Case Study: FluentMapper

Floyd May
SOFTWARE CRAFTSMAN

@softwarefloyd https://medium.com/@floyd.may

Subtle Differences Are Hard to Spot

```
var target = new AddressDTO
{
    Address1 = source.Street1,
    Address2 = source.Street2,
    City = source.City,
    State = source.State,
    Country = source.Country,
    Zip = source.PostalCode
};
```

Our Target API

```
var mapper = FluentMapper
    .ThatMaps<AddressDTO>().From<Address>()
        .ThatSets(tgt => tgt.Address1).From(src => src.Street1)
        .ThatSets(tgt => tgt.Address2).From(src => src.Street2)
        .ThatSets(tgt => tgt.Zip).From(src => src.PostalCode)
        .IgnoringSourceProperty(x => x.Notes)
    .Create();
```

Agenda

Map types with matching property names and types

Define mappings between properties with different names

Ignore properties

Null source behavior

Beginning API

FluentMapper

TargetTypeSpec<TTgt>

TypeMappingSpec<TTgt, TSrc>

IMapper<TTgt, TSrc>

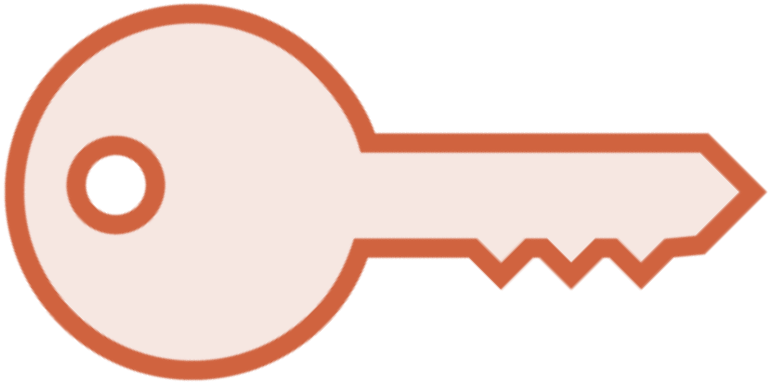
Use Conventions for High-level Tests

Embrace the relationship between tests, documentation, and feedback

Cohesive groups of tests related to a single feature or concept

Patterns for communicating

Throw exceptions for unmapped properties



FluentMapper
Key Feature

FluentMapper: Unmapped Properties

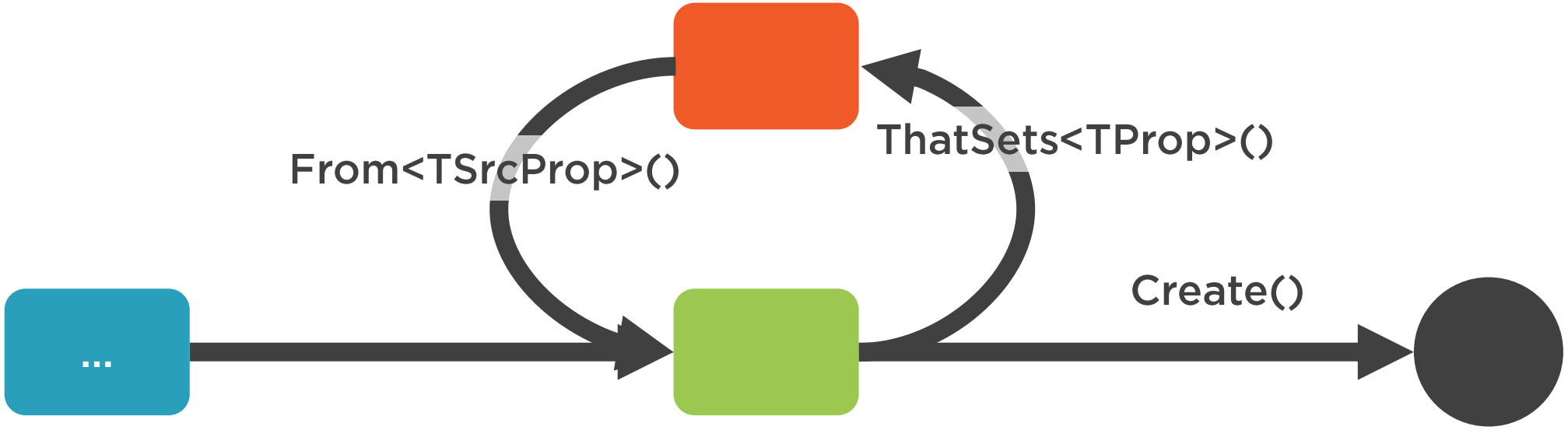
Classes are maintained over time

- New properties
- Deleted properties

Put FluentMapper client code under test

Manual mapping code won't detect all cases of new or deleted properties

Context Arc



TypeMappingSpec<TTgt, TSrc>

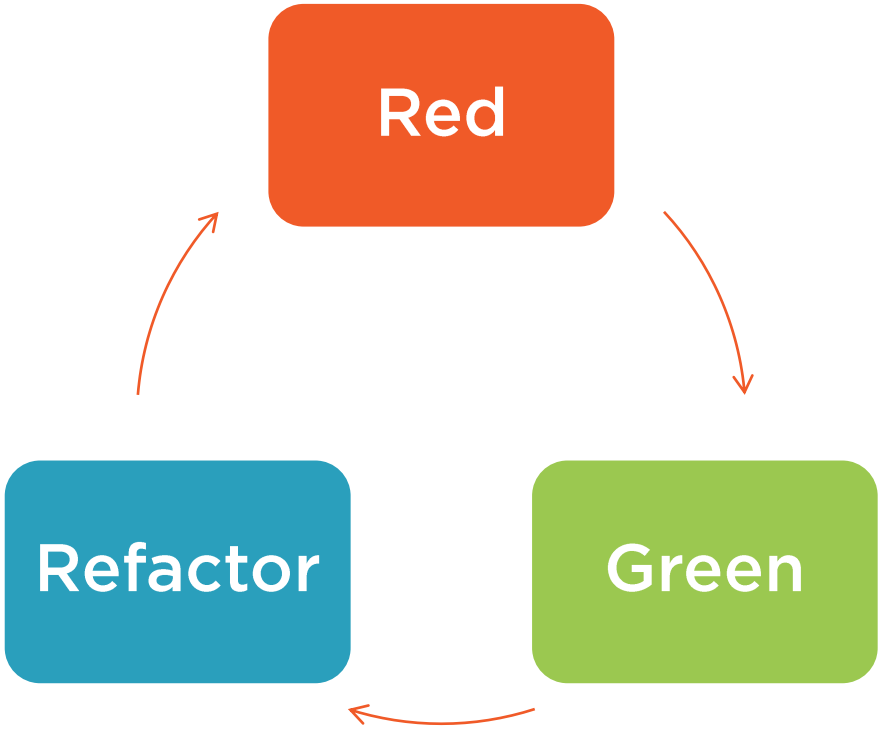
SetterSpec<TTgt, TSrc, TProp>

IMapper<TTgt, TSrc>

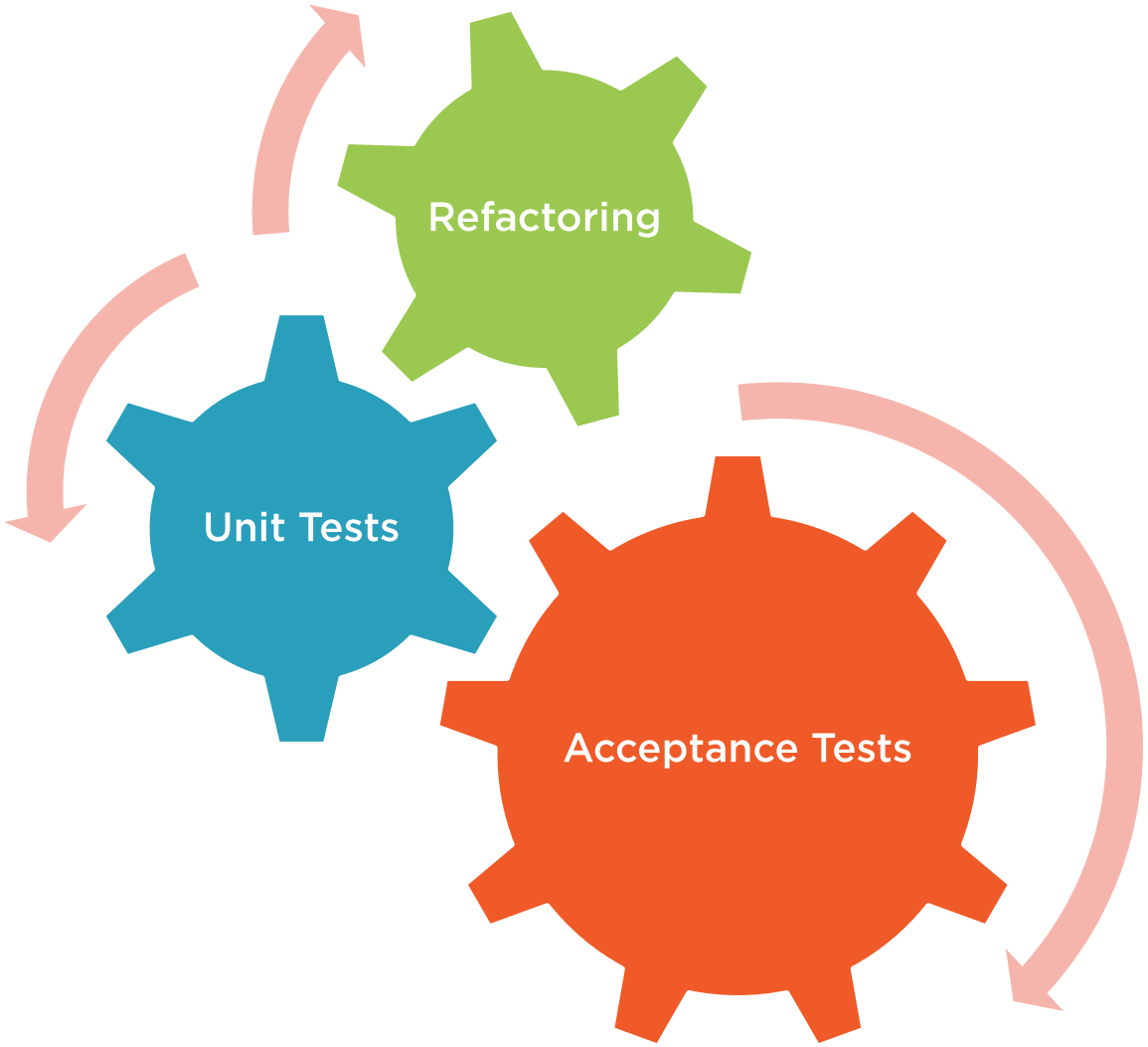
Context Arc

A path from one context type to another that then returns back to the original context type.

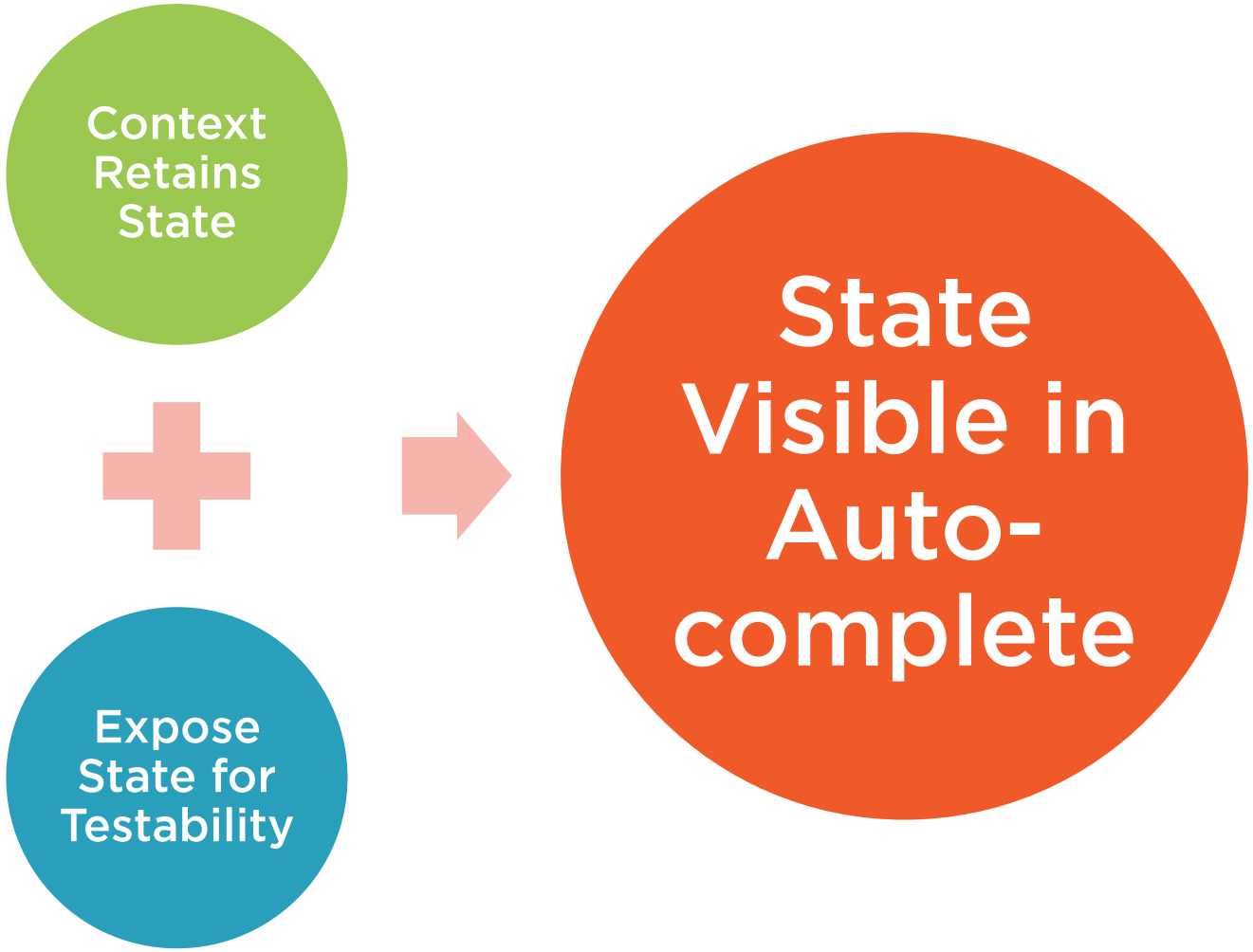
The TDD Cycle



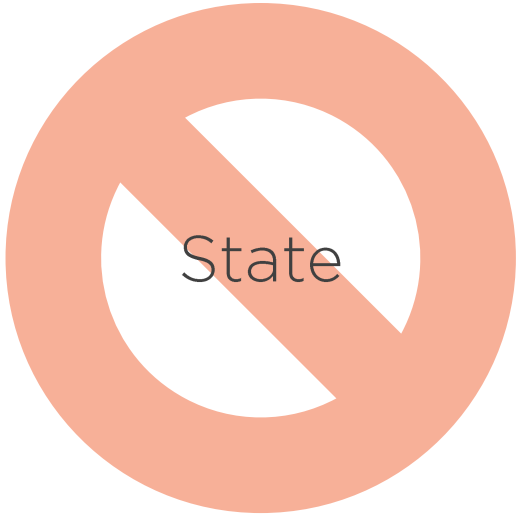
Acceptance Test Driven Development



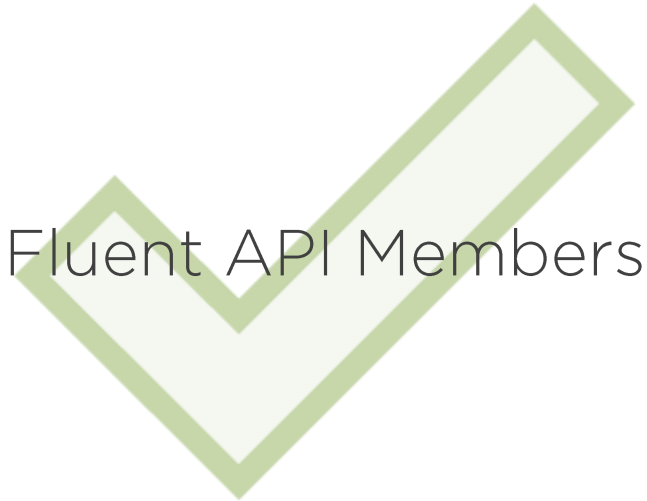
Test-Driven Design and Fluency



Public Members in Context Classes



- **Great for testability**
- **Bad for fluency**



- **Lead the user through the design**
- **Harness IDE auto-completion**

```
public class SomeFluentContext
{
  public IEnumerable<string> SomeState { get; }
}
```

Avoid Public State Properties in Context Classes

Public properties are visible in Intellisense

We need a better way to expose state for testability

```
public class SomeFluentContext : IContextProperties
{
  IEnumerable<string> IContextProperties.SomeState { get; }
}
```

Use Explicit Interface Implementation

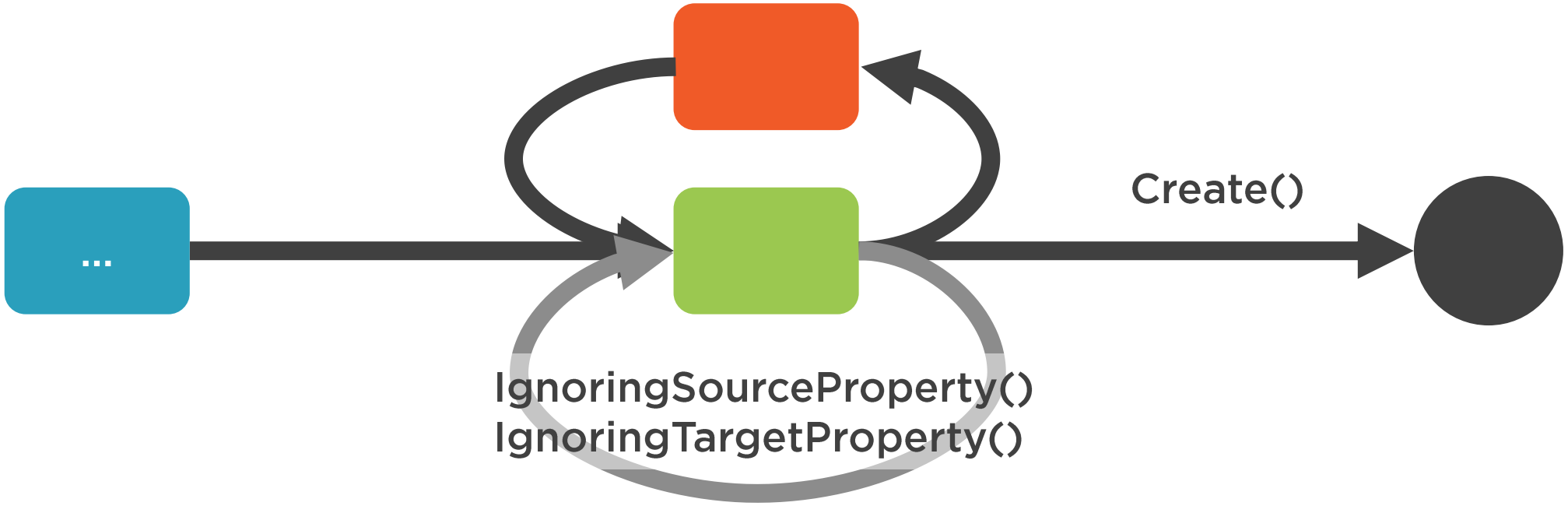
State is available for testability

Properties are hidden from IDE auto-completion

```
FluentMapper.ThatMaps<Target>()
    .From<Source>()
    .ThatSets(tgt => tgt.Name)
        .From(src => src.Text)
    .Create();
```

◀ Action<TTgt, TSrc>

Ignoring Properties

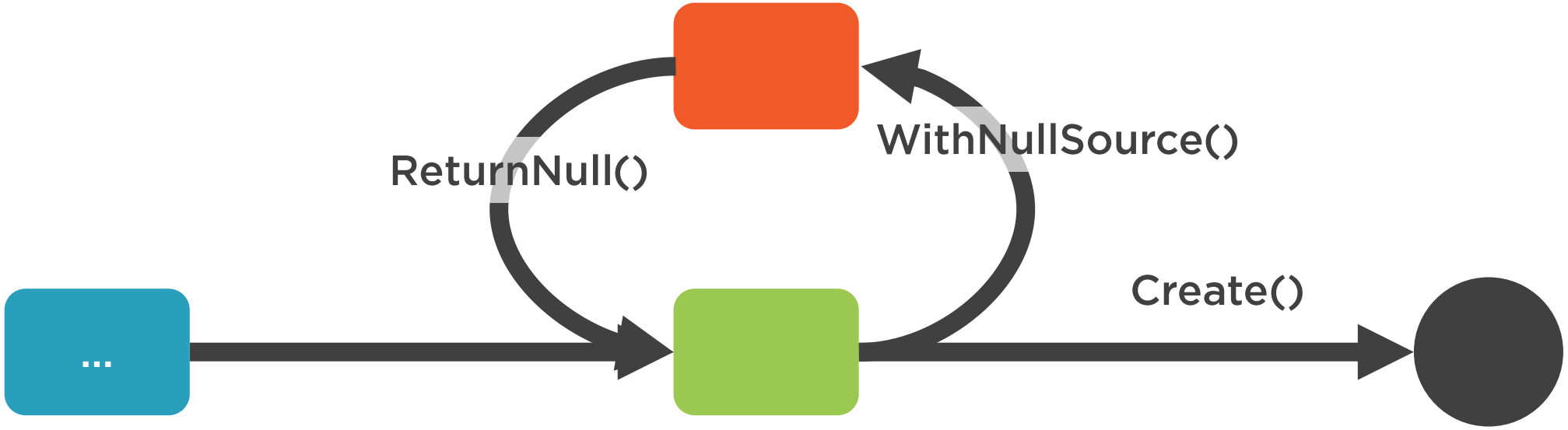


TypeMappingSpec<TTgt, TSrc>

SetterSpec<TTgt, TSrc, TProp>

IMapper<TTgt, TSrc>

Null Source Behavior



TypeMappingSpec<TTgt, TSrc>

NullSourceBehavior<TTgt, TSrc>

IMapper<TTgt, TSrc>

Embrace Immutability

Readable

Testable

Extensible

Wrapping Up

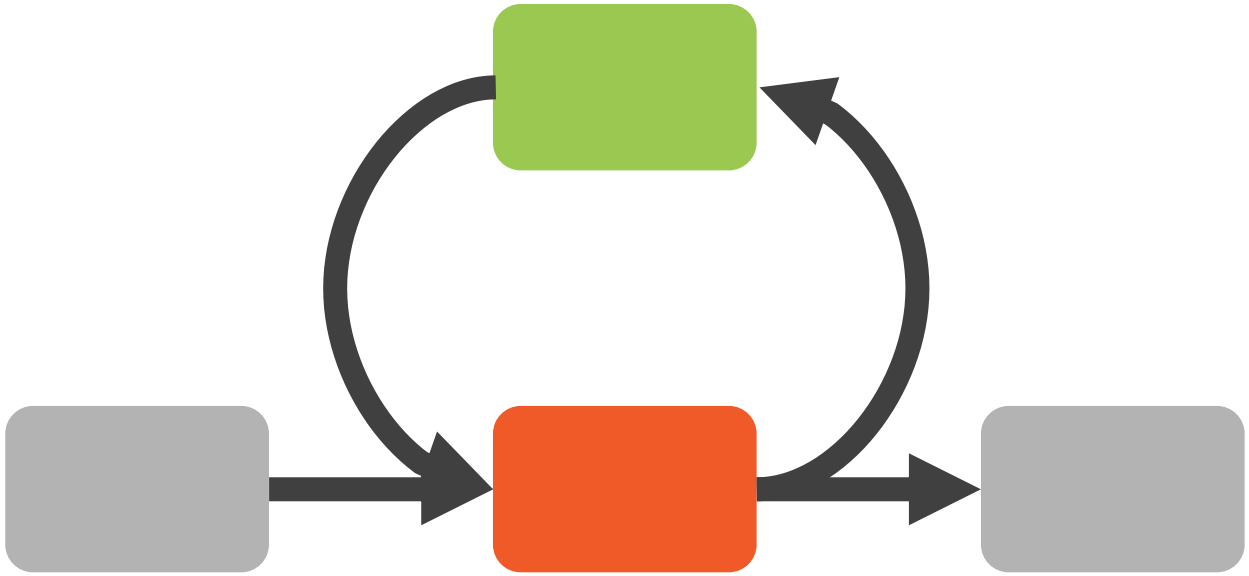
Acceptance tests are central to fluent API design

Break down complex acceptance tests into fine-grained unit tests

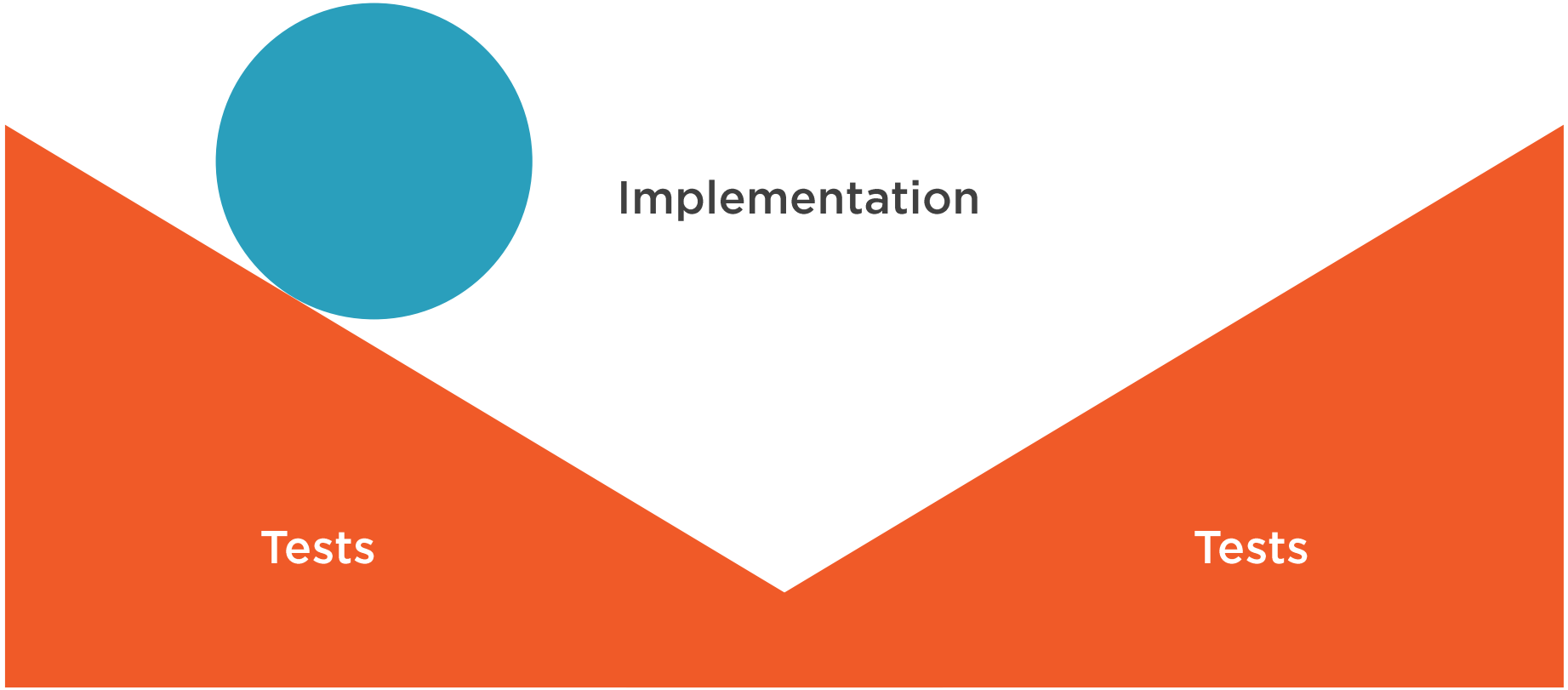
Take time to refactor

Keep your code clean

Context Arc



Essential Design Elements



Keep Public APIs Clean

Use explicit interface implementation to hide irrelevant implementation details.

Parting Thoughts

Test first

Embrace collaboration

Incorporate feedback into tests & documentation

Leverage vocabulary & patterns

Illustrate with context graphs

Thanks For Watching!