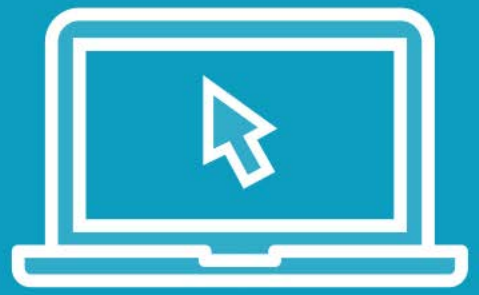
Designing Fluent APIs in C#

UNDERSTANDING FLUENT APIs – AN INTRODUCTION

Floyd May
SOFTWARE CRAFTSMAN
@softwarefloyd https://medium.com/@floyd.may

Demo



Examples of fluent APIs

Fluent APIs emphasize the intent of your code

Fluent APIs enable you to write less code

Fluent APIs express complex concepts succinctly

Intermediate C# skills

- Inheritance
- Generics
- Unit testing

You build code for others to use

You can talk to your users

- Face-to-face
  or
- Digital

Desire for creative elegant solutions

What We'll Cover

Fundamentals

- Vocabulary
- Definitions
- Patterns

Demonstrations

- Seeman's INotifyPropertyChanged API
- FluentMapper
- Test-driven approach

Design Process

- Language arts
- Collaboration

Domain-Specific Languages (DSLs)

Targeted for a specific purpose

Limited scope

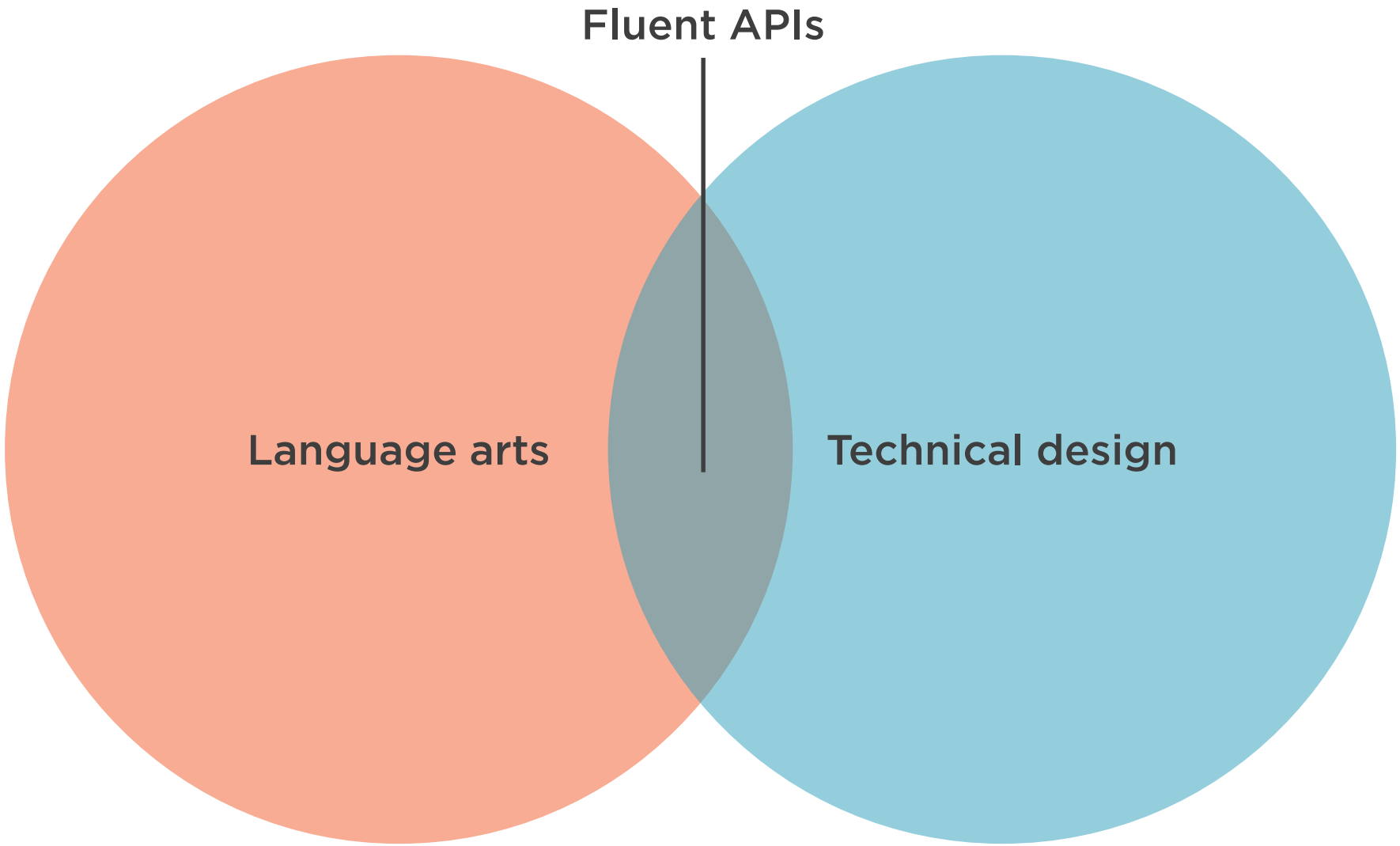
Solves a particular problem domain *really well*

Embedded DSL

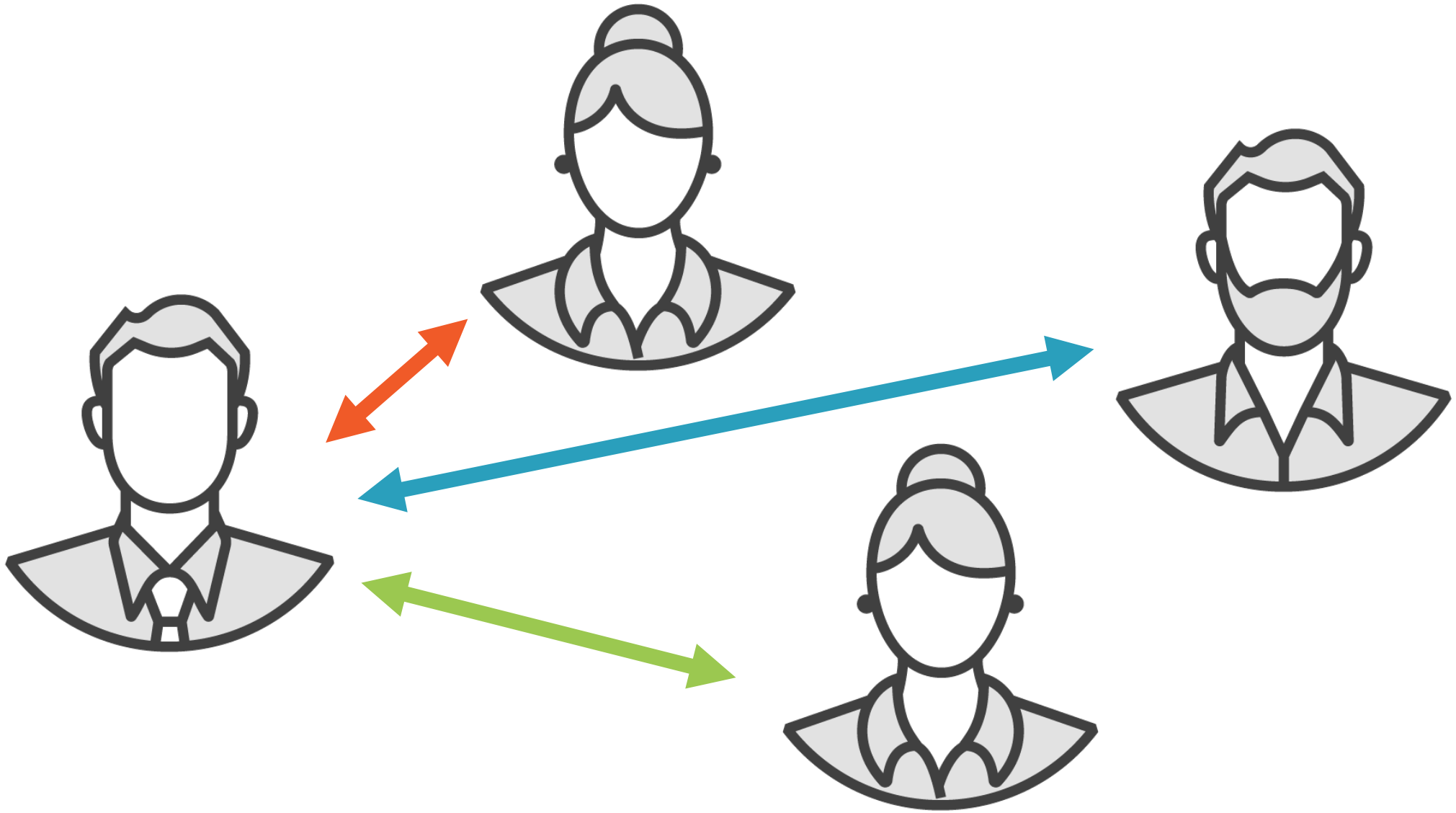
A domain-specific language (DSL) that is expressed using the syntax and semantics of a *host language*.

Targeting the problem domain

An Important Combination



Fluent API Design Is Social



```
testee.ShouldNotNotifyFor(x => x.LastName)
    .And(x => x.AnotherProperty)
    .When(x => x.FirstName = "Something");
```

A Feature Request – Negative Testing

I'd like to verify that property change notifications do *not* occur when a particular action is taken.

```
testee.ShouldNotifyFor(x => x.FirstName)
    .And(x => x.FullName)
    .ButNot(x => x.PhoneNumber)
    .And(x => x.EmailAddress)
    .When(x => x.FirstName = "Something");
```

An Observation – Mix-and-match Is Unclear

Using “And” to express both positive and negative notifications is confusing.

```
testee.ShouldNotifyFor(x => x.FirstName)
      .And(x => x.FullName)
   .ButNot(x => x.PhoneNumber)
      .Nor(x => x.EmailAddress)
   .When(x => x.FirstName = "Something");
```

A Clearer Expression – “And” vs. “Nor”

Using two different conjunctions, “and” and “nor”, improves the clarity.

Establishing best-practice indentation also adds clarity.

```
testee.ShouldNotifyFor(x => x.FirstName)
      .And(x => x.FullName)
      .Nor(x => x.EmailAddress) // syntax error
   .When(x => x.FirstName = "Something");
```

Tightening the Rules

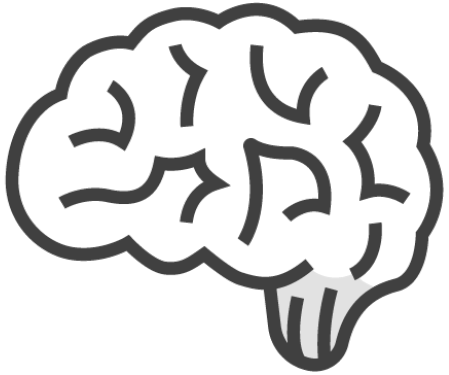
Don’t allow mixing “and” and “nor”.

Specify all positive tests, then all negative tests.

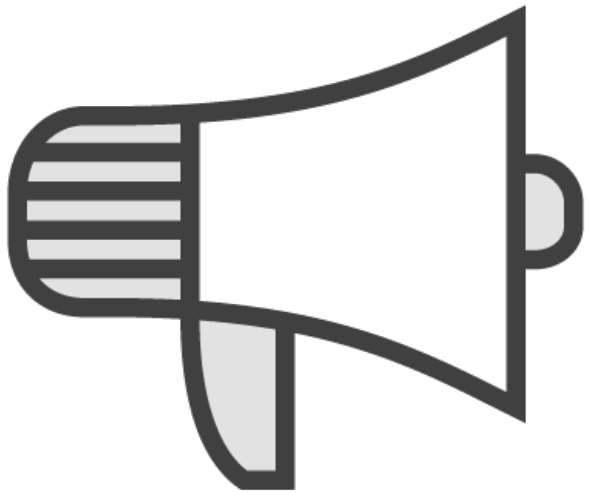
Design Guidelines

Embrace collaboration

Consider “what if?” scenarios

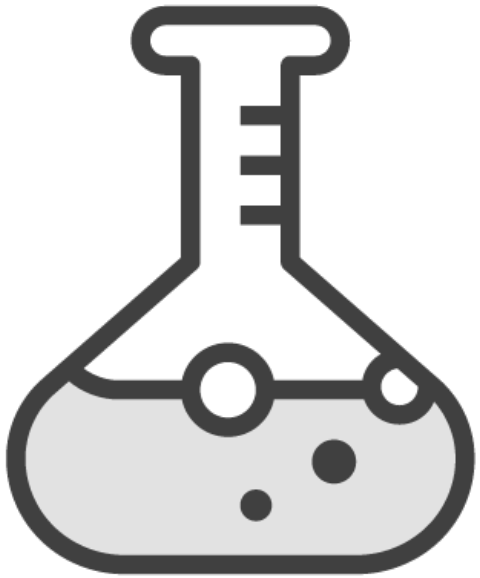


Apply language skills



User feedback reflects real-world desires and expectations

Keep those desires and expectations safe



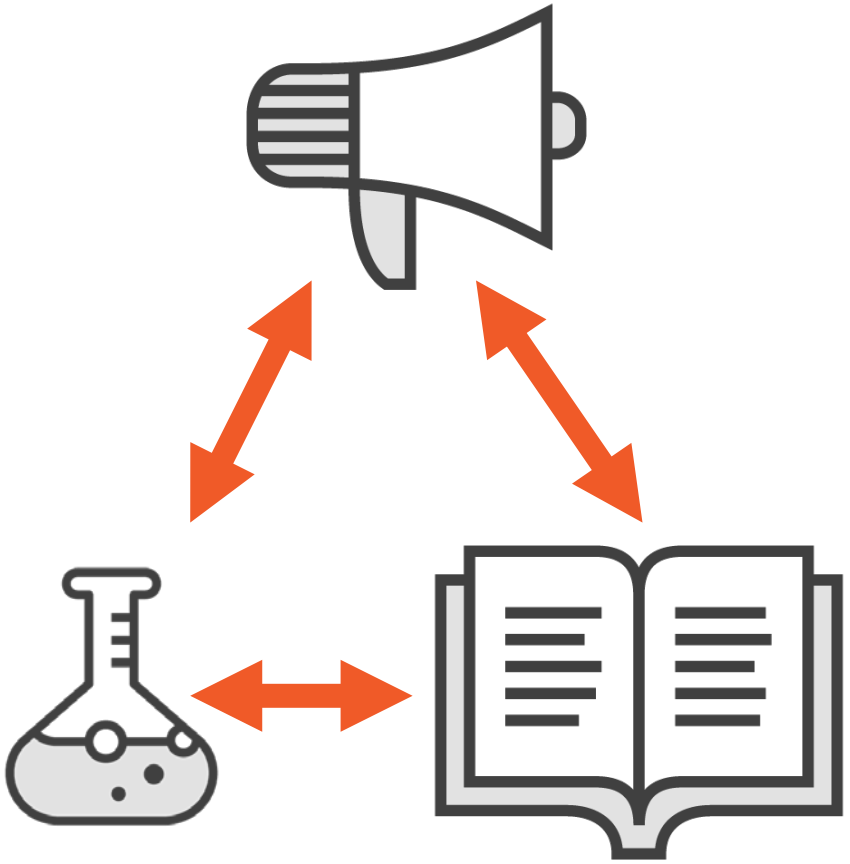
Encode user feedback into tests

Protect API behavior over time

Living documentation – never out of date

Fine-grained information

API documentation is *code*



User feedback drives tests and documentation

The same code samples in all three places

Recap: Benefits of Fluent APIs

Emphasizes the intent of the code

Enables you to write less code

Expresses complex concepts succinctly

Recap: Domain-specific Languages (DSLs)

Targeted at a specific purpose

Limited in scope

Solves a specific type of problem
really well

Recap: Embedded DSLs

Expressed within the syntax of a *host language* (C#)

No compiler writing

Recap: Two Disciplines

Language arts

Technical design

Recap: Collaboration

Creativity

Constructive criticism

Iterating and refining drafts

Recap: Three for One

User feedback

Acceptance tests

Documentation