Functional programming in C++

Alex Bolboacă, @alexboly, alex.bolboaca@mozaicworks.com

February 2018

Intro

A practical view on functional programming

Mindset

Core Building Blocks

Improve design with functional programming

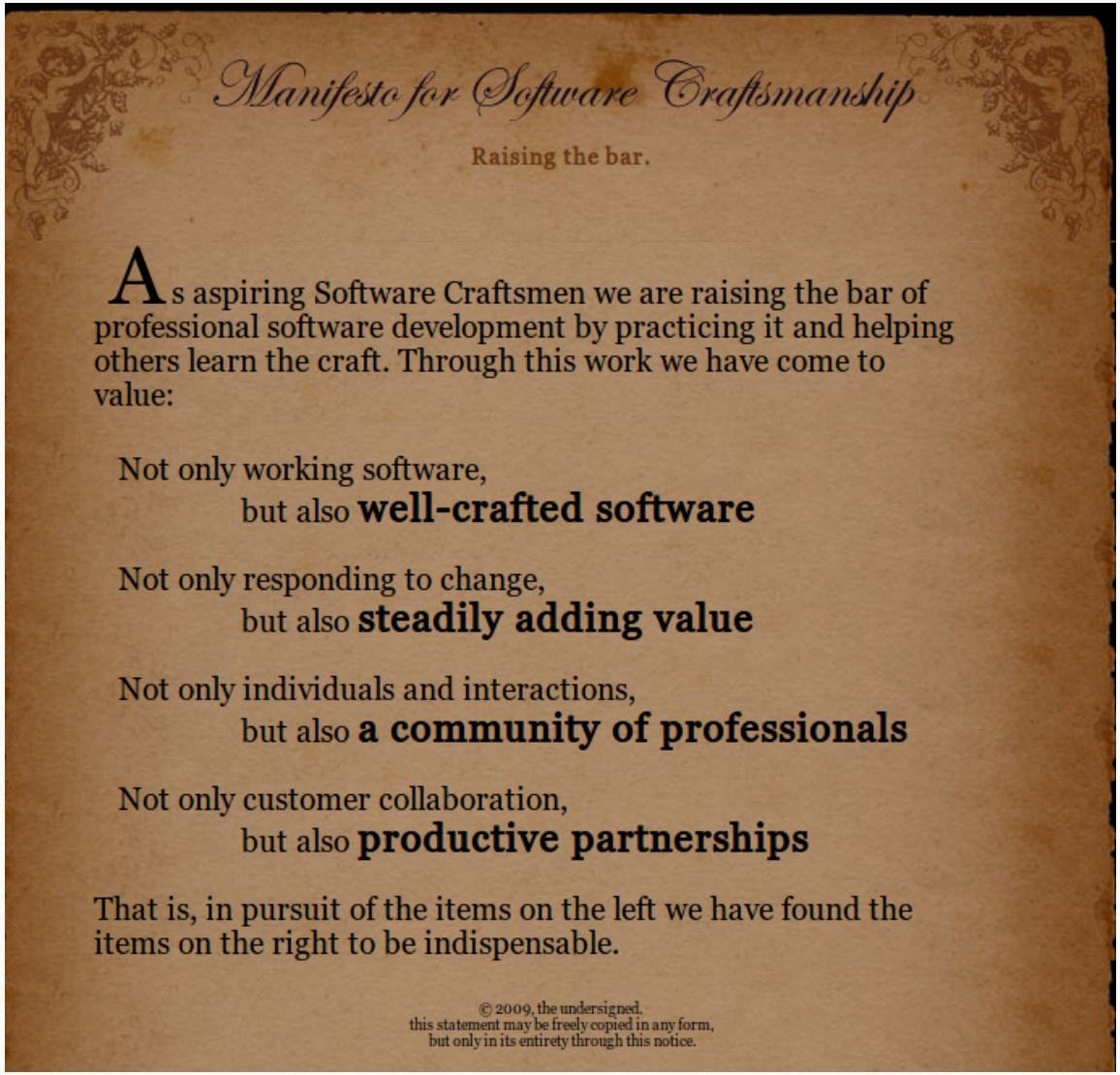
Bad design with functional programming

OOP and functional programming

Conclusions

Intro

Why I care about functional programming



Manifesto for Software Craftsmanship

Raising the bar.

As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

That is, in pursuit of the items on the left we have found the items on the right to be indispensable.

I identify with the software craft movement

Learning & Improving

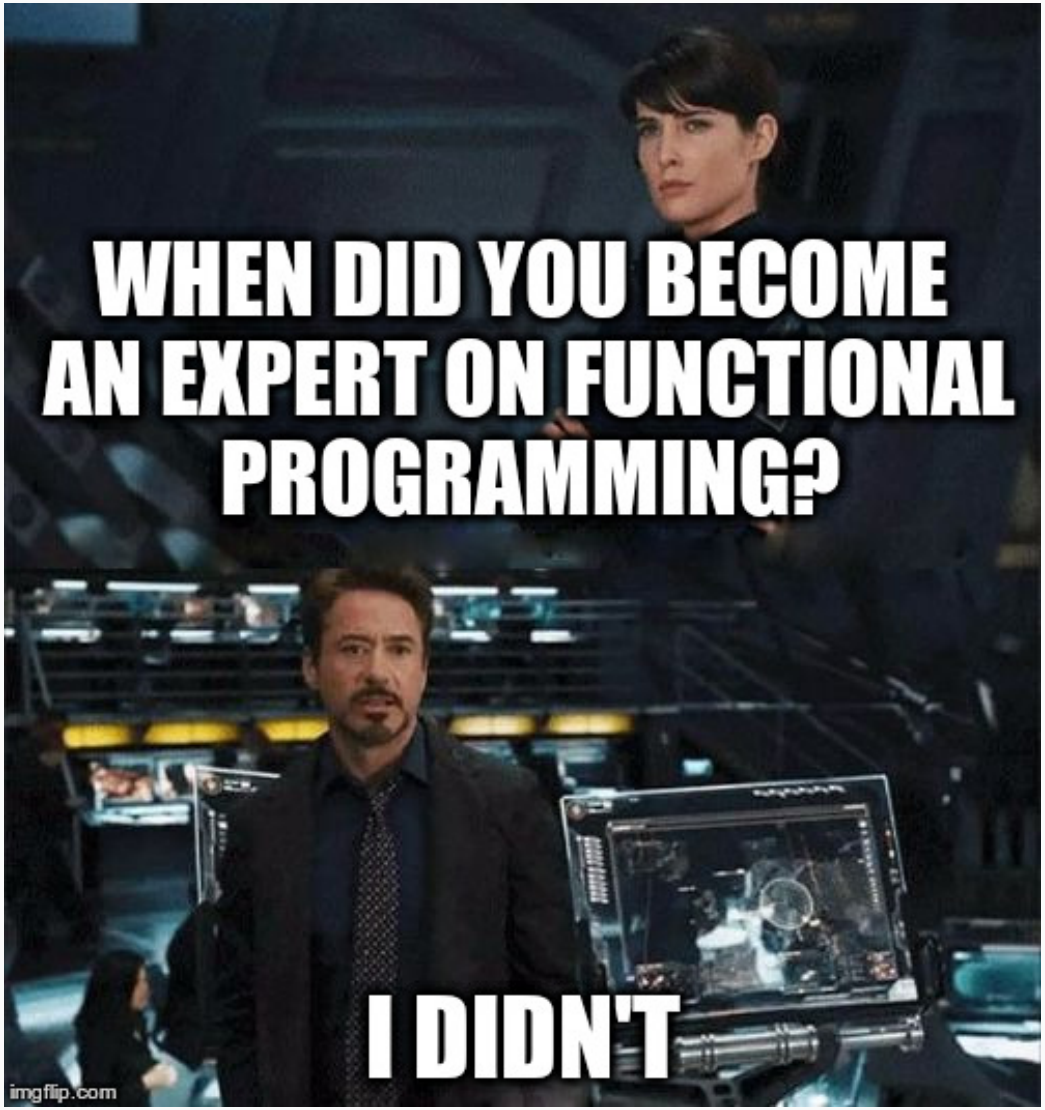
I want to learn as much as possible about my craft.

The evolution of the software industry is just recycling old ideas, and functional programming is old.

All programming languages support functional constructs today.

More restrictions on code (eg. immutability) lead to better code and better designs.

Disclaimer



I am an enthusiast, not an expert

Disclaimer

The code on the slides may be simplified for presentation purpose.

Find the valid examples on github: `https://github.com/alexboly` and `https://github.com/MozaicWorks`

My First Touch of Functional Programming

```
(defun sqrt-iter (guess x)
  (if (good-enough-p guess x)
      guess
      (sqrt-iter (improve guess x) x)))
```

Source: Professor Forrest Young, Psych 285

What is a monad?

> *The essence of monad is thus separation of composition timeline from the composed computation's execution timeline, as well as the ability of computation to implicitly carry extra data, as pertaining to the computation itself, in addition to its one (hence the name) output, that it will produce when run (or queried, or called upon).*

Source: `https://wiki.haskell.org/Monad`

A shorter explanation

> *All told, a monad in X is just a monoid in the category of endofunctors of X, with product × replaced by composition of endofunctors and unit set by the identity endofunctor.*

Saunders Mac Lane, “Categories for the Working Mathematician”

But don't worry, just learn Category Theory

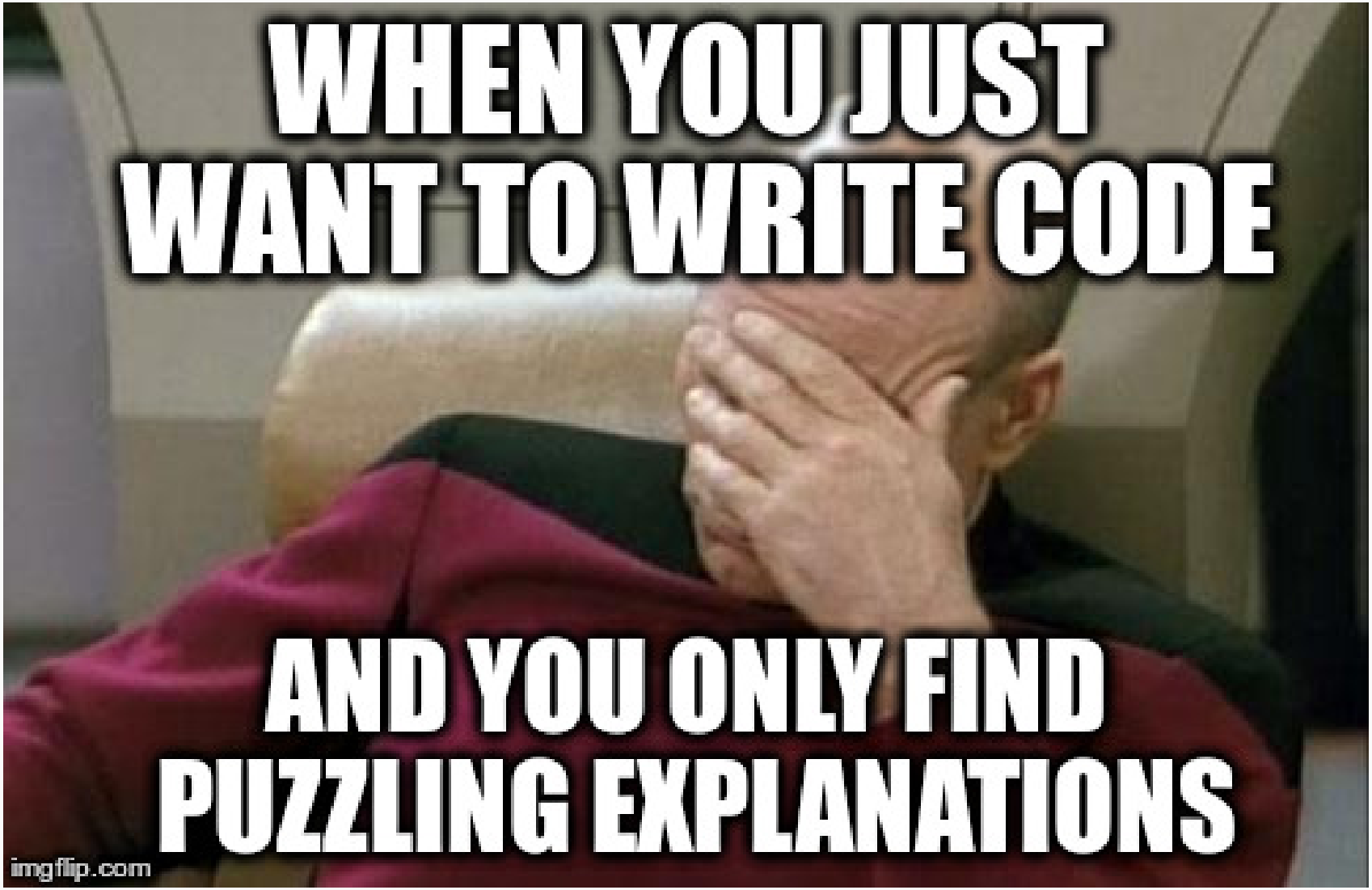
Category theory formalizes mathematical structure and its concepts in terms of a labeled directed graph called a category, whose nodes are called objects, and whose labelled directed edges are called arrows (or morphisms). A category has two basic properties: the ability to compose the arrows associatively and the existence of an identity arrow for each object.

...

Category theory has practical applications in programming language theory, for example the usage of monads in functional programming.

Wikipedia

Only one reaction



Whyyyyyyyyyyyy?

A practical view on functional programming

What I thought it was:

A way of writing unreadable code with many parantheses, with a strange order of operations, in a community that likes complicated explanations

How I understand it now:

A style of software design that focuses on functions and immutability to simplify code, to allow removal of certain types of duplication, and to show intent better

Mindset

OOP

Design the objects and interactions between them

Key idea: message passing

Functional programming

Start from input data and apply a set of transformations to get it to the desired output.

Key ideas: pure data structures and pure functions

Example 1: increment all elements of a list

Structured programming:

```
incrementAllElements(list& aList){
    for(int i = 0; i < aList.size(); ++i){
        aList[i] ++;
    }
}
```

Functional programming:

```
//Simplified C++ code
transform(aList, [](const int item){ return item + 1 });
```

Example 2: Get a string with 5 ‘t’ characters

Structured programming:

```
string result;
for(int i = 0; i < 5; ++i){
    result += 't';
}
```

Example 2 (cont'd)

Functional programming:

```
// Data in: a range from 1 to 5
// Data out: a string with 5 't'
// Transformations: transform each element from range
// to 't'(map), then join them (reduce)
// Groovy code
[1..5].collect{'t'}.join()

// C++ doesn't support it yet, except for boost
// Simplified C++ code using boost's irange
transform(irange(1, 5), [](){return 't';});
```

Example 3: Pacman moves on a line to the right

OOP

Let's create classes for: Pacman, Board, Wall, Dot, Movement etc.

Example 3 (cont'd)

Functional programming

Data in: a line with pacman on it

……>……

Data out: a line with pacman moving one square to the right, and a missing dot

…… >…..

Transformations:

- Get everything before pacman
- Get everything after pacman
- Create a new line formed from: everything before pacman, an empty space, pacman, everything after pacman except first element

Example 3 (cont'd)

```
const Line tick(const Line& initialLine){
    return (
            beforePacman(initialLine) +
            KindOfToken::Empty +
            KindOfToken::Pacman +
            removeFirst(afterPacman(initialLine))
           );
}
```

Full example at: https://github.com/alexboly/pacmanCpp/

Conclusion

FP mindset: Data in -> Transformations -> Data out

Each transformation is a pure function, except at the edge of the system (I/O).

Core Building Blocks

Pure functions

A function that returns the **same output** whenever receiving the **same input**

In C++ **const** is your friend

Example: Not pure function

```
list.insert(5); // throws exception if too many elements
                // otherwise, adds 5 to the list
```

Example: Pure function

```
const list insert(const list& aList, const int value) { ... };
newList = insert(aList, 5);
// always has the same behavior
// when passing in the same arguments
// unless externalities (eg. memory is full)
// Immutability!
```

Example: Immutable data structures

Immutable C++ example: `https://github.com/rsms/immutable-cpp`

```
auto a = Array<int>::empty();
a = a->push(1);
a = a->push(2);
a = a->push(3);
```

Lambdas / Anonymous functions

```
// Lambda variable
auto increment = [](auto value){return value + 1;};
assert(increment(2) == 3);
```

Lambda variable with specific captured context

```
int valueFromContext = 10;
auto appendValueFromContext =
    [const auto valueFromContext](auto value){
        return value + valueFromContext;
    };
assert(appendValueFromContext(10) == 20);
```

Lambda variable capturing all used variables by copy

```
int valueFromContext = 10;
auto appendValueFromContext =
    [=](auto value){
        return value + valueFromContext;
    };
assert(appendValueFromContext(10) == 20);
```

More details:

http://en.cppreference.com/w/cpp/language/lambda

Chaining functions

```
listOf3Elements = add(add(add(emptyList, 1), 2), 3);
```

Curry

Not this curry!

Curry. Haskell Curry.

Curry. Haskell Curry

Currying

```
// Without curry
list = add(list, 5);
list = add(list, 1000);

// With curry (bind in C++ 11)
auto addToEmptyList = bind(add, list<int>(), _1);
assert addToEmptyList(5) == list<int>({5});
assert addToEmptyList(1000) == list<int>({1000});
```

Functional composability

```
auto threeElementsList = [](int first, int second, int third){
    return add(add(addToEmptyList(first), second), third);
};

// In some programming languages,
// composing functions has its own syntax
// Eg. Groovy
def oneElementList = add << addToEmptyList
// same as add(addToEmptyList(), _)

assert(threeElementsList(0, 1, 2) == list<int>({0, 1, 2}));
```

Currying and composability

```
auto addTwoElementsToEmptyList = [](const int first, const int
    return add(addToEmptyList(first), second);
};

auto listWith0And1 = bind(addTwoElementsToEmptyList, 0, 1);

assert(listWith0And1() == list<int>({0, 1});
```

Higher level functionvalues

We can pass functions as parameters

```
auto incrementFunctionResult = [](const auto aFunction){
    return aFunction() + 1;
};

auto function1 = [](){
    return 1;
};

assert(incrementFunctionResult(function1) == 2);
```

Short list of higher level functions

Find them in or

- find_if
- transform
- reduce / accumulate
- count_if
- all_of / any_of / none_of
- ...

See examples on `https://github.com/MozaicWorks/functionalCpp`

Almost anything can be a function

```
//TRUE = {x -> { y -> x}}
auto TRUE = [](auto x){
    return [x](auto y){
        return x;
    };
};

//FALSE = {x -> { y -> y}}
auto FALSE = [](auto x){
   return [](auto y){
        return y;
   };
};
```

Church encoding for booleans (cont'd)

```
//IF = {proc -> { x ->  { y -> proc(x)(y) }}}
auto IF = [](auto proc){
    return [proc](auto x){
        return [x, proc](auto y){
            return proc(x)(y);
        };
    };
};

CHECK(IF(TRUE)("foo")("bar") == "foo");
```

Source: https://github.com/alexboly/ChurchEncodingCpp

Conclusions

- Pure functions are the basic building blocks of functional programming
- The best functions are small and polymorphic
- When the same parameter is passed to multiple function calls, consider using **curry** (bind in C++)
- When the return of a function is passed to another function multiple times, consider using **functional composition**
- Reuse existing functions as much as possible: **map** (transform in C++), **reduce**, **find** etc.
- Experiment with functional programming: **TicTacToe score**, **pacman**, **tetris**
- Replace traditional loops with functional loops as much as possible

Improve design with functional programming

Clarify loops intent

```
for(auto element=list.begin(); element != list.end(); element++
// I have to read all of this to understand what the loop does
}
```

Functional loops

```
transform(....); // transform a collection into another!

auto increment = [](auto value){return value + 1;};
transform(list, increment); // transform a list into another
                            // with incremented elements

transform(list, increment); // compute the sum
                            // of all incremented elements

// compute the sum of all incremented even numbers
reduce(find(transform(list, increment), evenNumber), plus)
```

What about web apps?

Data in: an HTTP request

Data out: HTML + a response code + something saved in a store

Transformations:

- validation
- sanitization
- canonicalization
- business rules
- save to database (mutable!)
- or read from database (mutable!)
- convert to view
- pass on to html rendering

Everything in between request and database, and database and response, is immutable!

Bad design with functional programming

Abstract functions != Readability

```
computeNextBoardOnAxis(board, currentAxis, nextAxis) {
    def movableTokensForAxis = KindOfToken.values().findAll {
        it.axis == nextAxis && it.isMovable
    }
    def rotateForward = { aBoard ->
        rotateBoardOnAxis(aBoard, currentAxis, nextAxis)
    }
    def rotateBack = { aBoard ->
        rotateBoardOnAxis(aBoard, currentAxis, nextAxis)
    }
    return rotateBack(
            computeNewBoard(rotateForward(board),
            movableTokensForAxis
            ))
}
```

My ugly code

```
def createCertificate(self, attendeeText, fileName):
    imageOpenOperation = OpenImageOperation(self.baseImagePath)
    attendeeTextWriteOperation = TextWriteImageOperation(attend
    courseTextWriteOperation = TextWriteImageOperation(self.cou
    dateTextWriteOperation = TextWriteImageOperation(self.dateT
    signatureOverlayOperation = ImageOverlayOperation(self.trai
    filters = ImageOperations([imageOpenOperation,
        attendeeTextWriteOperation, courseTextWriteOperation,
        dateTextWriteOperation, signatureOverlayOperation])
    filters.execute(fileName)
    return fileName
```

```
class TextWriteImageOperation:
    def __init__(self, text):
            self.text = text

    def execute(self, image):
            draw = ImageDraw.Draw(image)
            textPositionX = self.__getXPositionForCenteredText(
                if self.text.shouldCenter()
                else self.text.column()
            self.__drawText(draw, textPositionX)
            del draw
            return image;
```

OOP and functional programming

Relationship between OOP and functional programming

A class is nothing more than a set of **cohesive, partially applied pure functions**

– via JB Rainsberger

Remember first example?

```
// OOP version: Cohesion
class List{
    private listStorage

    add(element){....}
    removeLast(){....}
}

// Functional version
add(list, element)
removeLast(list)
```

```
// Equivalence

def add = [](auto listStorage, auto element){return ....};

// For oop bind function parameters
// to the data members of the class
auto oopAdd = [](element){ return add(initialListStorage, eleme

// Or curry (bind)
def oopAdd = bind(add, initialListStorage)

oopAdd(5)
```

Conclusions

My thoughts

Good:

- Functional programming mindset is very useful for data-centric applications
- Higher level functions simplify and clarify intent for data transformations
- Pure functions are very easy to test
- Clear separation between mutable and immutable state simplifies everything

Careful:

- Beware of too high abstraction & ensure your colleagues understand the code
- Carefully mix OOP with functional constructs

Can’t ignore:

- Functional programming is here to stay due to CPUs going multicore
- New applications: big data, reactive programming etc.
- All modern languages have FP constructs built in
- Clojure, Scala, Haskell, F# are used and will be used
- AI is built using FP (composability)

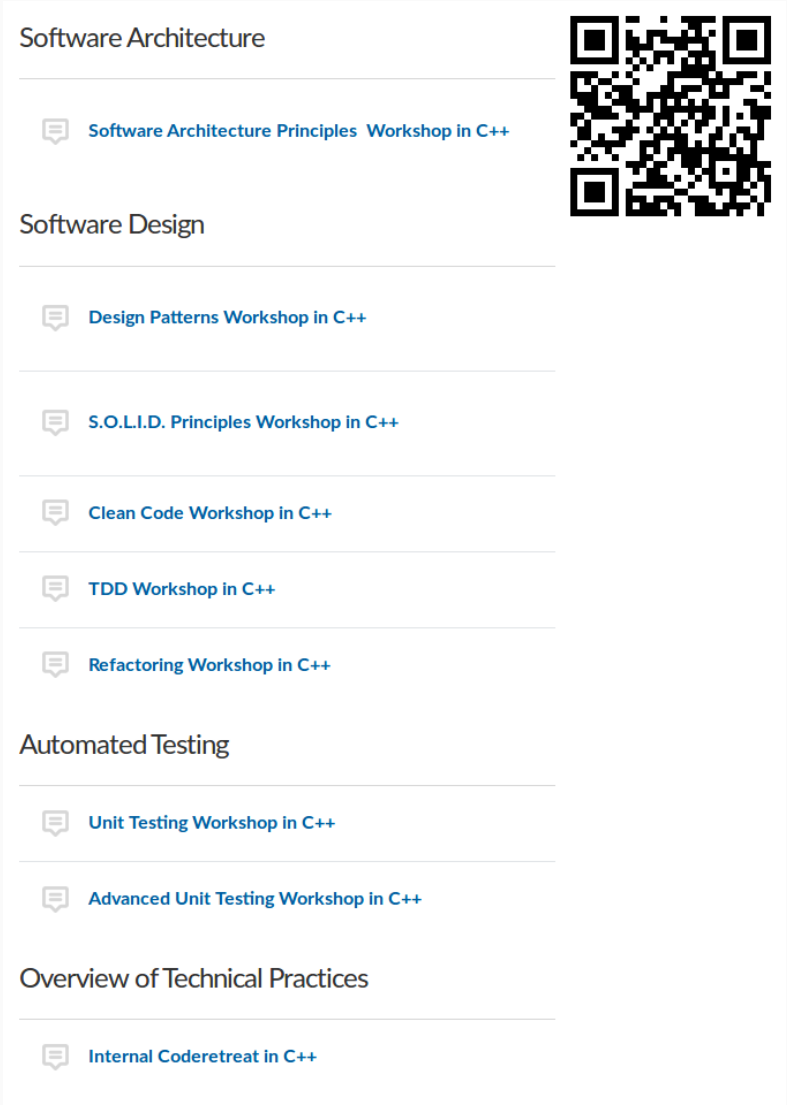
Learn more

Hidden loops:

`https://www.slideshare.net/alexboly/hidden-loops`

Removing structural duplication: `https://www.slideshare.net/alexboly/removing-structuralduplication`

Learn more at Mozaic Works workshops



https://mozaicworks.com/training/c-plus-plus/

Thank you!

I've been Alex Bolboacă, @alexboly, `alex.bolboaca@mozaicworks.com`

programmer, trainer, mentor, writer

at Mozaic Works

Think. Design. Work Smart.

`https://mozaicworks.com`

Q&A