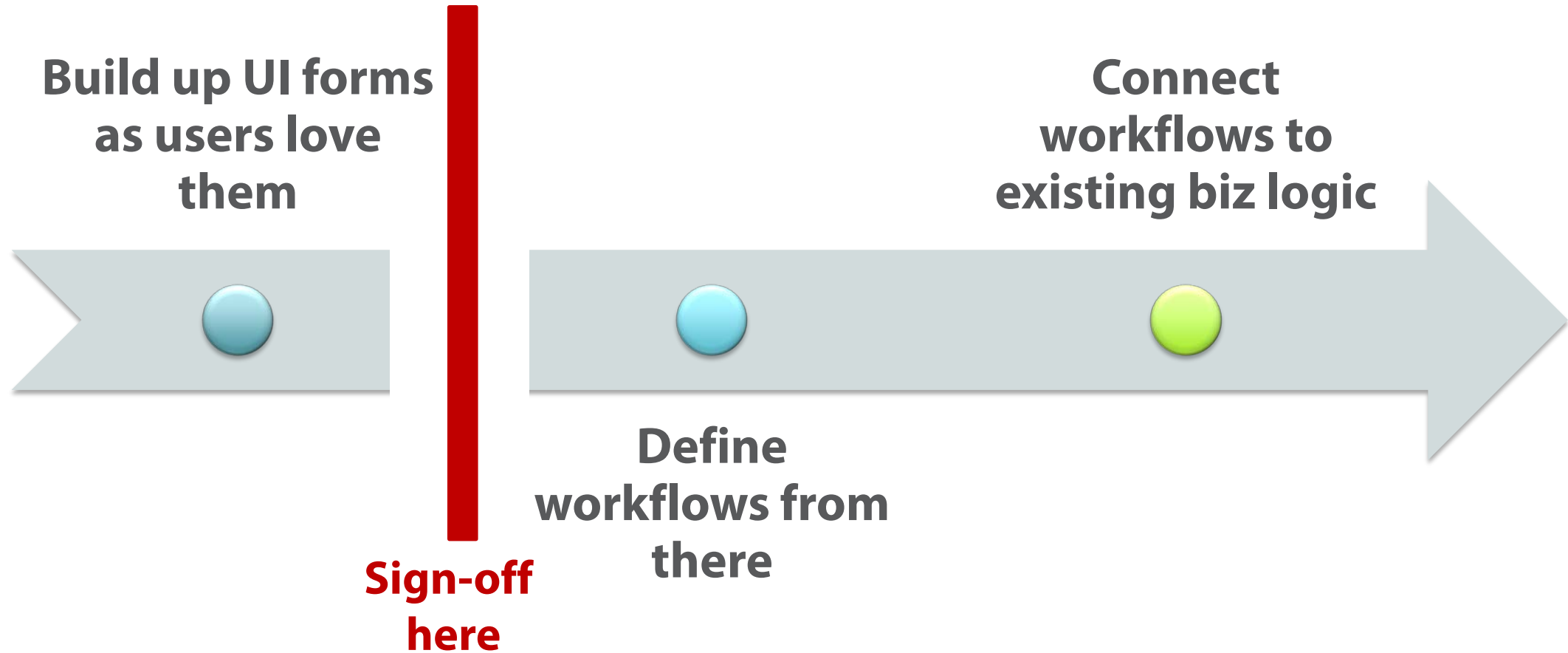
**Passive users
accepting any UI
enforcements**



**Bottom-up
Design**



UX-driven Design in 3 Steps



In Summary

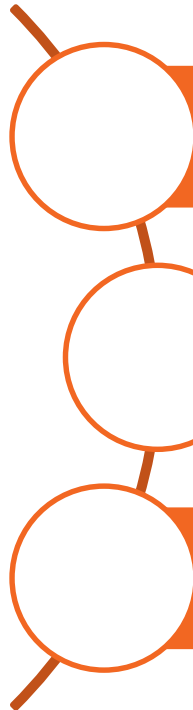
Sign off on what users really want

- Sketches/wireframes
- In some cases, mockups

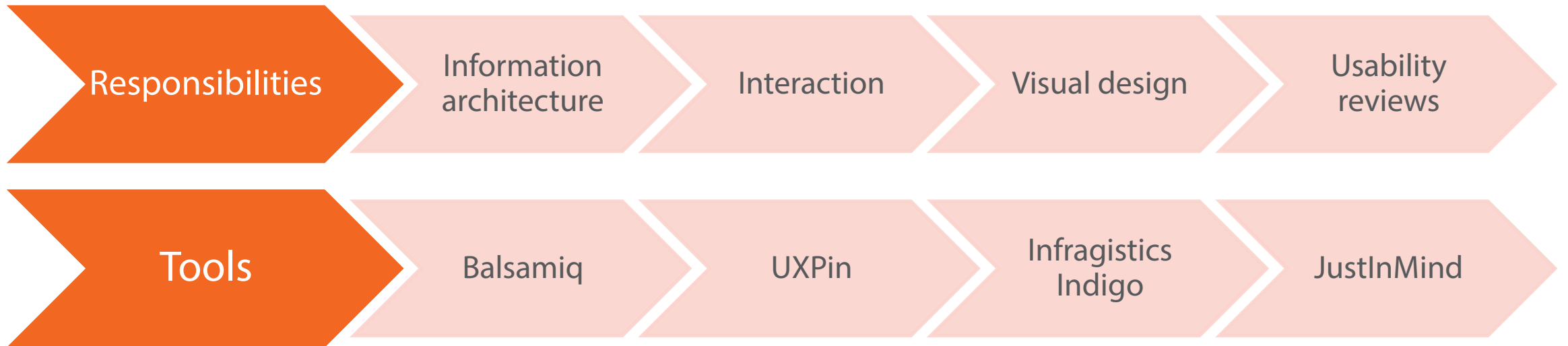
Build prototypes from requirements

- Do not start on billable hours until signed off

For each screen have a basic flowchart

- 
- 1 Determine what comes in and out and create **view model** classes
 - 2 Make application layer endpoints **receive/return** such DTO classes
 - 3 Make application layer **orchestrate** tasks on layers down the stack

The UX Architect



Pillars of Modern Software

