



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Angular Testing

5 - Testing Strategies

HttpTest

```
it("should use Angular's http mock", () => {
```

```
  TestBed.configureTestingModule({  
    declarations: [RequestInfoComponent],  
    imports: [ReactiveFormsModule, HttpClientTestingModule],  
  });
```

Instead of HttpClientModule

```
  const httpController = TestBed.inject(HttpTestingController);  
  const fixture = TestBed.createComponent(RequestInfoComponent);  
  fixture.componentInstance.search();
```

Runs AFTER http
request

```
  const request = httpController.match((req) => !!req.url.match(/nominatim/))[0];  
  request.flush([{ street: "Domgasse", streetNumber: 5 }]);  
  expect(lookupResult.textContent.trim()).toBe("Address found");  
});
```



RoutingTest

- RouterTestingModule provides routing functionality for tests
- Location can verify the expected url
- RoutingConfiguration is required



Two Competing Schools of Unit Testing



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Differences

London

- 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

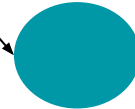
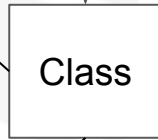
Detroit (Chicago)

- 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



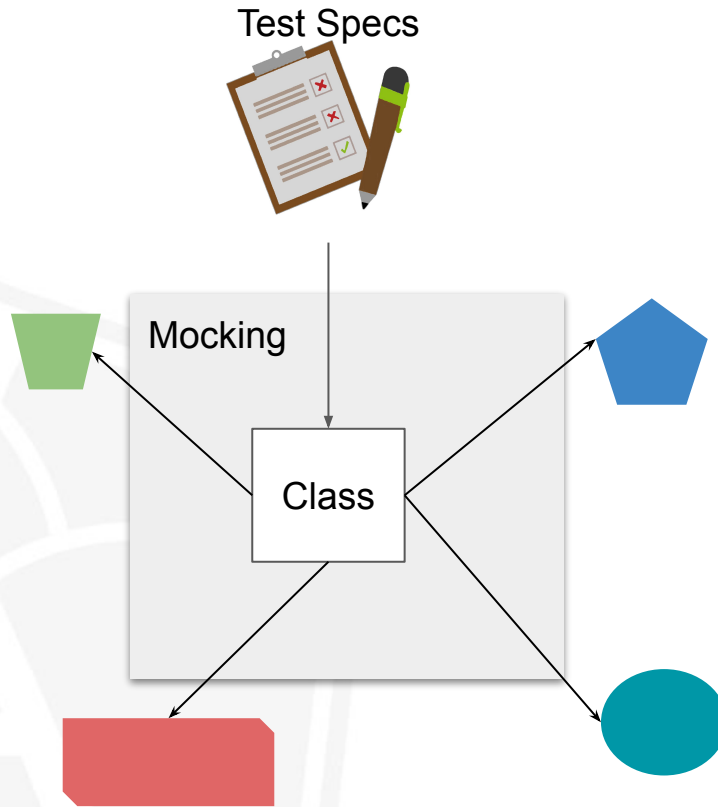
London Style

Test Specs



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

London Style



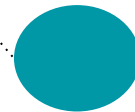
London Style

Test Specs



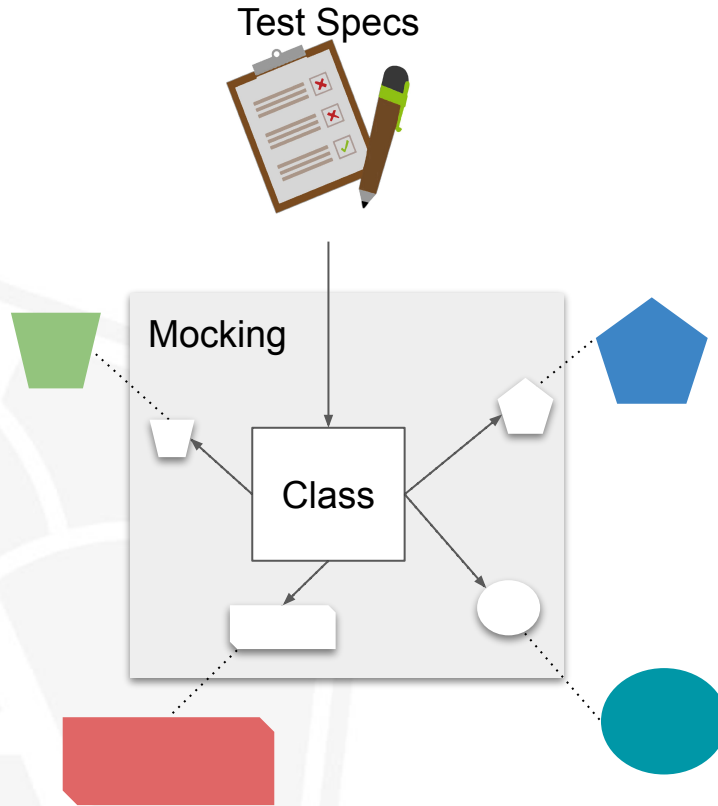
Mocking

Class



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

London Style

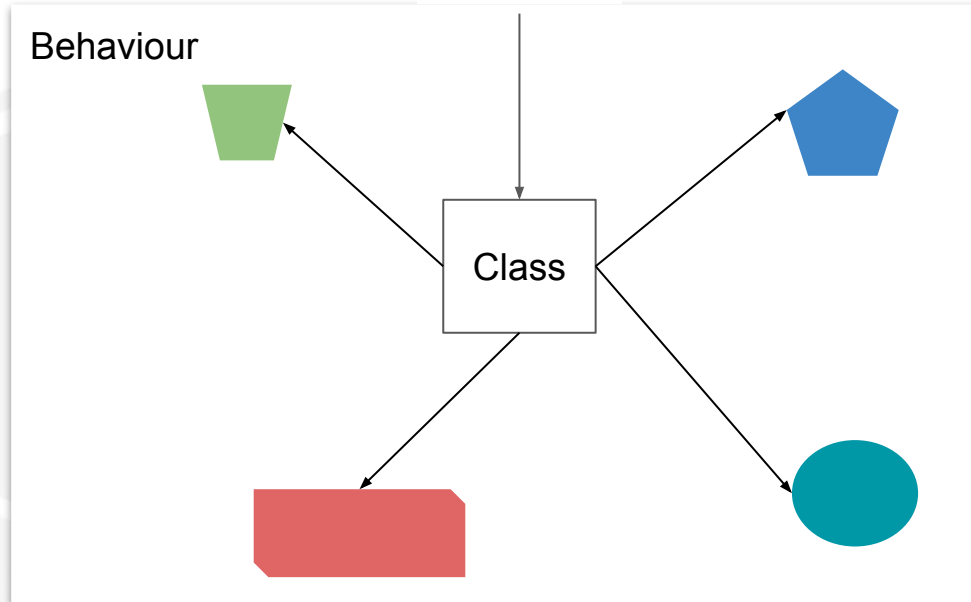


Detroit Style

Test Specs



Behaviour



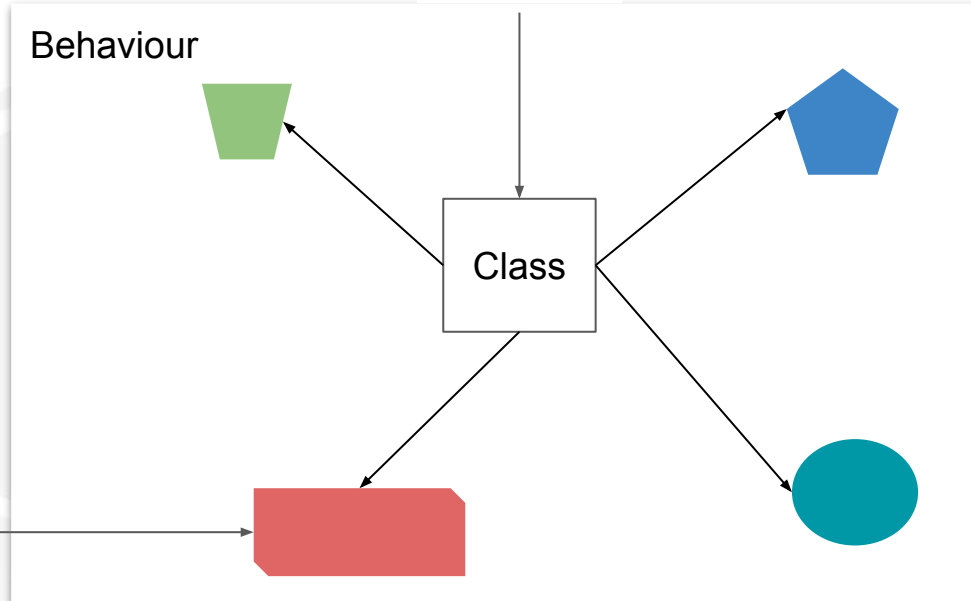
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Detroit Style

Test Specs



Behaviour



Out-of-System Dependency
e.g. HttpClient



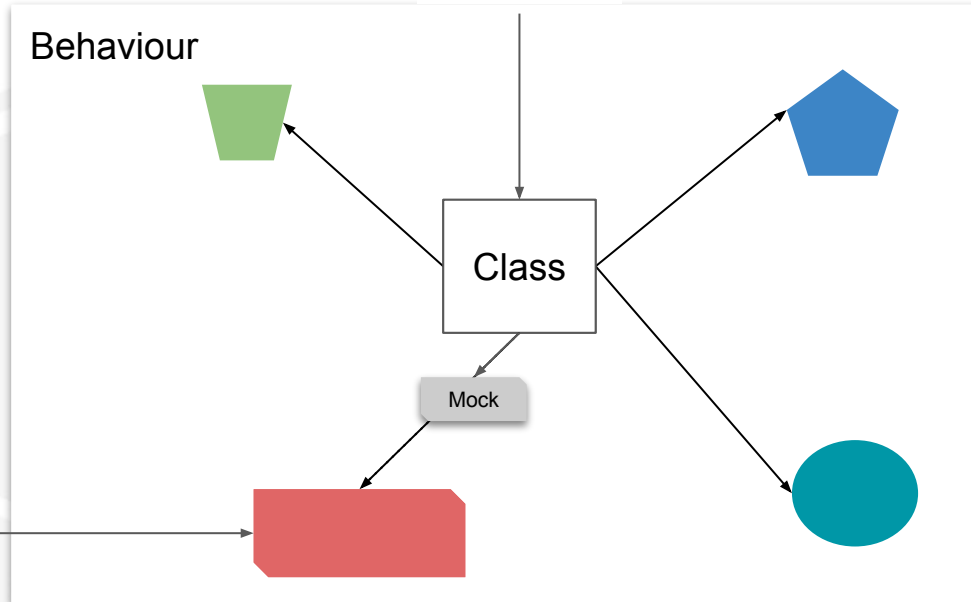
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Detroit Style

Test Specs



Behaviour

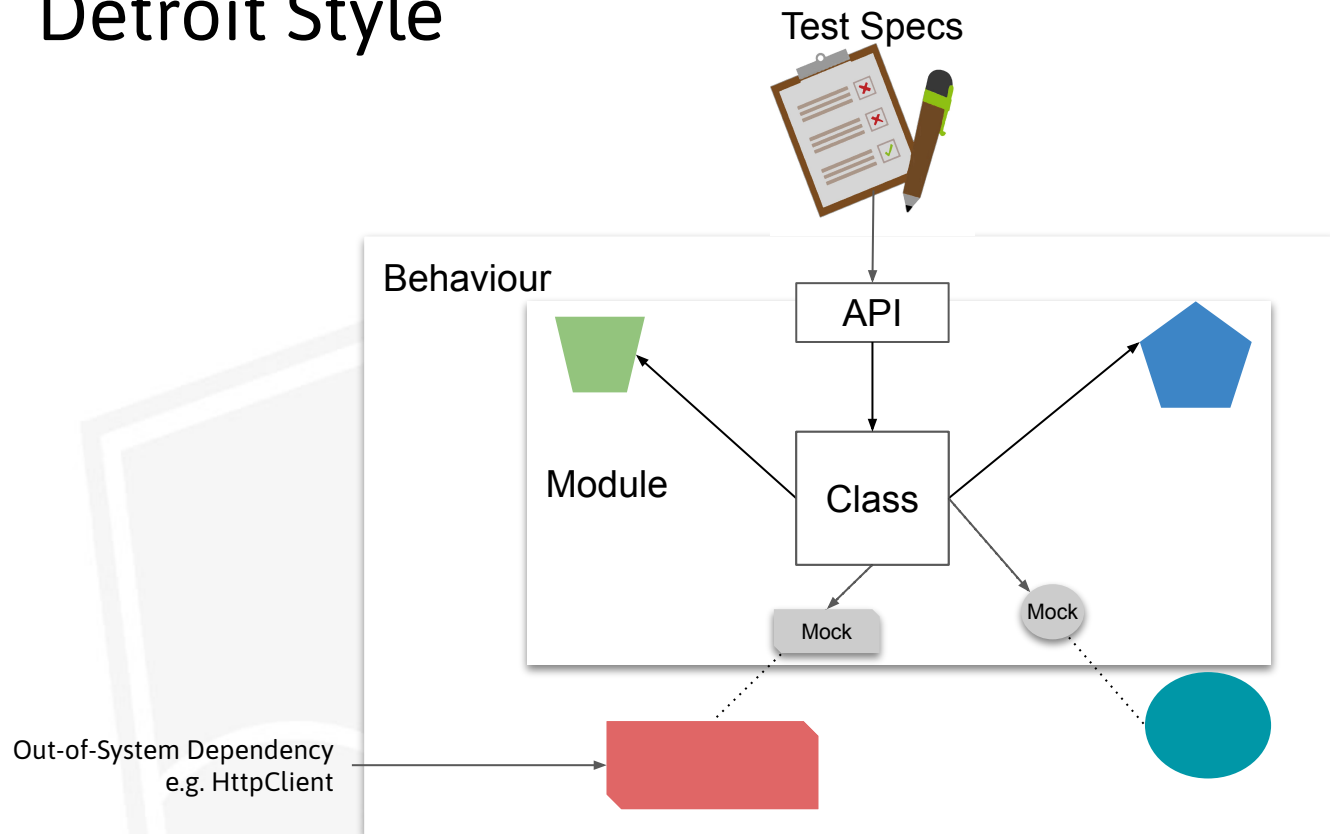


Out-of-System Dependency
e.g. HttpClient



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Detroit Style



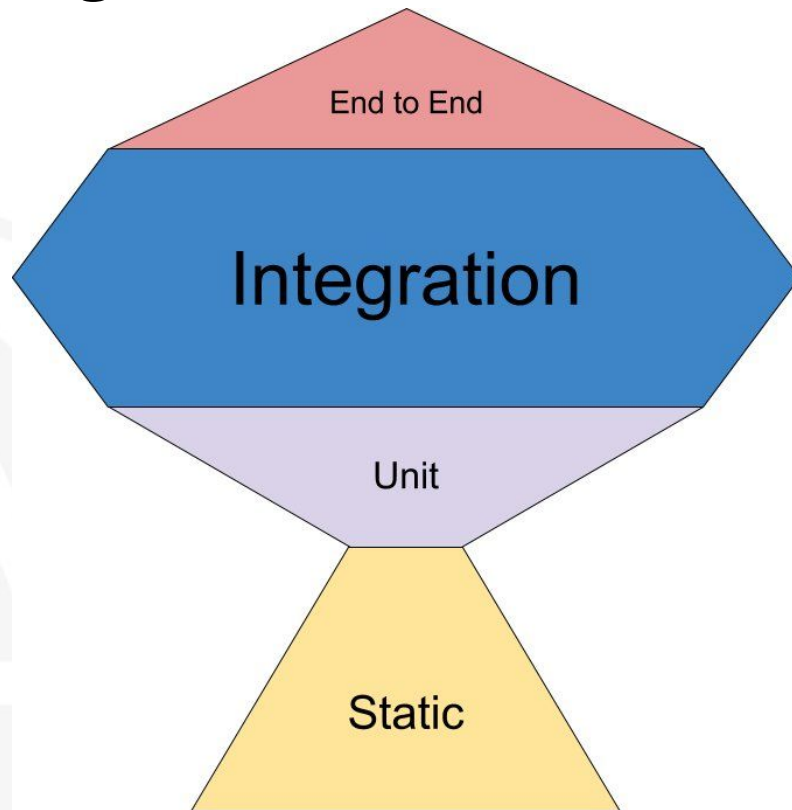
Criteria

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Testing According to ROI



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

4: UI Library

- Library Vendor
- Storybook
 - Visual Regression
 - Cypress



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

...



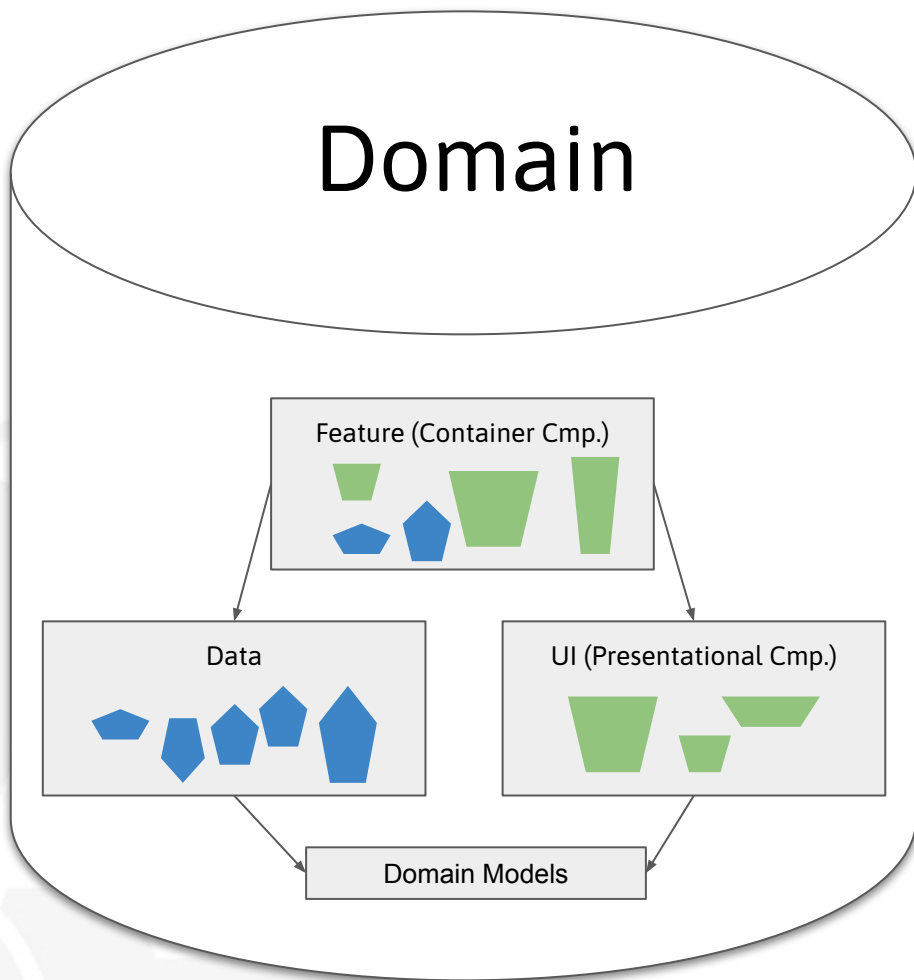
Component



Service



Module



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

