ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Angular Testing

5 - Testing Strategies

Two Competing Schools of Unit Testing

Differences

London ~ Unit Test

- Unit is a **class**
- Mock everything except the class
  - Very tightly coupled to implementation
- Disadvantages
  - No refactoring
  - Lots of code for mocking
  - No interplay testing
- Advantages
  - Edge cases, finding bugs, exploratory
  - Great code quality (FP)
  - Fast

Detroit (Chicago) ~ Integration Test

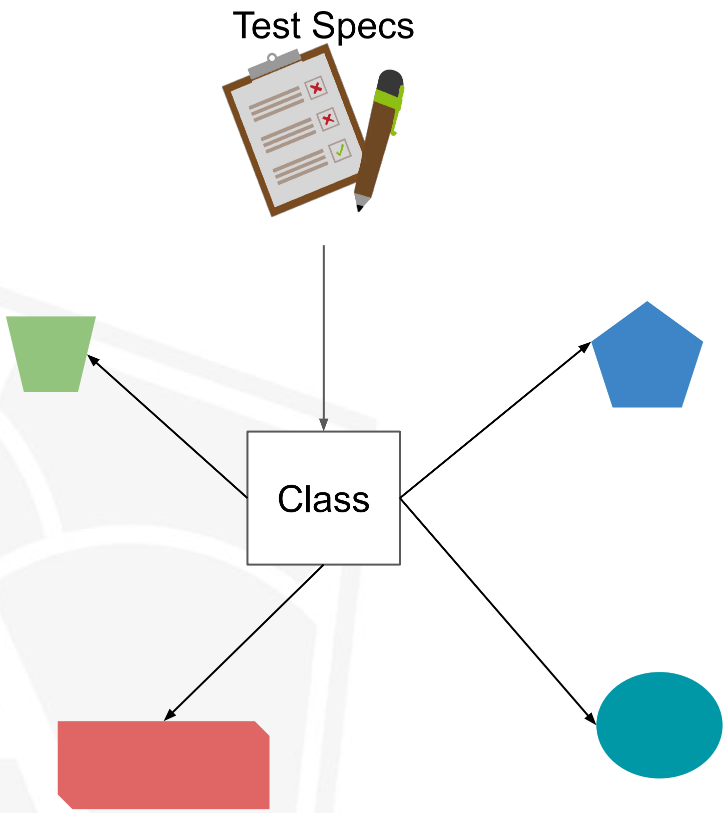
- Unit is a **behaviour**
- Mock out-of-system dependencies
  - Runs against an API (UI)
- Advantages
  - Great for refactoring
  - Efficient (coverage)
- Disadvantages
  - Large setup required
  - Slow
  - Hard (Async, Change Detection, DOM,...)
  - Code Quality is of no concern

It is not Unit vs. Integration

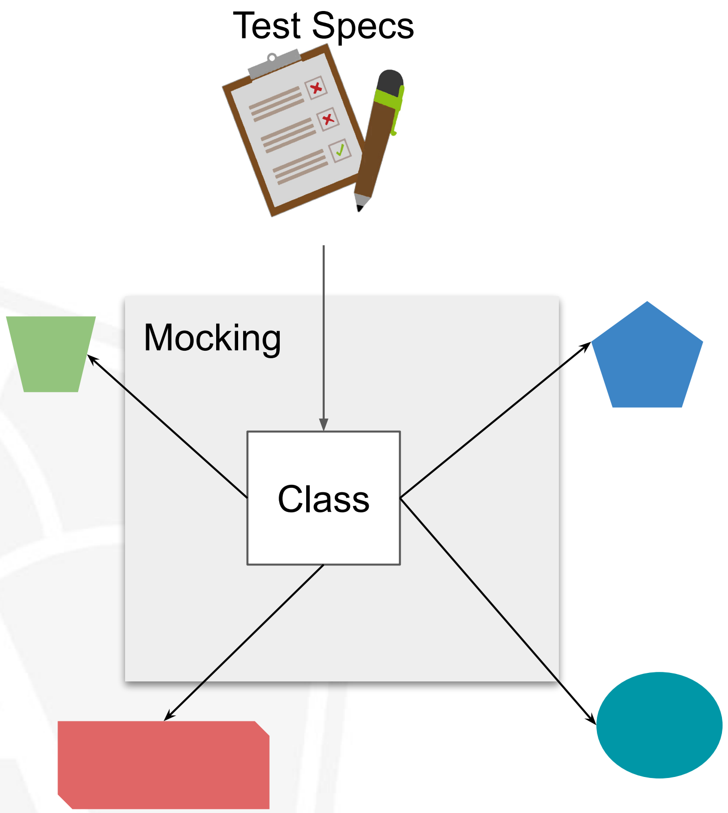
It is about the right balance

Removing Unit Tests???

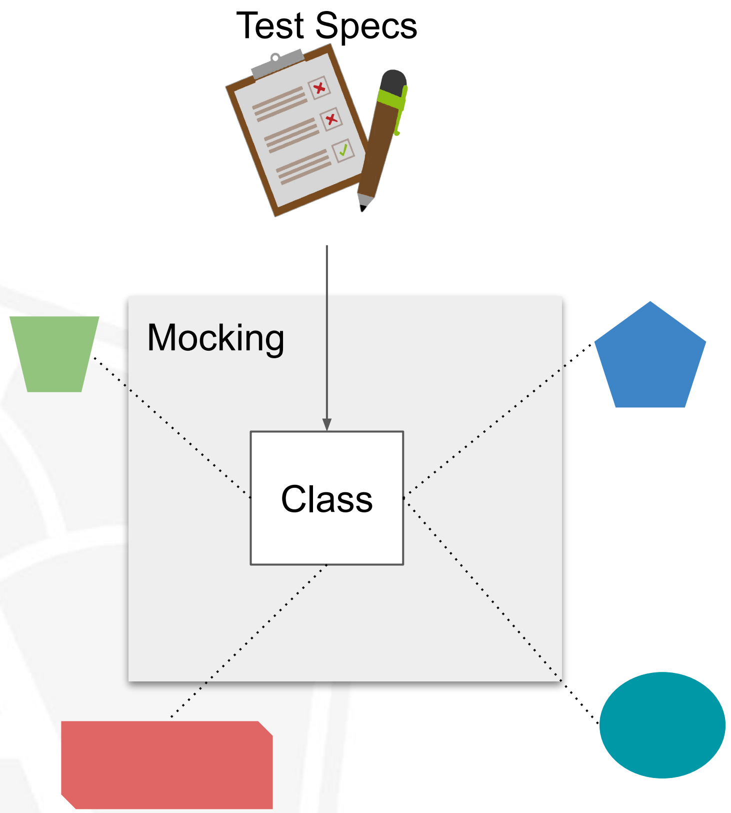
London Style



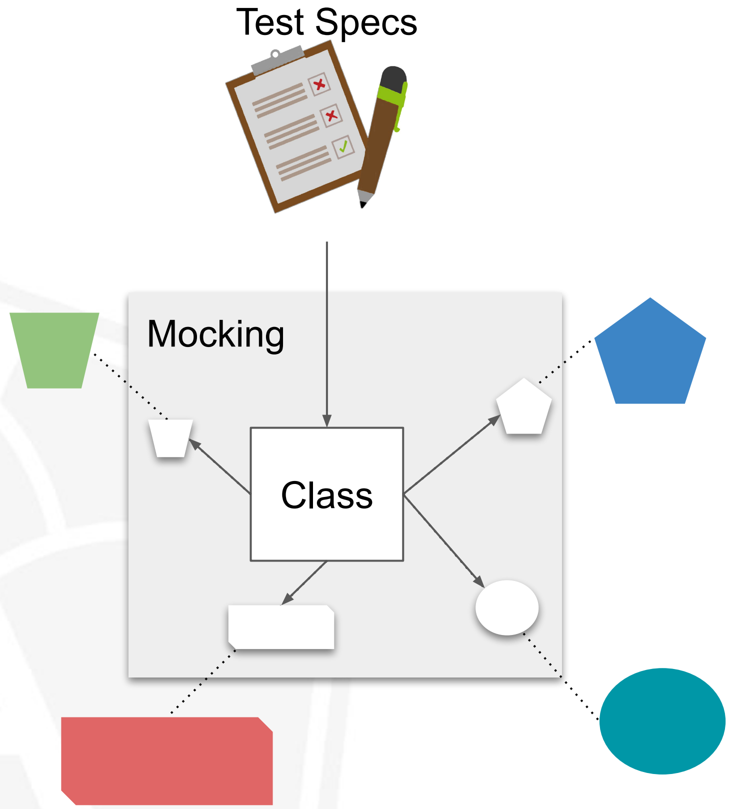
London Style



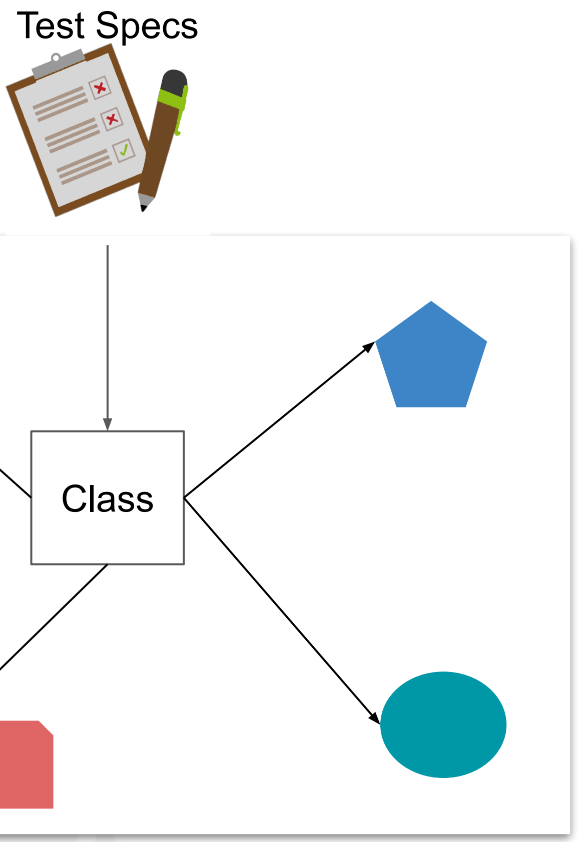
London Style



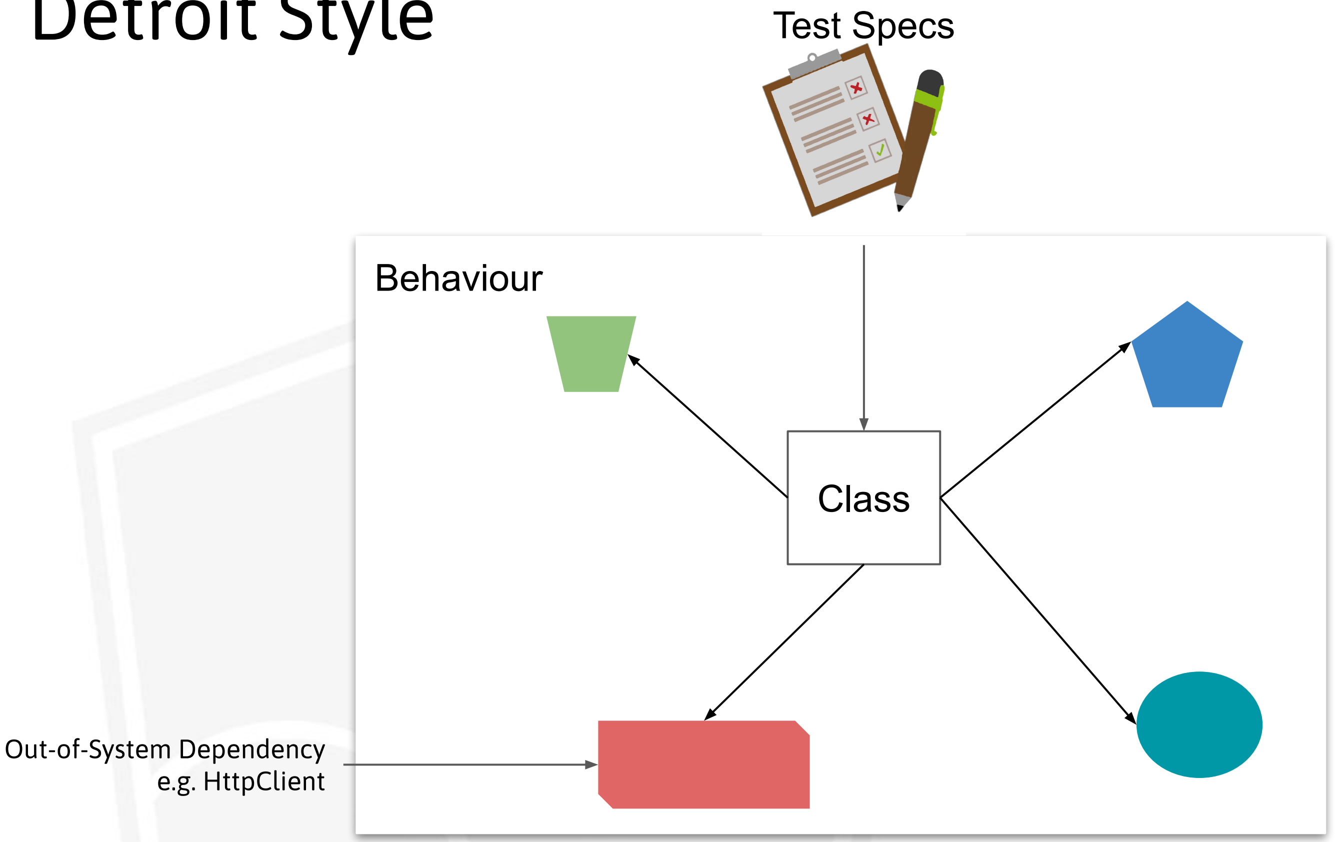
London Style



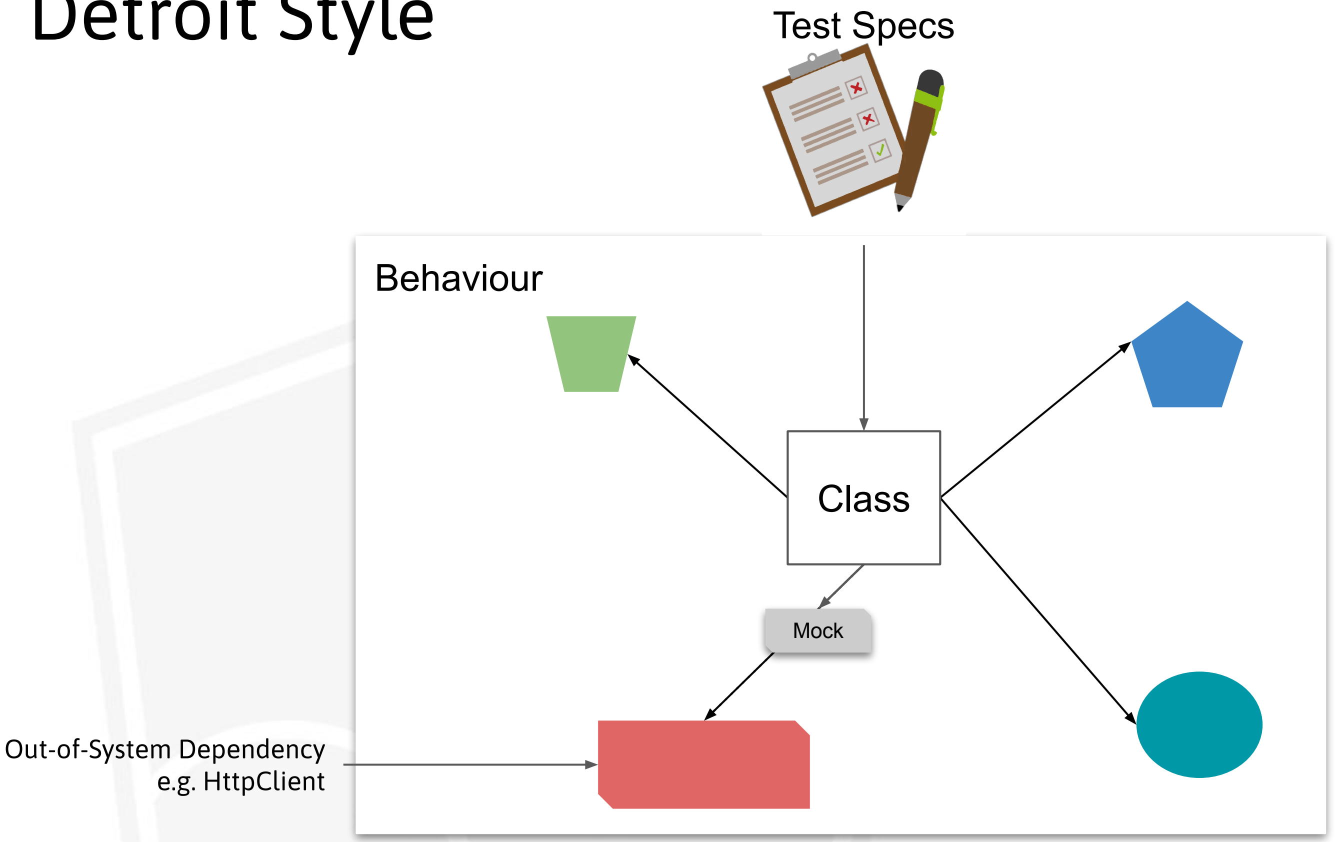
Detroit Style



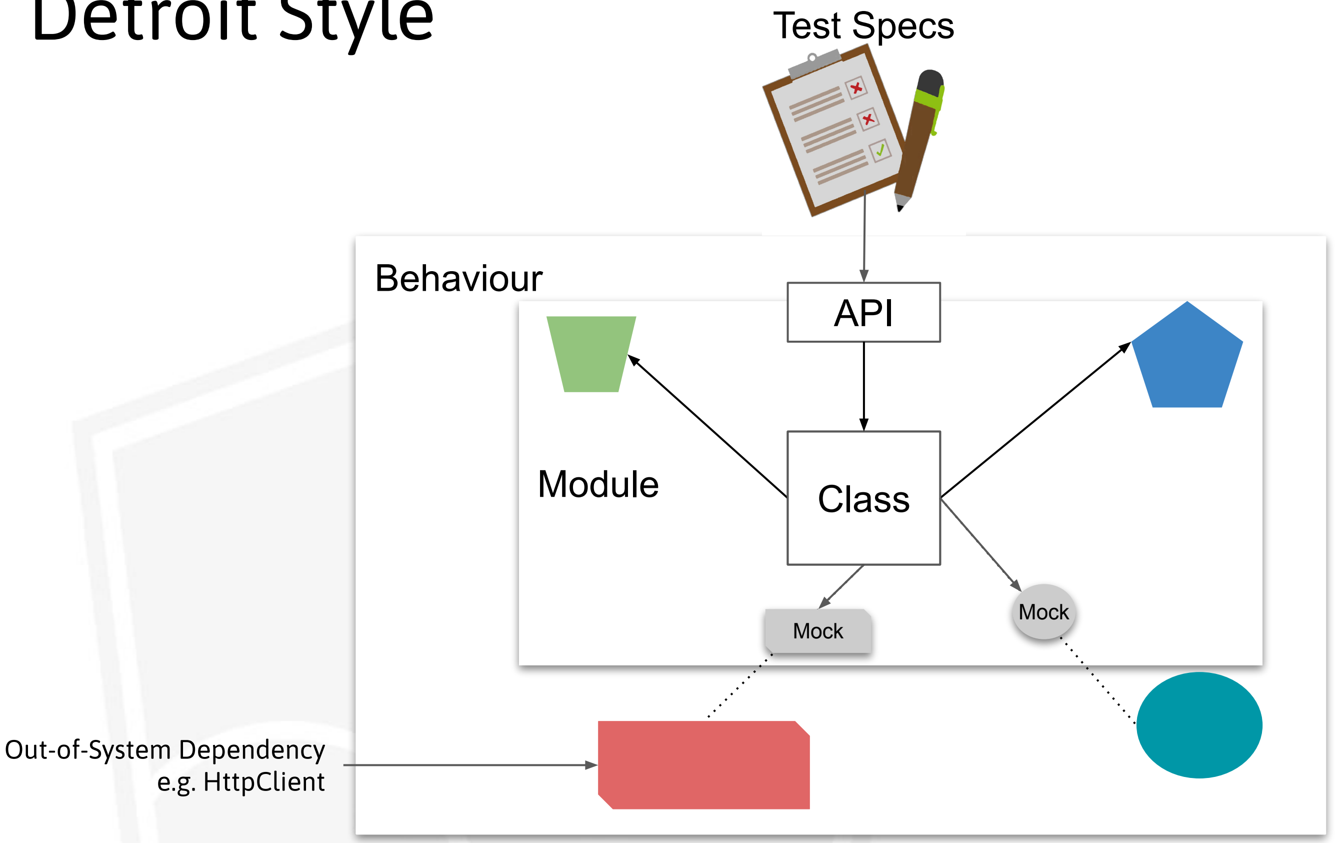
Detroit Style



Detroit Style



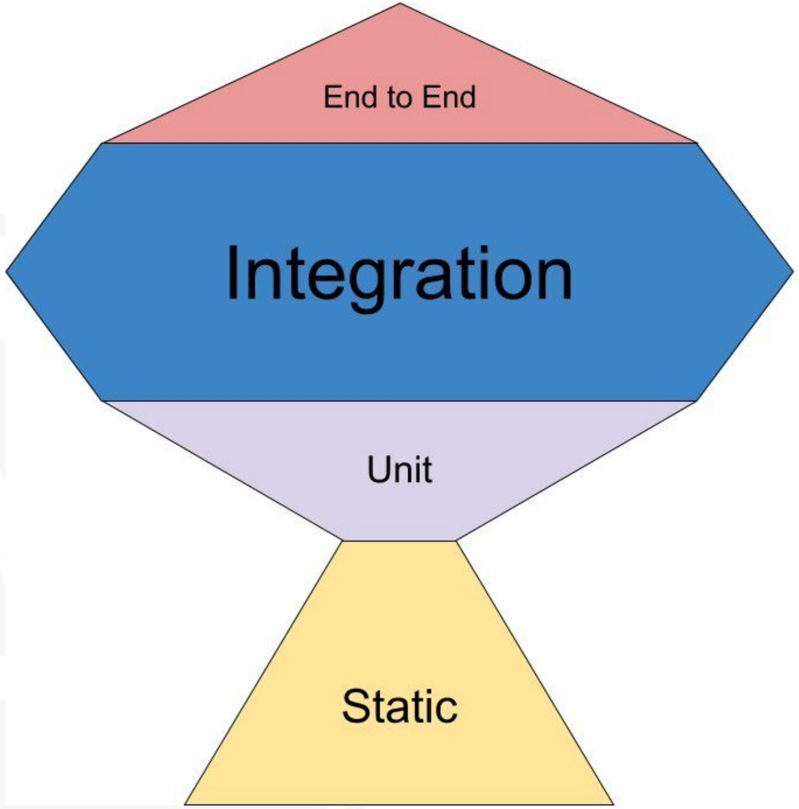
Detroit Style



Criterias

- Speed
  - Execution
  - Writing & Maintaining
  - CI & Local Setup
- Timing
- Industry
- Effectiveness
- Application Type

Testing According to ROI



Kent C. Dodds: https://twitter.com/kentcdodds/status/960723172591992832

Application Types

1: Anemic

- Most parts of data processing (unit test) done in backend
- Frontend as "proxy" → less logic
- Integration is King

2: Autonomous

- Backend acts as Store
- Lots of Logic in Frontend
- Unit Tests & Integration Tests are critical

3: Complex UI

- ViewState in different variations
- Go for Component Tests

4: UI Library

- Library Vendor
- Storybook
  - Visual Regression
  - Cypress

5: Too big to Test

- "I can only run unit tests because of build time"
- Architectural Engine
- Split Logic and UI from each other
- Functional Style
- Example: RulesEngine for Workflow
  - Unit Tests for RulesEngine
  - Integration Tests for Execution into UI

Trust your instincts!

- It doesn't feel right
- What are we actually testing here? If a function is called? Really?!
- I don't see any value in testing.
- I never discovered a bug with my tests.
- I am wasting my time with writing tests instead of producing "real" code.

Testable Architecture

Different Testing Techniques

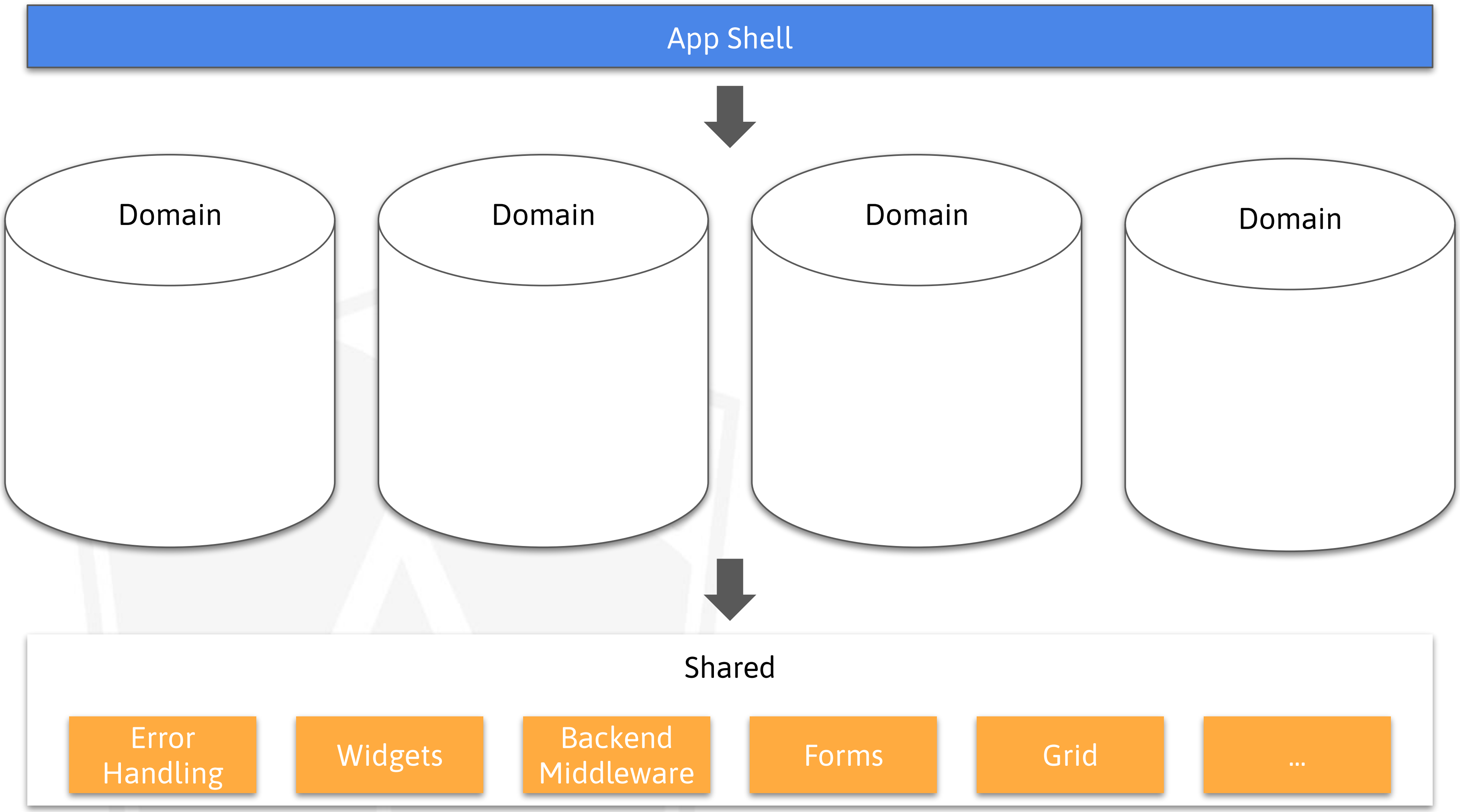
1. Unit / Integration Range
   a. Full mocking, no TestBed
   b. Selected mocking, without DOM interaction
   c. Selected mocking, DOM interaction
   d. Most minimal mocking, DOM Interaction
2. Exotic
   a. RxJs Marbles
   b. Visual Regression
   c. Component Tests via Storybook/Cypress (E2E)

Potential Problems

- Unit Tests (London)
    - What technique should be applied?
    - Too much mocking
    - Should I have unit test for everything?
- Integration Tests (Detroit Unit)
    - Too much setup required → feels like E2E
    - What should I mock?

Testable Architecture

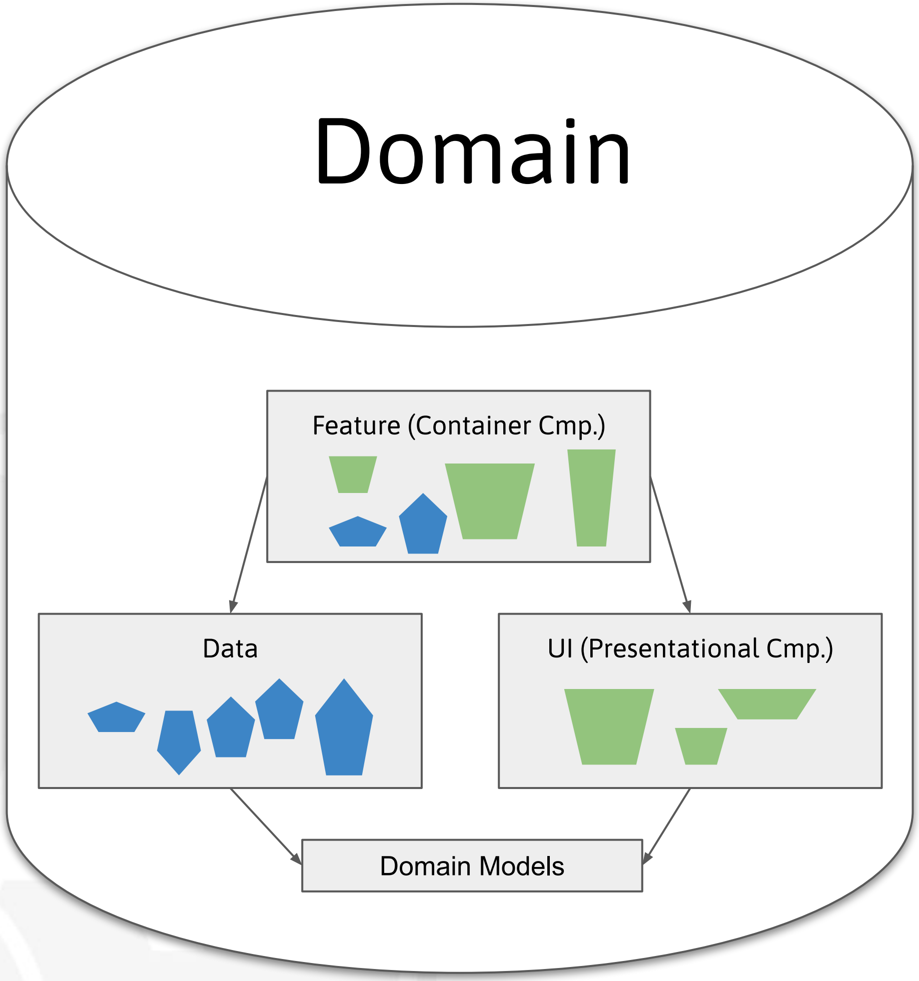
- Unit Tests
  - Class has a defined type
  - One testing technique per Type
- Integration Tests
  - Reduction of dependencies via domain/feature boundaries
  - Integration Test per Domain/Feature
  - Entry point is the feature component


App Shell
Domain
Domain
Domain
Domain
Shared
Error Handling
Widgets
Backend Middleware
Forms
Grid
...

🟩 Component

⬟ Service

▭ Module


Domain
Feature (Container Cmp.)
Data
UI (Presentational Cmp.)
Domain Models