Functional Programming with C#

What Is Functional Programming?

Dave Fancher

@davefancher | davefancher.com

Benefits of Functional Programming

Greater Predictability

Easier Extensibility

Improved Testability

Agenda

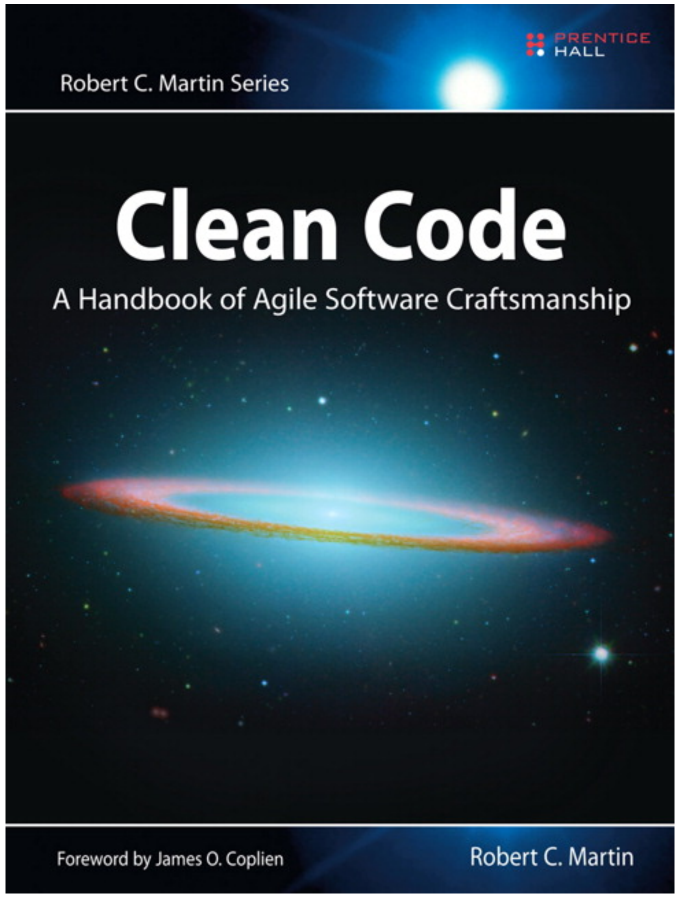
- What is Functional Programming?
- Express Yourself
- Functional Thinking
- Going with the Flow

No matter what language you work in, programming in a functional style provides benefits. You should do it whenever it is convenient, and you should think hard about the decision when it isn't convenient.

— John Carmack
http://ubm.io/1HOVGWs

Clean Code

- Keep functions small
- Don't repeat yourself
- Do one thing
- Avoid side-effects
- Functions should accept no more than 3 parameters


PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

Are they really so different?

OO makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts.

— Michael Feathers

http://bit.ly/1HOVBSM

What is functional programming?

Functional programming is…

…a paradigm which concentrates on computing results rather than on performing actions.

http://bit.ly/1gONrnw

1

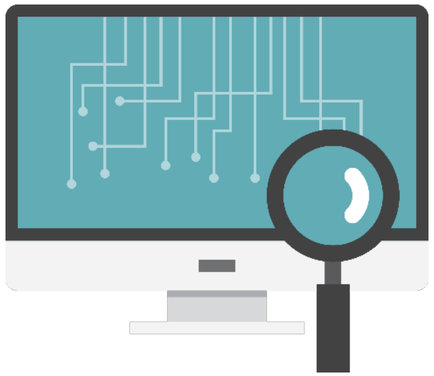
Tamed Side Effects

A side effect is…

1. A secondary, typically undesirable effect of a drug or medical treatment
2. Any accompanying or consequential and usually detrimental effect

http://bit.ly/1WbKEoh

Why are side effects bad?



Example: GetOrderItems Method

Functional Purity

- Purely Functional
- Impure

C# is Impure

2 Expression-Based

Statements vs Expressions

Statements

- Define actions
- Executed for their side-effect

```
string posOrNeg;

if (value > 0)
    posOrNeg = "positive";
else
    posOrNeg = "negative";
```

Expressions

- Produce results
- Executed for their result

```
var posOrNeg =
    value > 0
        ? "positive"
        : "negative";
```

Expression Composition

```
string posOrNeg;

if (value > 0)
    posOrNeg = "positive";
else
    posOrNeg = "negative";

var msg = $"{value} is {posOrNeg}";
```

Composability

Statements

```
var msg =
    $"{value} is {(value > 0 ? "positive" : "negative")}";
```

Composability

Expressions

3

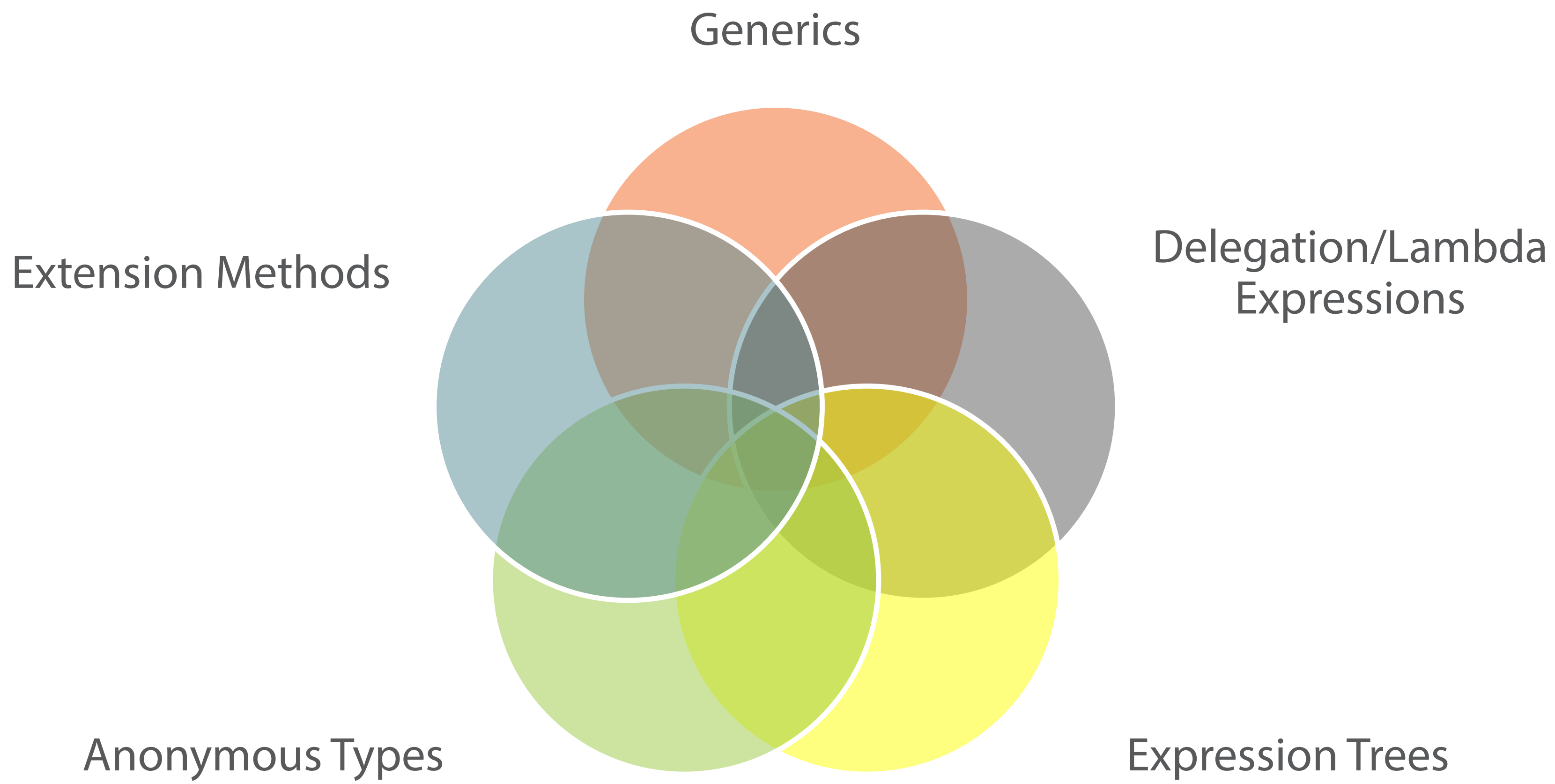
Treat Functions as Data

Higher-Order Functions

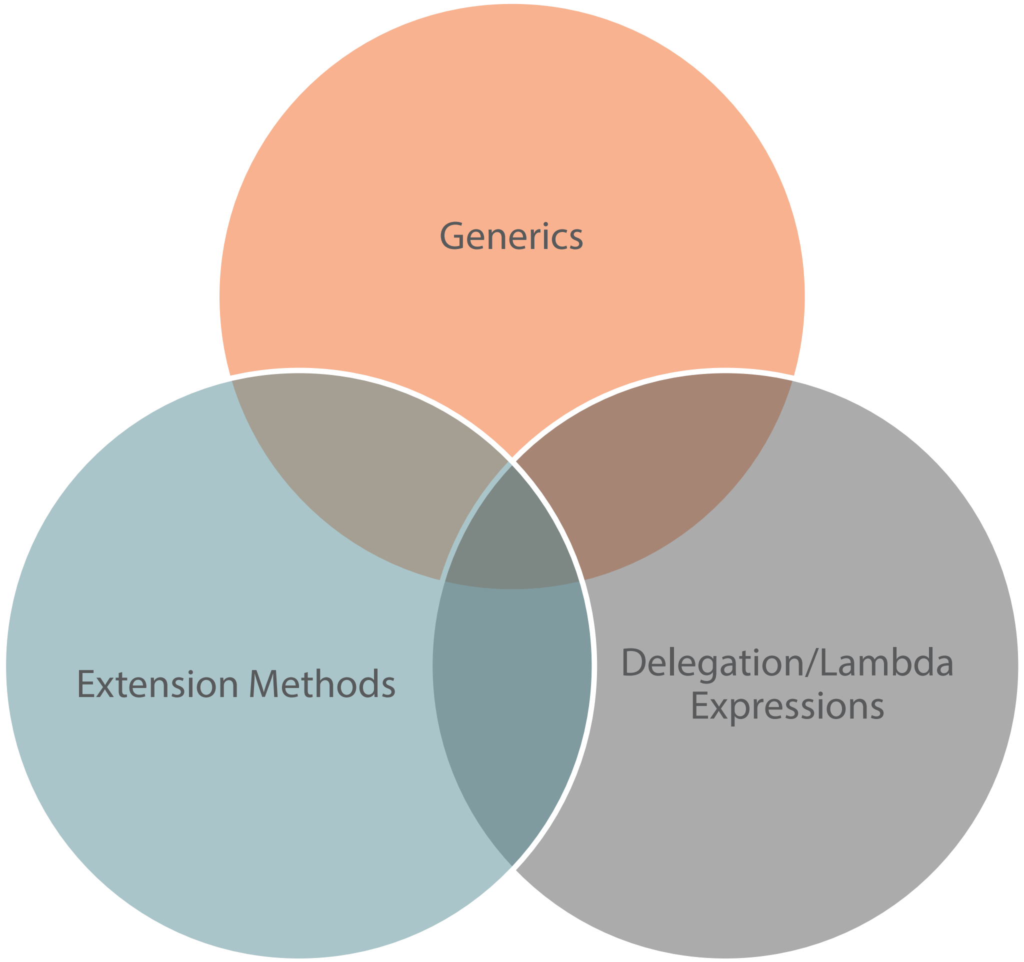
- Functions which accept other functions
- Functions which return functions

LINQ is built on functional principles

What Is LINQ?


Generics
Extension Methods
Delegation/Lambda Expressions
Anonymous Types
Expression Trees

LINQ to Objects



Filtering & Sorting

The Imperative Way

```
var ix = 0;
while (ix < myList.Count)
{
    if (myList[ix] % 2 != 0)
    {
        myList.RemoveAt(ix);
    }
    else
    {
        ++ix;
    }
}

myList.Sort();
```

```
from x in myList
where x % 2 == 0
orderby x
select x;
```

Filtering & Sorting

The LINQ Query Syntax Way

```
myList
    .Where(x => x % 2 == 0)
    .OrderBy(x => x);
```

Filtering & Sorting

The LINQ Method Syntax Way

In Review

Taming side effects

Emphasizing expressions

Treating functions as data