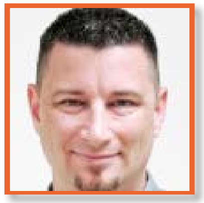
Specification Pattern

Steve Smith
Ardalis.com
@ardalis

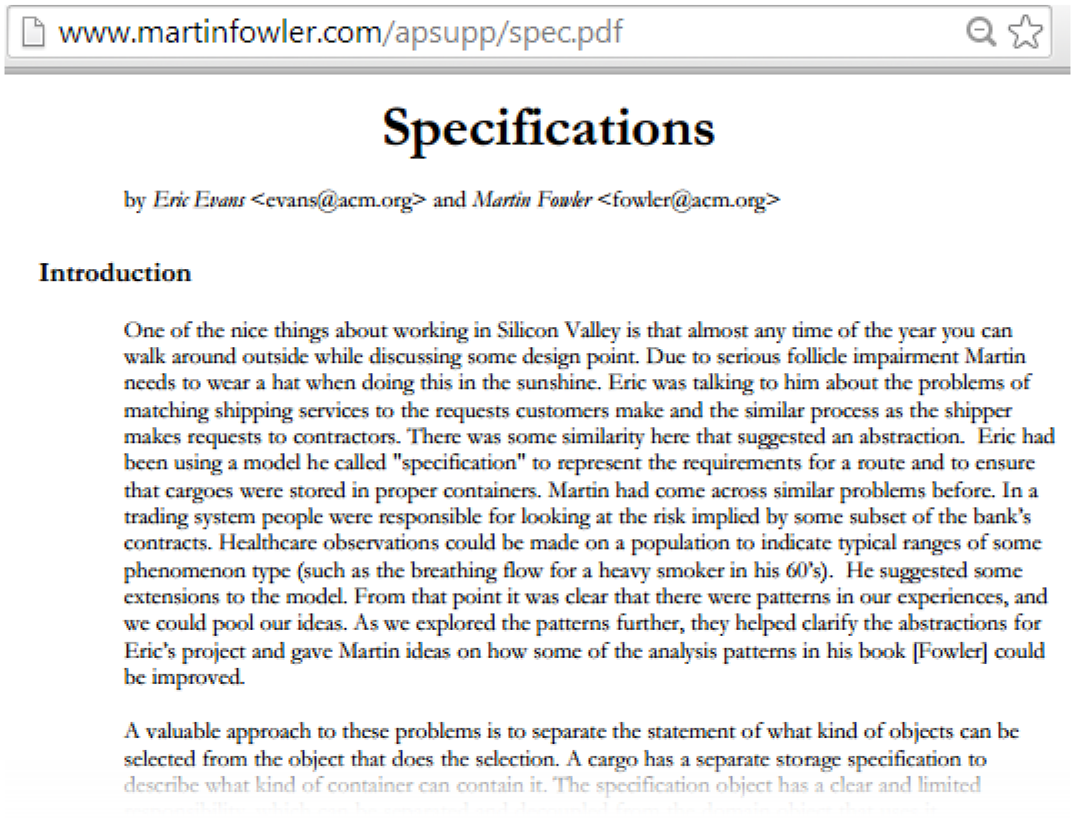


pluralsight
hardcore dev and IT training

Overview

- Origin
- Motivating Examples
- Intent
- Implementation
- Benefits
- Common Uses
- Specifications and Repositories
- Specifications and Entity Framework

Domain-Driven Design



www.martinfowler.com/apsupp/spec.pdf

Specifications

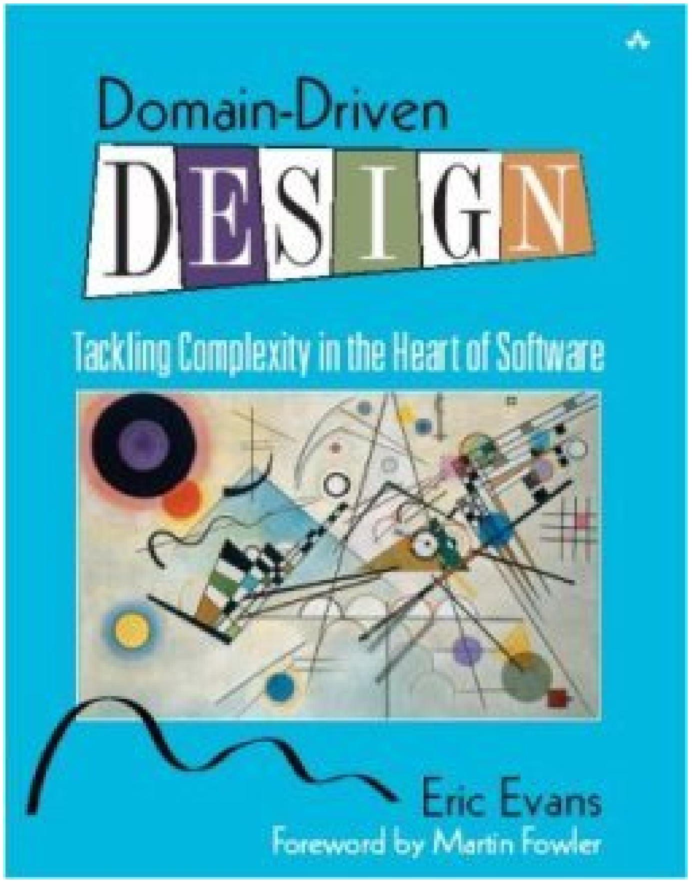
by *Eric Evans* <evans@acm.org> and *Martin Fowler* <fowler@acm.org>

Introduction

One of the nice things about working in Silicon Valley is that almost any time of the year you can walk around outside while discussing some design point. Due to serious follicle impairment Martin needs to wear a hat when doing this in the sunshine. Eric was talking to him about the problems of matching shipping services to the requests customers make and the similar process as the shipper makes requests to contractors. There was some similarity here that suggested an abstraction. Eric had been using a model he called "specification" to represent the requirements for a route and to ensure that cargoes were stored in proper containers. Martin had come across similar problems before. In a trading system people were responsible for looking at the risk implied by some subset of the bank's contracts. Healthcare observations could be made on a population to indicate typical ranges of some phenomenon type (such as the breathing flow for a heavy smoker in his 60's). He suggested some extensions to the model. From that point it was clear that there were patterns in our experiences, and we could pool our ideas. As we explored the patterns further, they helped clarify the abstractions for Eric's project and gave Martin ideas on how some of the analysis patterns in his book [Fowler] could be improved.

A valuable approach to these problems is to separate the statement of what kind of objects can be selected from the object that does the selection. A cargo has a separate storage specification to describe what kind of container can contain it. The specification object has a clear and limited

2002


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

2003

Intent

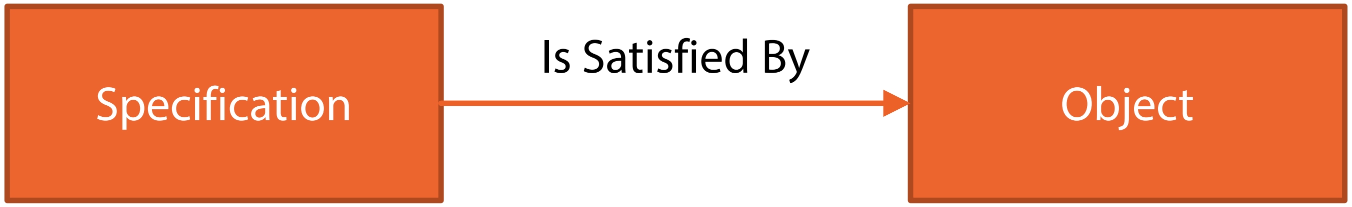
- *Specify*, in an object, work to be done
- Separate the *specification* of the work from performing the work itself

Candidate Problems

- **Queries**
  - Specify criteria for selection
  - May be applied to data, as well as object capabilities

- **Persisting or Transporting Criteria**
  - Saved searches or “smart” lists
  - Object creation instructions or requirements for a Factory

Implementation



The Specification object should be implemented as a Value Object

(Value Objects are discussed in *Domain-Driven Design Fundamentals*)

Boolean Rules on Entities

Avoid cluttering Entities with complex rules and rule evaluation logic

```
public bool ShouldIncludeInSmartList()
{
    return ((Rating.HasValue && Rating.Value >= 4)
            && (DateTime.Now.Year - this.Year > 5)
            && (Length > TimeSpan.FromMinutes(1))
            && (Length < TimeSpan.FromMinutes(5)));
}
```

Predicates

- **Rules that evaluate to a Boolean can be modeled as *predicates***
- **Rules can use Func<T,bool> or Expression<Func<T,bool>>**
- **Example (for a Song): s => s.Rating > 3**
- **Work best for simple, small conditions**

Where does complex query logic belong?

- **Rules may involve multiple Entities or Value Objects**
  - Or may clutter such types

- **Moving logic outside of domain model reduces value of the model**

- **Working with complex predicates is not ideal**
  - Doesn’t communicate intent
  - Difficult to test
  - Often not primary focus of code in which they reside

Enter the Specification

A *Specification*

- States a constraint on the state of another object (which may or may not be present)
- Can test any object (of the appropriate type) to see if it satisfies certain criteria
- Is modeled as a Value Object

Common Uses

- **Validation**
  - Is it ready?
  - Is it suitable for a certain set of requirements?
- **Selection from a collection**
  - Provide constraints as part of a query
  - Frequently combined with the Repository pattern
- **Specify the creation of an object to suit a certain need**
  - May be combined with the Factory pattern

Validation Example

```
public class CourseReadyToPublishSpecification : ISpecification<Course>
{
    1 reference
    public bool IsSatisfiedBy(Course course)
    {
        if (course.Modules.Count == 0) return false;
        if (course.AuthorContracts.Count == 0) return false;
        if (!course.PublicationDate.HasValue) return false;
        if (IsNullOrEmpty(course.Description)) return false;
        if (course.Modules.Any(m => IsNullOrEmpty(m.SlideUrl))) return false;
        if (course.Modules.Any(m => IsNullOrEmpty(m.MaterialsUrl))) return false;
        if (course.AuthorContracts.Any(c => !c.Signed)) return false;
        return true;
    }
}
```

```
    public interface ISpecification<T>
    {
        1 reference
        bool IsSatisfiedBy(T target);
    }
```

Specifications and Data

```
public IEnumerable<Course> List(ISpecification<Course> spec)
{
    return _courses
        .Where(c => spec.IsSatisfiedBy(c))
        .AsEnumerable();
}
```

```
public IEnumerable<Course> List(ISpecification<Course> spec)
{
    return _courses.ToList() // convert to in-memory list
        .Where(c => spec.IsSatisfiedBy(c))
        .AsEnumerable();
}
```

Specifications with Predicates

```
public interface ISpec<T>
{
    1 reference
    Expression<Func<T,bool>> CriteriaExpression { get; }
}
```

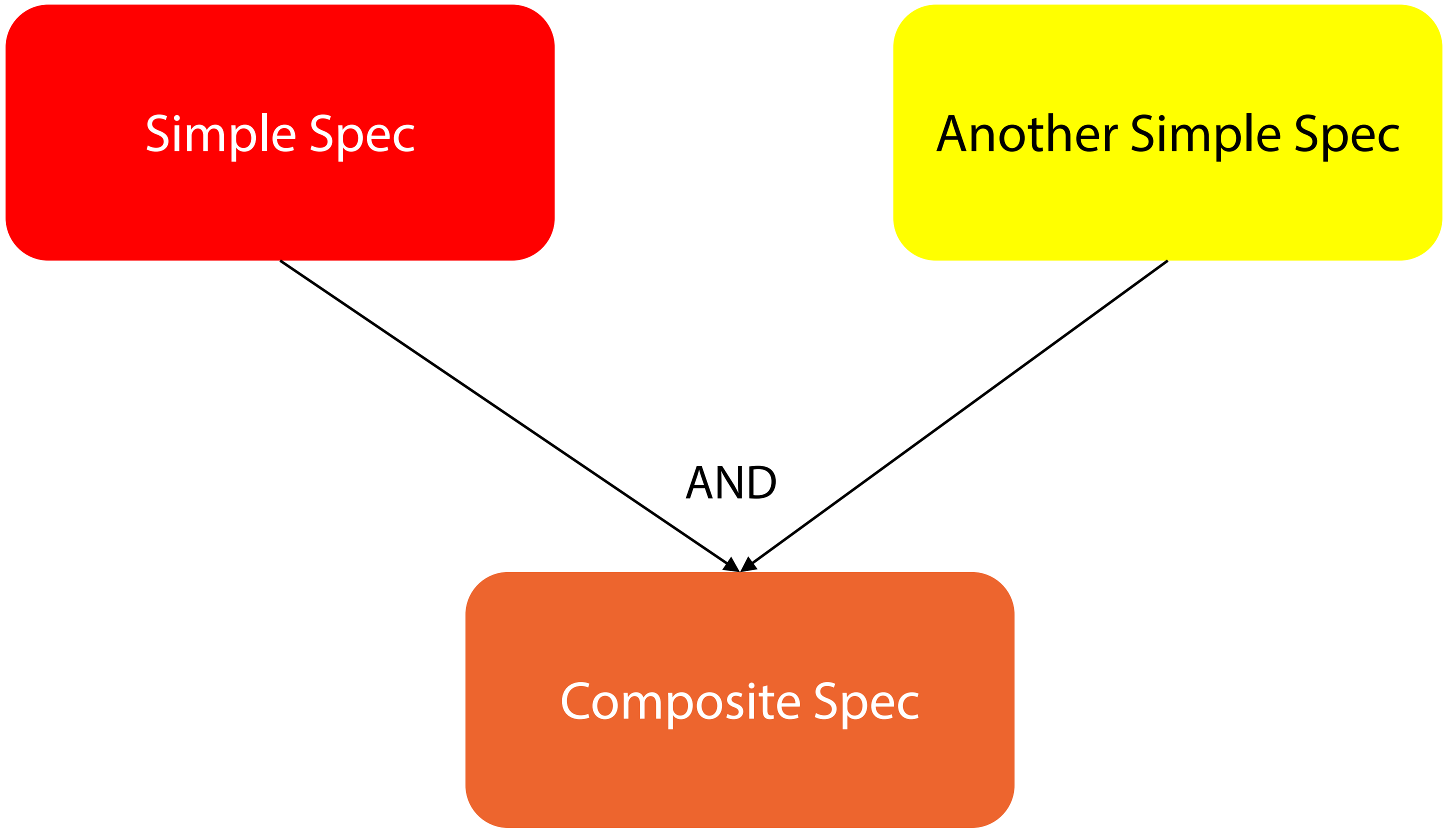
```
public Expression<Func<Course, bool>> CriteriaExpression
{
    get
    {
        return c => (c.Modules.Count > 0)
                    && (c.AuthorContracts.Count > 0)
                    && c.PublicationDate.HasValue
                    && !IsNullOrEmpty(c.Description)
                    && c.Modules.All(m => !IsNullOrEmpty(m.SlideUrl))
                    && c.Modules.All(m => !IsNullOrEmpty(m.MaterialsUrl))
                    && c.AuthorContracts.All(ac => ac.Signed);
    }
}
```

Predicates and Data

```
public IEnumerable<Course> List2(Expression<Func<Course, bool>> predicateExpression)
{
    return _courses
        .Where(predicateExpression)
        .AsEnumerable();
}
```

```
public IEnumerable<Course> List2(ISpec<Course> spec)
{
    return _courses
        .Where(spec.CriteriaExpression)
        .AsEnumerable();
}
```

Combining Predicates and Specifications



Building Objects To Order

- From Scratch
- From existing object

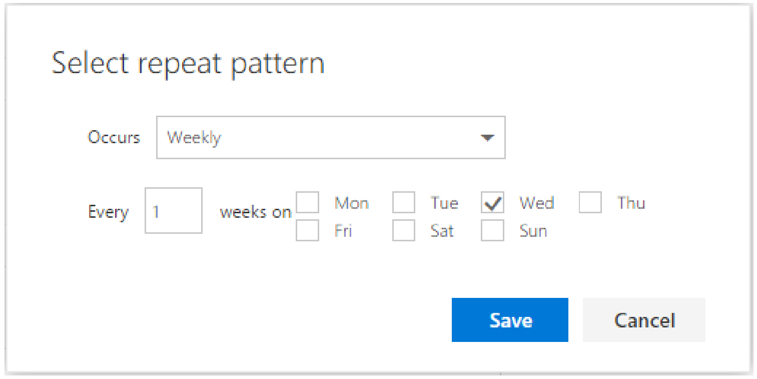
Example:

A route consists of several segments. Routes from one point to another can be created from scratch as a series of segments. Alternately, if a segment becomes unavailable, a new detouring Route can be created from a partially completed one.

Frequently combined with a Factory design pattern

Real World Examples

- “Smart” Playlists
- Cargo containers equipped for special cargo (e.g. refrigerated)
- Routes
- Medical populations
- Stock portfolios
- Sales regions
- Event recurrence (e.g. Outlook repeating events)



Select repeat pattern

Occurs: Weekly

Every 1 weeks on: [ ] Mon [ ] Tue [✓ Wed] [ ] Thu [ ] Fri [ ] Sat [ ] Sun

Specification Benefits

- Separation of Concerns
- Decoupling
- Easier to Test in Isolation
- More expressive and declarative

Demo
Creating Music Playlists

References

- Open-Closed Principle – SOLID Principles of Object Oriented Design (Pluralsight)
  - http://bit.ly/solid-smith
- Domain-Driven Design Fundamentals (Pluralsight)
  - http://bit.ly/PS-DDD
- Repository Pattern
  - http://app.pluralsight.com/courses/patterns-library
- Factory Pattern
  - http://app.pluralsight.com/courses/patterns-library

Summary

- ***Specification*** **is generally used for validation, filtering, or build-to-order scenarios**
- **A specification defines a method or predicate that matches qualifying objects**
- **The specification pattern can help separate concerns and decouple code**
- **Specifications can be more expressive than other approaches**