Container

Coding

Patrick Schwisow

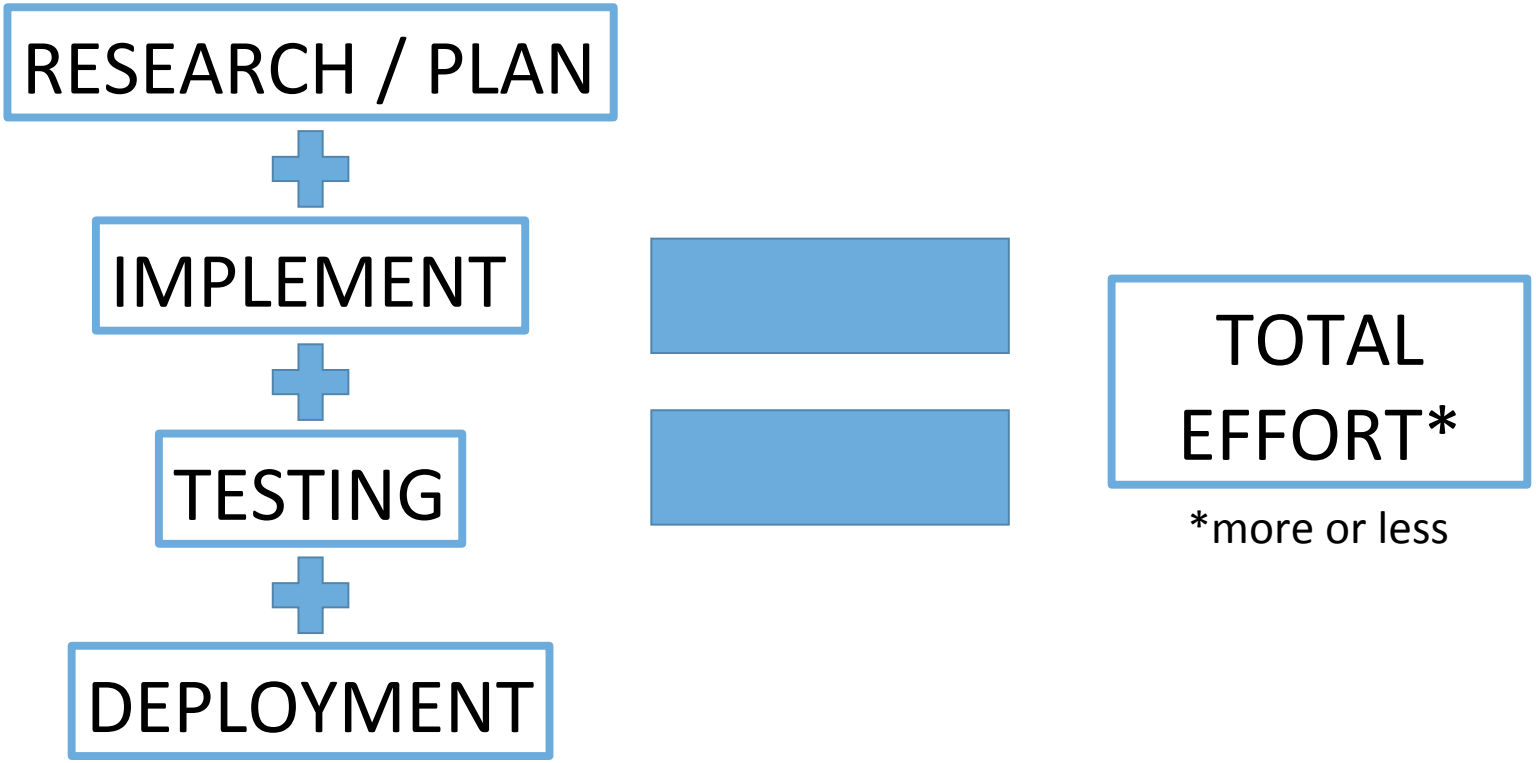
Midwest PHP Conference

X Y Z Q

March 14, 2015

A Long Time Ago, in an Office Far, Far Away…

Problem or No Problem?



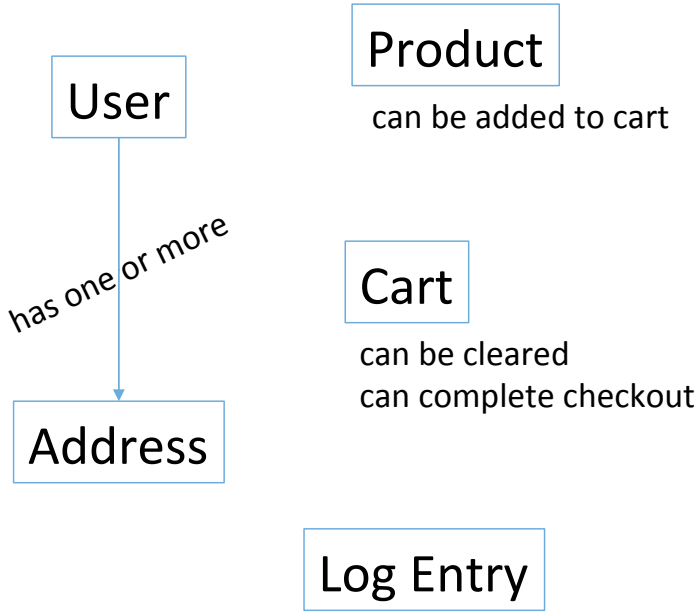
“Old School PHP”

- No code structure / minimal structure
- No separation of concerns
- Code is typically procedural (not OO)

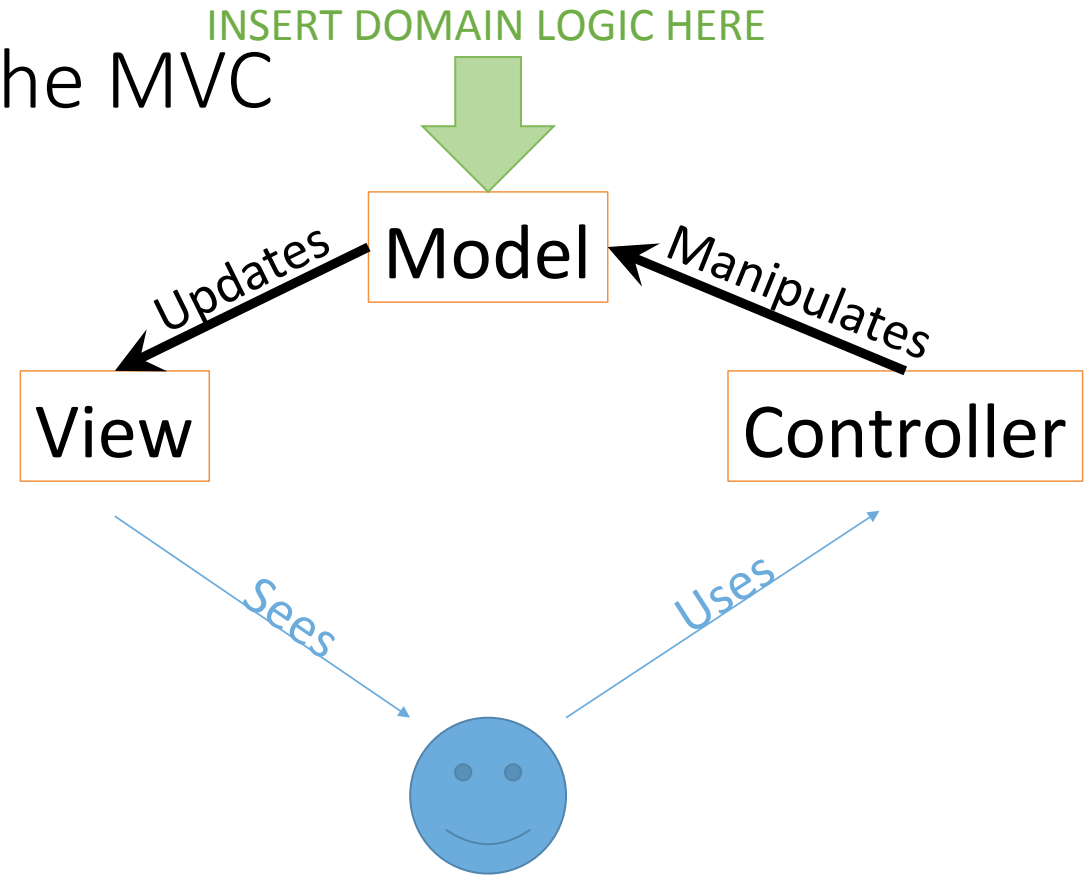
What is “Domain Logic”?

A set of rules that defines:

- The different types of “things” that your software cares about
- How the “things” relate to each other
- What processes and workflows are allowed in the system



Bring in the MVC



What's in the Model?

At this stage, Table Modules

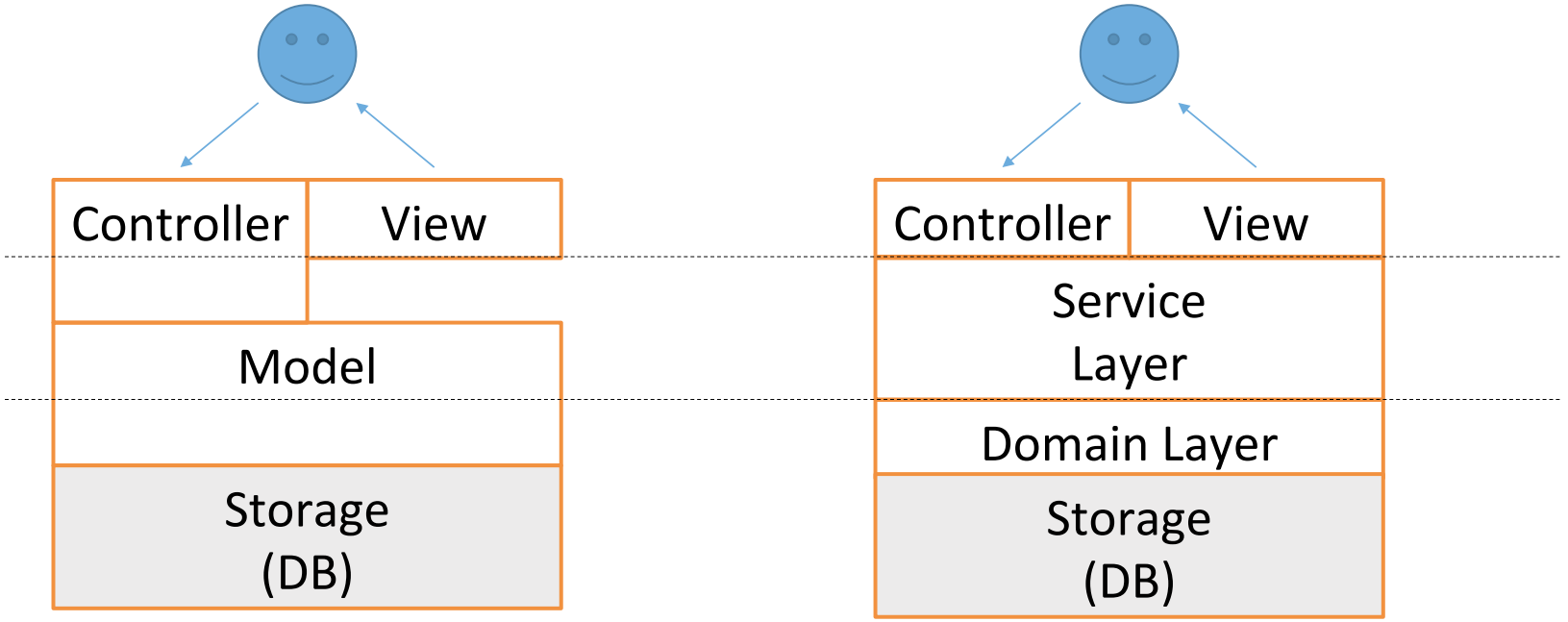
- One class per table with all logic to fetch / update data
- One instance per invocation
- Each class can only do single table operations

Scorecard: MVC

- Improvements:
  - Controller and View don't need to care how tax is calculated → ***reduced time to plan and implement***

- Problems:
  - Domain logic is still intertwined with persistence (database) logic → ***testing is still difficult***

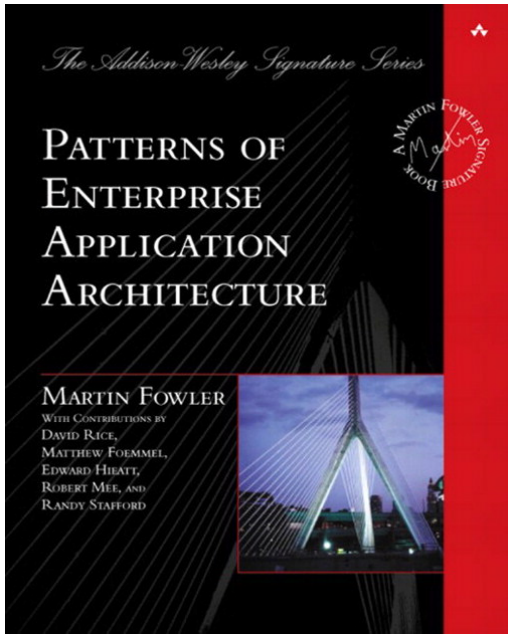
Splitting the Layers



Domain Layer *vs.* Service Layer

- Service Layer
  - Unified means of accessing business logic
  - Includes some application logic for accessing things
- Domain Layer
  - Representation of our business logic
  - Application- and invocation-independent

- Recommended Reading: *Patterns of Enterprise Application Architecture* by Martin Fowler



Scorecard: Service / Domain Layer Split

- Improvements:
  - Service layer can be tested in isolation (by mocking domain layer) → ***reduce testing time, improve testing reliability***
  - Complete extraction of domain logic from controllers → ***reduce planning and implementation time***

- Problems:
  - Domain Layer still mixes persistence with domain logic
  - Domain logic cannot be tested in isolation from persistence logic

ORM to the Rescue?

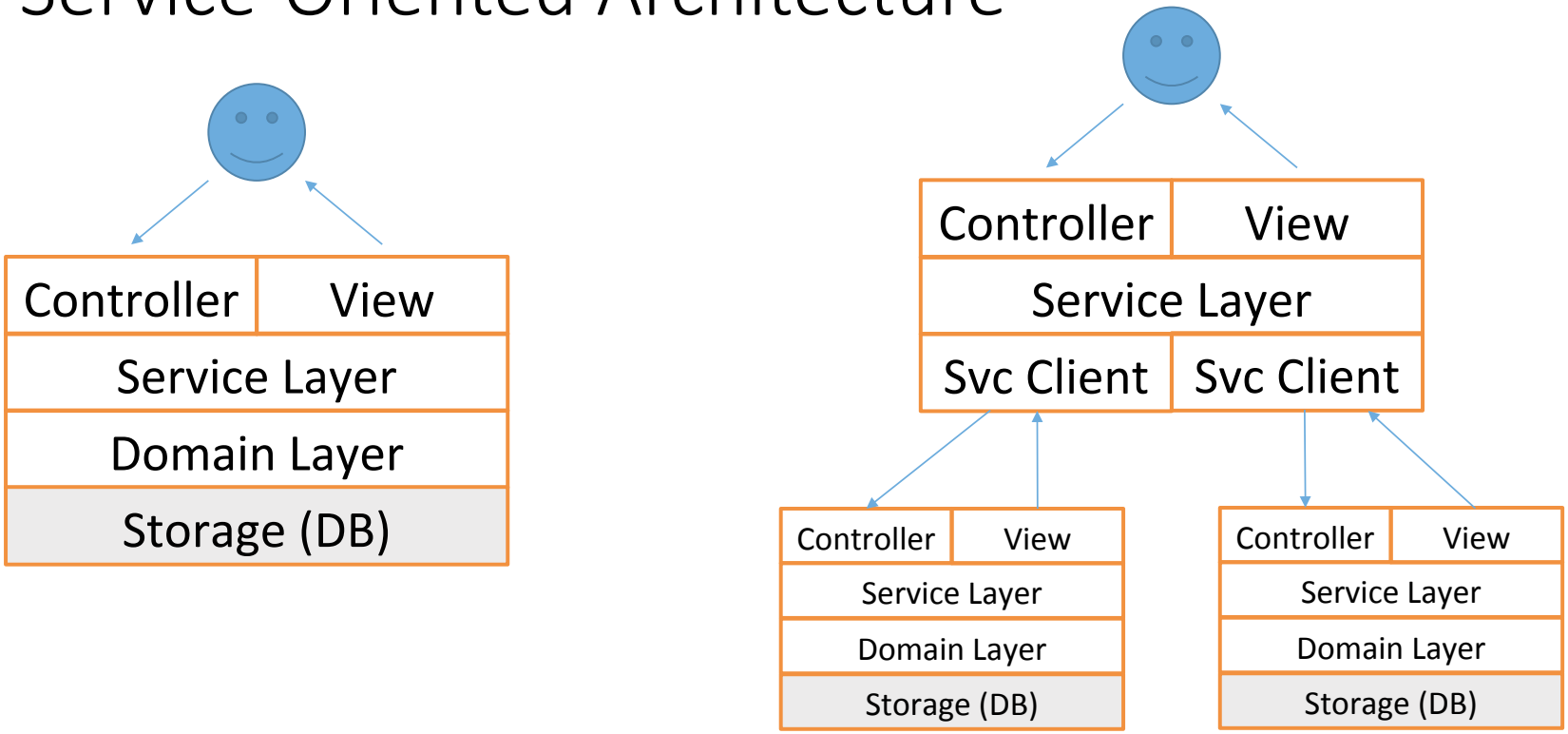
- Object-relational mapping (ORM) replaces Table Modules with Entities
- Doctrine 2 ORM is a PHP implementation
- Entities
  - Plain Old PHP Objects (POPO's)
  - Don't directly access databases
  - Entity logic can easily be tested in isolation

Scorecard: ORM

- Improvements:
  - Complete separation of domain logic and persistence logic → ***reduce testing time, improve testing reliability***
  - Changes to domain logic (typically) require updates in a smaller area of code → ***reduce planning and implementation time***

- Problems:
  - Learning curve for ORM
  - Data architecture must conform to ORM
  - Some negative performance impact, limited opportunities to optimize

Service-Oriented Architecture



Scorecard: Service-Oriented Architecture

- Improvements:
  - Designing to API simplifies definition of test cases and creation of test mocks → ***reduced time creating tests, improved test reliability***
  - Services can serve many different clients
  - Opportunities to scale development teams (separate ownership of services)
  - Services can be implemented with language / technology that is best suited

- Problems:
  - Coordination between apps and services (and their owners)
  - Network overhead

Takeaways

- Plan for change
- Manage technical debt
- More structured == more maintainable
- Increasing application size requires increasing separation of concerns
- Application architecture is a series of trade-offs

Who am I?
Feedback / Contact / Slides

- Software Engineer at Shutterstock
- Zend Certified Engineer – PHP 5 & Zend Framework
- Founder / Organizer of Lake / Kenosha PHP
- Email: patrick.schwisow@gmail.com
- Twitter: @PSchwisow
- Slides: https://github.com/PSchwisow/Miscellaneous/
- Sample Code: https://github.com/PSchwisow/container-coding/
- Joind.in: https://joind.in/talk/view/13087