Automating Patterns with Aspect-Oriented Programming and PostSharp

Ceci n'est pas une pipe

By Gael Fraiteur
ČVUT, May 2019

POSTSHARP

@gfraiteur

Agenda

- An evolutive history of programming languages
- A Language of Patterns
- The Missing ‘pattern’ Keyword
- Aspect-Oriented Programming
- AOP with PostSharp
- Case Study: Authorization

Section 1

An evolutive history of programming languages


back in
1951

hardware was fun
but quite simple to use

Instructions

Am	Transfer (m) to rX; add (rX) to rA); deliver the sum to rA; do not erase rX.
Bm	Erase rA and rX; transfer (m) to rA and rX.
Cm	Transfer(rA) to m; clear rA.

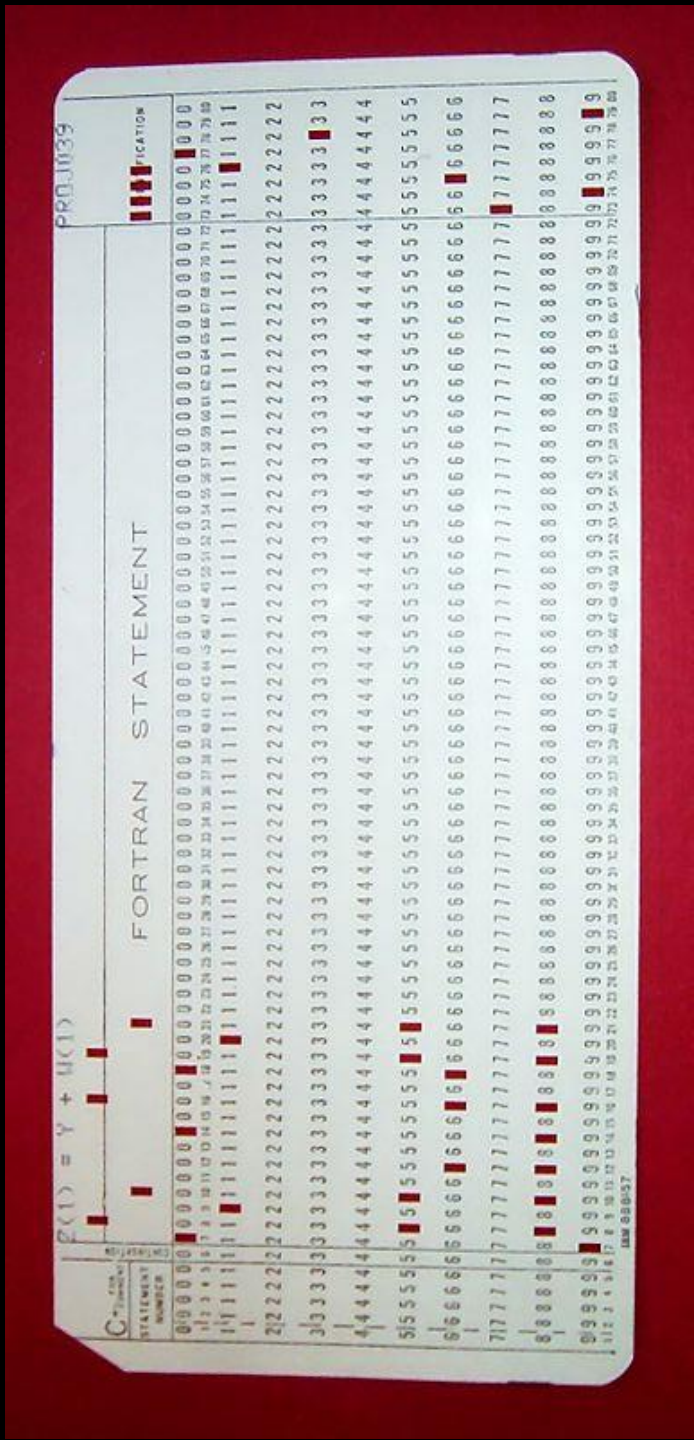
Example 1: (049) = x, (050) = y. Deliver the sum x+y = z to 051.

Mem. Loc.	Instruction		Remarks
020	B 049		x --> rA and rX
		A 050	y --> rX; x+y = z --> rA
021	C 051		(rA) = z --> 051; 0 --> rA.

Univac Assembly Language

FORTRAN (1955)

- Global Variables
- Arrays



“

The new invention caught quickly, no wonder, programs computing nuclear power reactor parameters took now

HOURS INSTEAD OF WEEKS

to write, and required much

LESS PROGRAMMING SKILL

COBOL (1959)

- Data Structures
- Natural-Language Syntax

```
PGM=MT200PG2 NOW WAITING AT +12AA : MOVE 936 SEND-MENU-MAP

 00925
 00926 SEND-MENU-MAP.
 00927*    EXEC CICS GETMAIN SET      (INITIAL-CA-BLL)
 00928*EXEC CICS GETMAIN SET (ADDRESS OF INITIAL-CA)
 00929*                     LENGTH  (INITIAL-CA-LENGTH)
 00930*                     INITIMG (BINARY-ZEROES)
 00931*                     END-EXEC.
 00932      MOVE '          00348    ' TO DFHEIV0
 00933      CALL 'DFHEI1' USING DFHEIV0  ADDRESS OF INITIAL-CA
 00934      INITIAL-CA-LENGTH BINARY-ZEROES.
 00935
TRAP          2ND TIME HERE
 00936      MOVE MENU-PGM TO PROG1 PROG2.
 00937*EXEC CICS SEND MAP     (MENU-MAP)
 00938*                MAPSET (MAPSET-NAME)
 00939*                ERASE
 00940*                MAPONLY
 00941*                END-EXEC.
 00942      MOVE '  0             00353    ' TO DFHEIV0
 00943      MOVE LENGTH OF DFHEICB TO DFHB0020
```

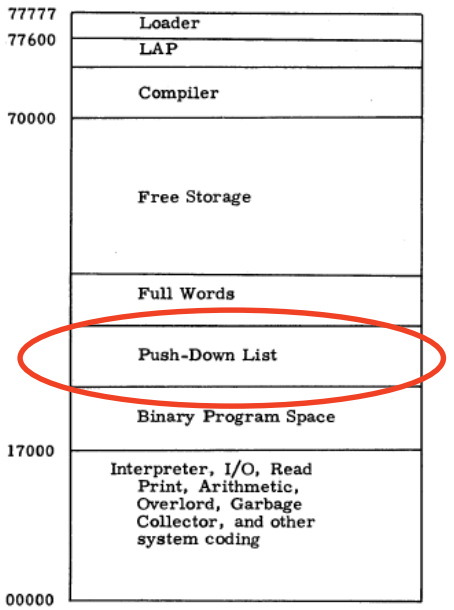
LISP (1959)

- Local Scopes (Recursion)
- Garbage Collection

APPENDIX G

MEMORY ALLOCATION AND THE GARBAGE COLLECTOR

The following diagram shows the way in which space is allocated in the LISP System.



The addresses in this chart are only approximate. The available space is divided among binary program space, push-down list, full-word space, and free-storage space as specified on the SIZE card when the system is made.

When the compiler and LAP are not to be used again, they may be eliminated by executing the pseudo-function excise. This part of the memory is then converted into free storage.

Free storage is the area in the computer where list structures are stored. This includes the property lists of atomic symbols, the definitions of all EXPR's and FEXPR's, evalquote doublets waiting to be executed, APVAL's, and partial results of the computation that is in progress.

Full-word space is filled with the BCD characters of PNAME's, the actual numbers

POSTSHARP

Simula (1967)

- Classes as abstraction mechanism
- Virtual procedures
- Object references
- Coroutines

```
Begin
   Class Glyph;
      Virtual: Procedure print Is Procedure print;
   Begin
   End;

   Glyph Class Char (c);
      Character c;
   Begin
      Procedure print;
        OutChar(c);
   End;

   Glyph Class Line (elements);
      Ref (Glyph) Array elements;
   Begin
      Procedure print;
      Begin
         Integer i;
         For i:= 1 Step 1 Until UpperBound (elements, 1) Do
            elements (i).print;
         OutImage;
      End;
   End;

   Ref (Glyph) rg;
   Ref (Glyph) Array rgs (1 : 4);

   ! Main program;
   rgs (1):- New Char ('A');
   rgs (2):- New Char ('b');
   rgs (3):- New Char ('b');
   rgs (4):- New Char ('a');
   rg:- New Line (rgs);
   rg.print;
End;
```

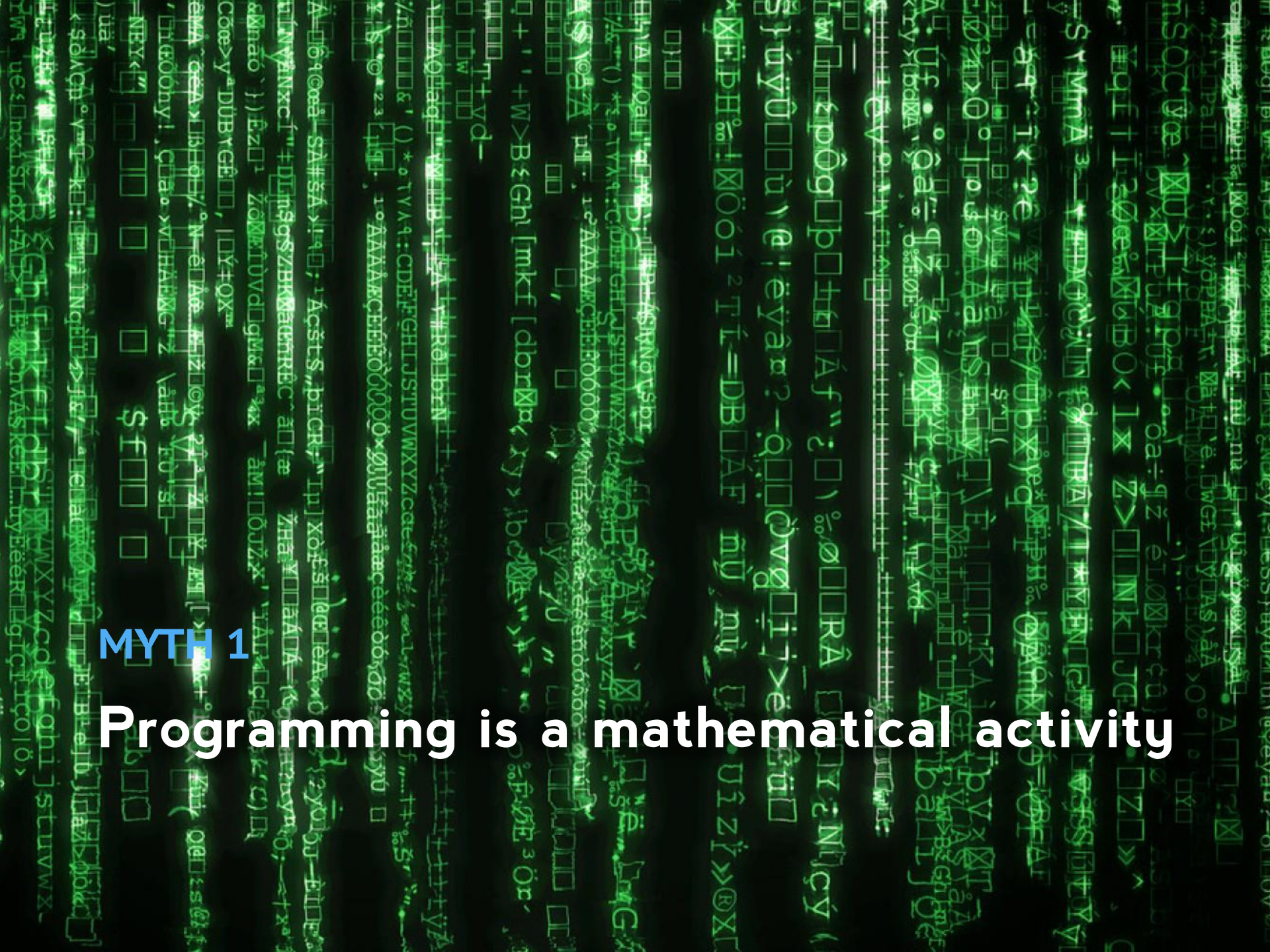
Programming languages get closer to the way we think,
not to the way we speak.

Human Language
Cognition
Communication
Social Organization

Programming Languages →

Machine Language
Execution

Copyright © by Arild Vågen



MYTH 1

Programming is a mathematical activity

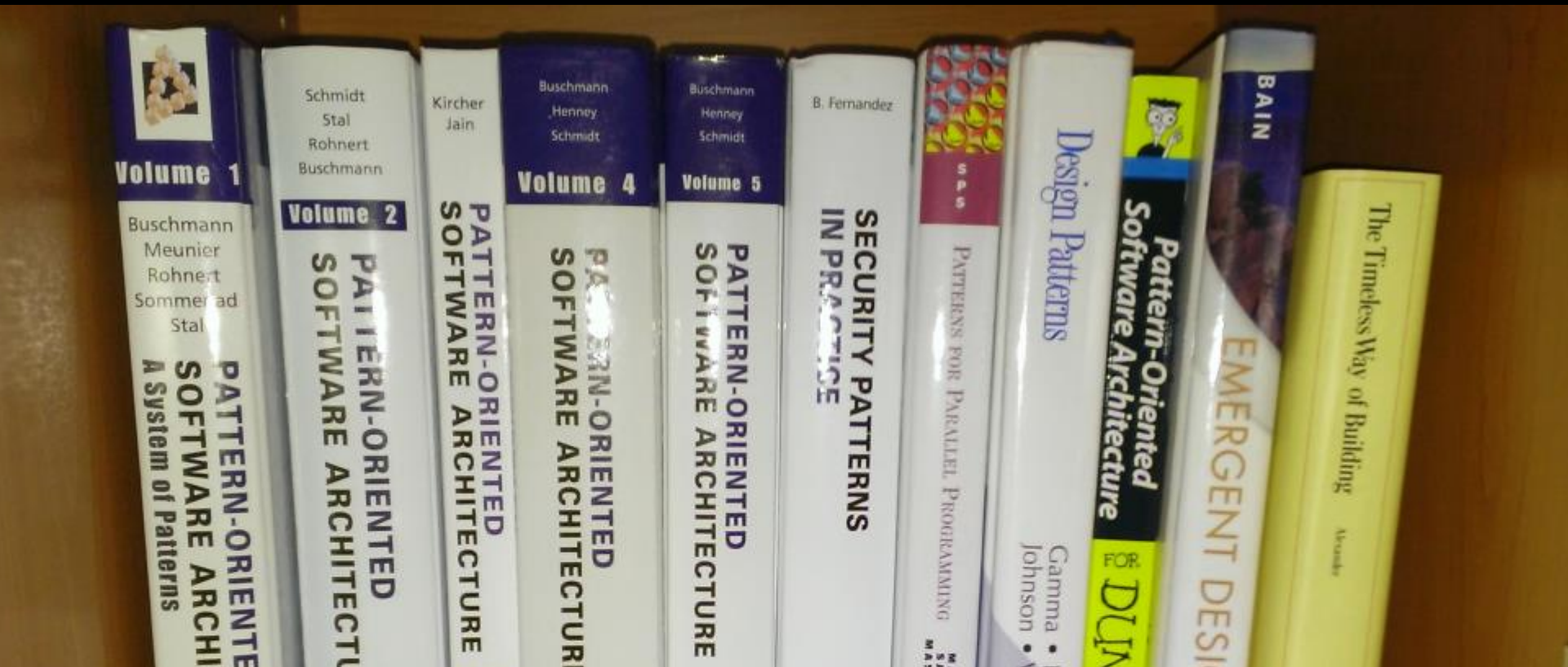
The real challenge of 21st century software industry is
to cope with the scarcity of human intelligence.

Section 1

A Language of Patterns

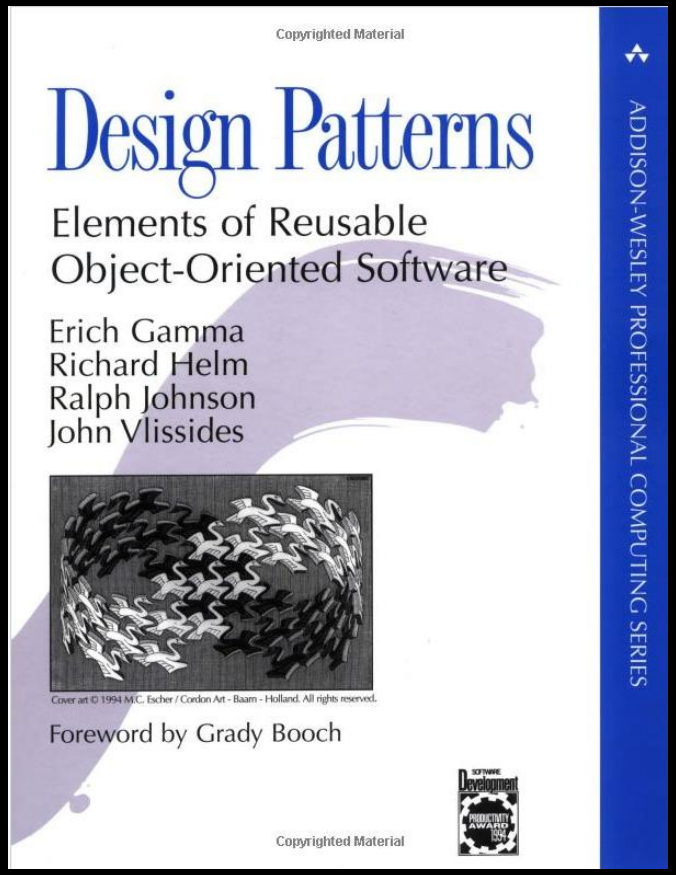
MYTH 2

All design patterns are found in books



Software Patterns (1995)

- Apply to OO design
- Canonic pattern description
- First catalogue of 23 patterns

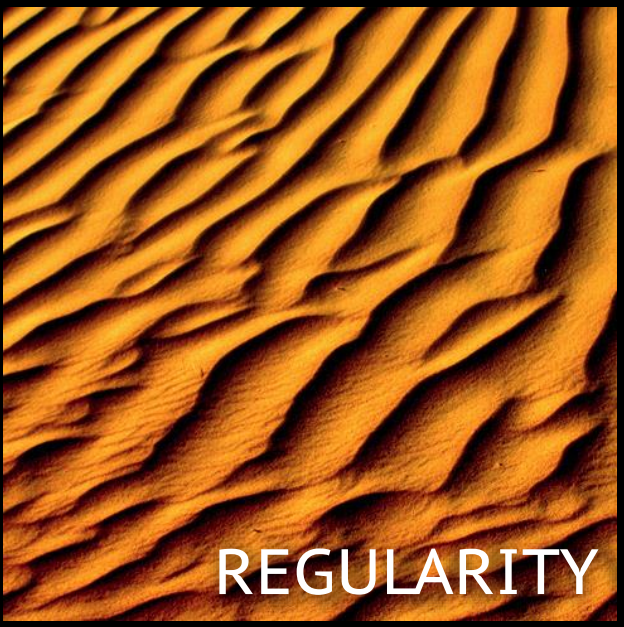


Back to the Sources (1977)

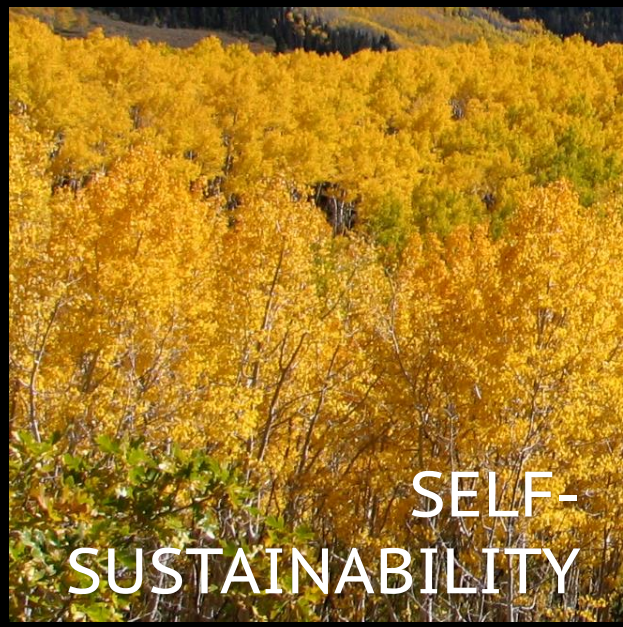
How is it possible that any simple farmer could make a house, a thousand times more beautiful than all the struggling architects of the last fifty years could do?

Christopher Alexander
in *The Timeless Way of Building, 1977*

Holašovice, Czech Republic

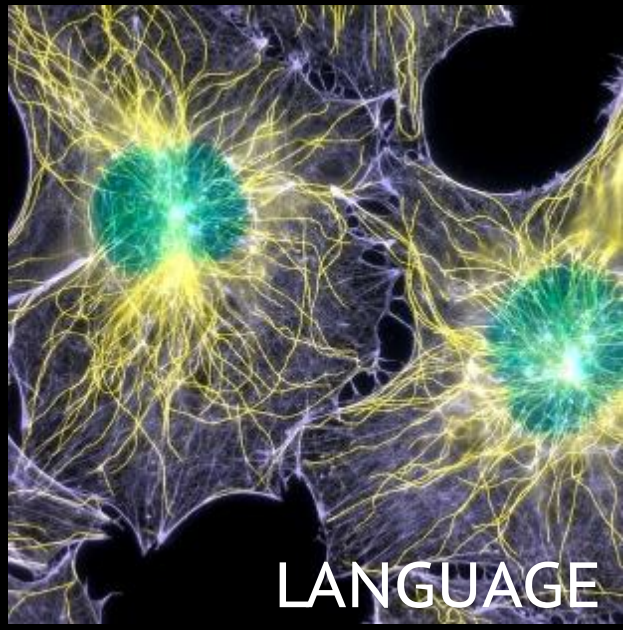

REGULARITY

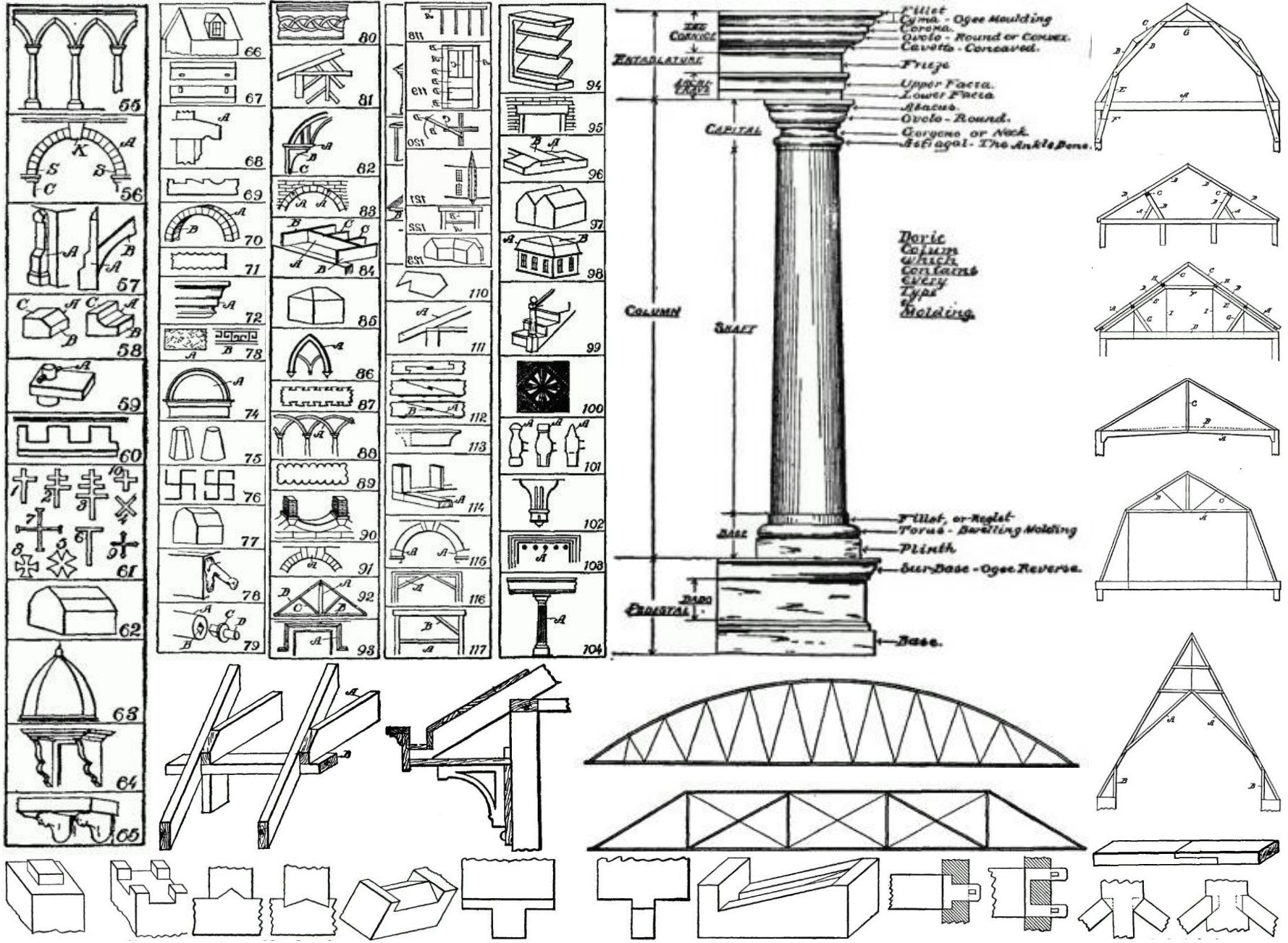

RESOLUTION
OF FORCES


SELF-
SUSTAINABILITY


VARIETY


UBIQUITY


LANGUAGE



Language for the Trade

Patterns are essential to the way humans communicate, reason, and improve their trades.


Patterns
belong to
the people!

The Pattern Manifesto

- ★ If it's repetitive, it deserves to be a pattern.
  Even if it isn't described in a book.
- ★ Patterns build languages,
  enable cognition and communication.
- ★ Programming is a human activity,
  not a mathematical abstraction.
- ★ Designing an API, a framework, an architecture…
  is to design for humans, not against the machine.

Section 3

The Missing ‘pattern’ Keyword

We Think with Patterns…

- Developers **think** at a high level of abstraction, using **design patterns.**

But There is No ‘Pattern’ Keyword!

- Therefore developers write repeating code:

  Boilerplate!

Typical Boilerplate

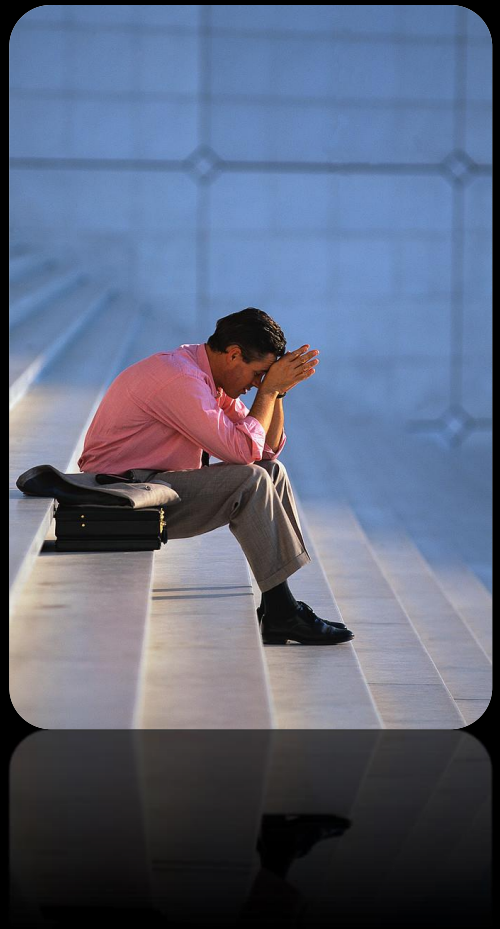
- INotifyPropertyChanged
- Undo/redo
- Code contracts (preconditions)
- Logging
- Transaction handling
- Exception handling
- Thread dispatching
- Thread synchronization
- Immutable
- Authorization
- Audit
- Caching

```
internal class CustomerProcesses
{
    private static readonly TraceSource trace =
        new TraceSource(typeof(CustomerProcesses).FullName);

    public bool ReserveBook(int bookId, int customerId)
    {
        if (bookId <= 0) throw new ArgumentOutOfRangeException("bookId");
        if (customerId <= 0) throw new ArgumentOutOfRangeException("customerId");

        trace.TraceInformation(
            "Entering CustomerProcesses.ReserveBook( bookId = {0}, customerId = {1} )",
            bookId, customerId);

        try
        {
            bool returnValue;

            for (int i = 0; ; i++)
            {
                try
                {
                    using ( var ts = new TransactionScope() )
                    {

                        Book book = Book.GetById( bookId );
                        Customer customer = Customer.GetById( customerId );


                        if ( book.BorrowedTo != null || book.ReservedTo != null )
                        {
                            returnValue = false;
                            goto exit;
                        }

                        book.BorrowedTo = customer;
                        returnValue = true;

                    }
                }
                catch (TransactionConflictException)
                {
                    if (i < 3)
                        continue;
                    else
                        throw;
                }

            }

            exit:
            trace.TraceInformation(
              "Leaving CustomerProcesses.ReserveBook( bookId = {0}, customerId = {1} )
                with return value {2}",
              bookId, customerId);

            return returnValue;
        }
        catch (Exception e)
        {
            trace.TraceEvent(TraceEventType.Error, 0,
                             "Exception: CustomerProcesses.ReserveBook(
                              bookId = {0}, customerId = {1} ) failed : {2}",
                             bookId, customerId, e.Message);

            if (ExceptionManager.Handle(e)) throw;


        }
    }
}
```

What is your ratio of useful code vs boilerplate?

Consequences of Boilerplate

- High development effort
- Poor quality of produced software
- Difficulty to add/modify functionality after first release
- Slow ramp-up of new team members



The Big Question

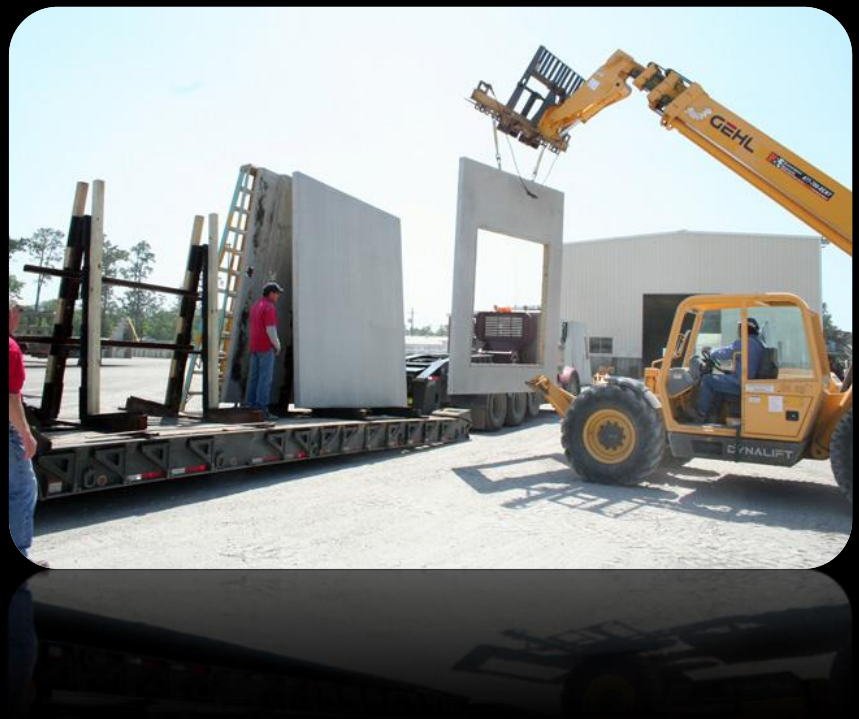
- How can we *write code* (not just design) with patterns…

  but without switching to a new language?

- You may want to consider…

Pattern-Aware Language Extensions

- Add support for patterns
- No more pattern hand-coding resulting in boilerplate!



Pattern-Aware Labguage Extensions

- Four main reasons to consider pattern-aware language extensions:
  1. Stop writing boilerplate code and deliver faster
  2. Build more reliable software
  3. Easier to add/modify functionality after first release
  4. New members contribute quicker

Reason 1.
Deliver Cheaper and Faster

- Fewer lines of code means fewer hours of work
  - Outsource repetitive work to compiler
  - Save time and costs immediately

Reason 2.
Build More Reliable Software

- Fewer lines of code means fewer defects
- Reliability becomes much more affordable
  - Cheaper to implement reliability features: logging, exception handling, caching, security,…
  - The “right” tool

Reason 3.
Easier to Modify Functionality

- Cleaner and shorter code is easier to understand.
- Business logic is more visible.
- Better architecture is future-proof.

Reason 4.
New Team Members Contribute Quicker

- Less to learn
- Tighter feedback loop with build-time validation.

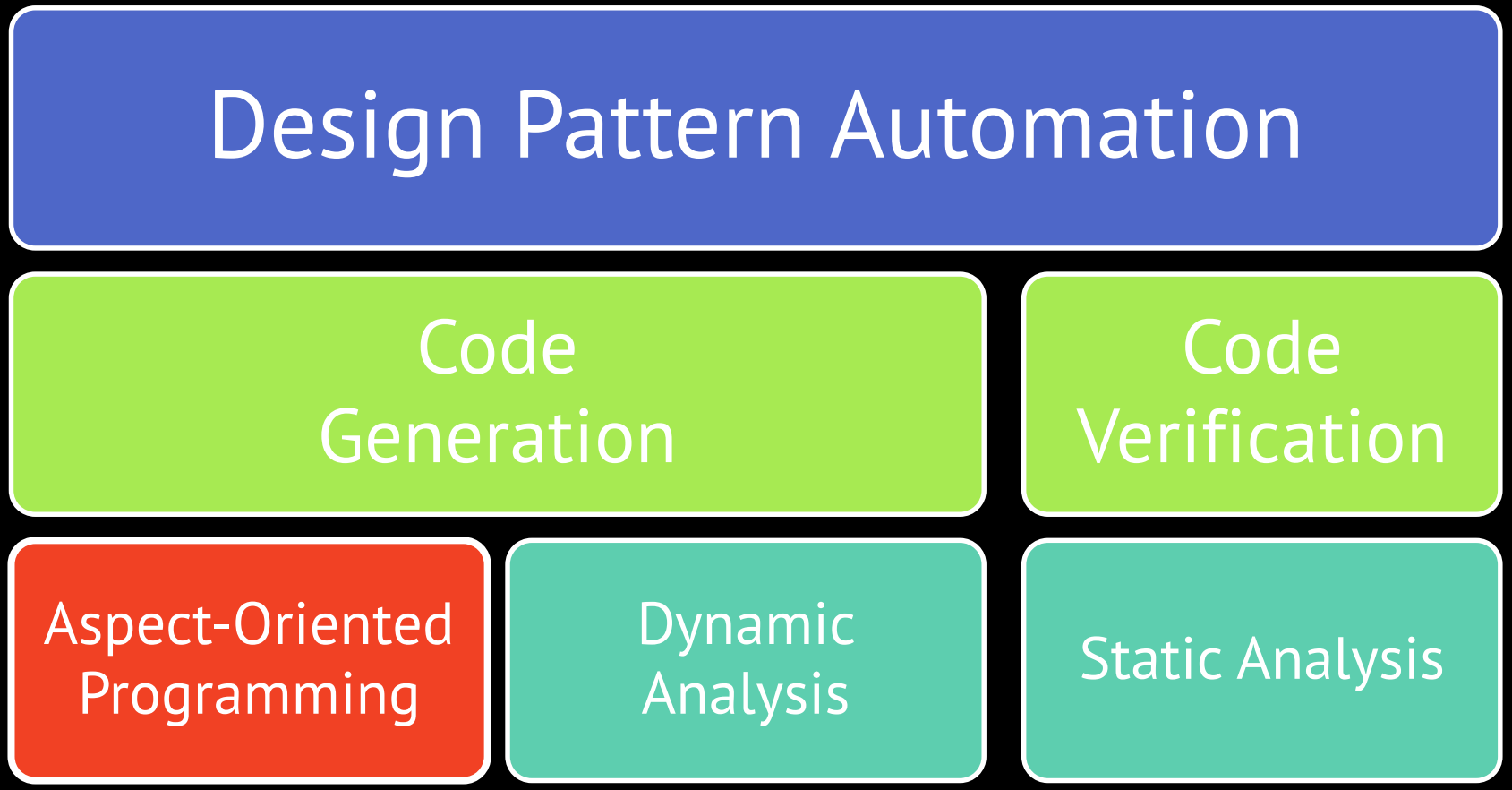
Lab 1: imagine a world with pattern-aware compilers

Life is Good!

Section 4

Aspect-Oriented Programming

Automating Design Patterns



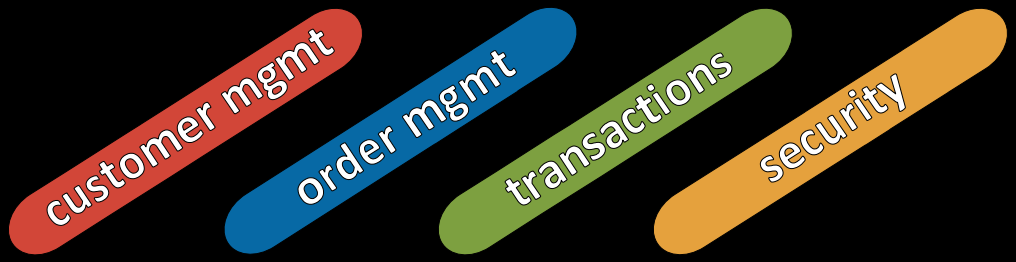
Aspect-Oriented Programming

Problem Domain

Cross-Cutting Concerns

Solution Domain

Separation of Concerns



What is AOP?

An extension of (not an alternative to) OOP that addresses the issue of cross-cutting concerns by providing a mean to:

- **Encapsulate** cross-cutting concerns
  into **Aspects** = collection of transformations of code
- **Apply** aspects to elements of code

AOP in .NET

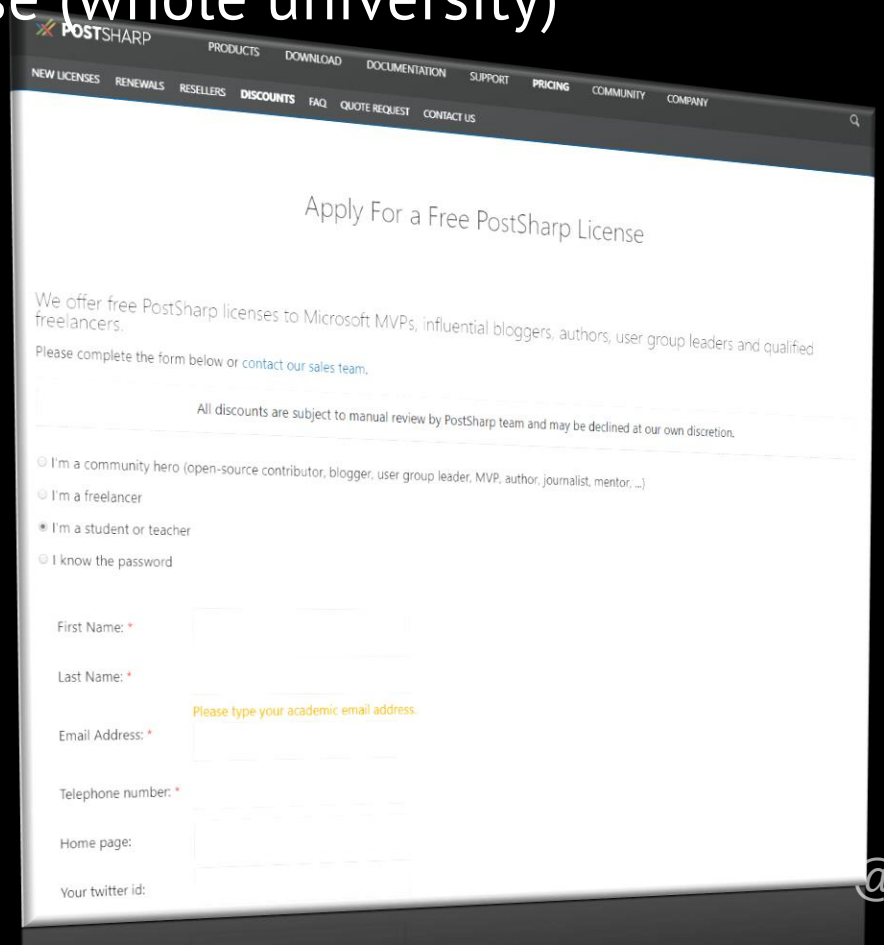
- PostSharp
- Several unmaintained PostSharp “clones”
- ASP.NET Action Filters
- Dynamic Proxies in most DI frameworks

Section 5

AOP with PostSharp

PostSharp For Academy

- Free licenses for student/teachers
- Free academic license (whole university)



About PostSharp

- OSS project started 2004
- Commercial since 2009
- Based in Prague, Czechia
- 9 employees, 5 full-time developers, 5 nationalities
- Zero sales person

PostSharp Components

Diagnostics

- Logging

XAML

- INotifyPropertyChged
- Dependency Prop.
- Code Contracts

Threading

- Threading Models
- Thread Dispatching
- Deadlock Detection

Other

- Caching
- Parent/Child

Aspect Framework

- Aspect Primitives
- Composite Aspects
- Multicasting
- Aspect Providers

Architecture Framework

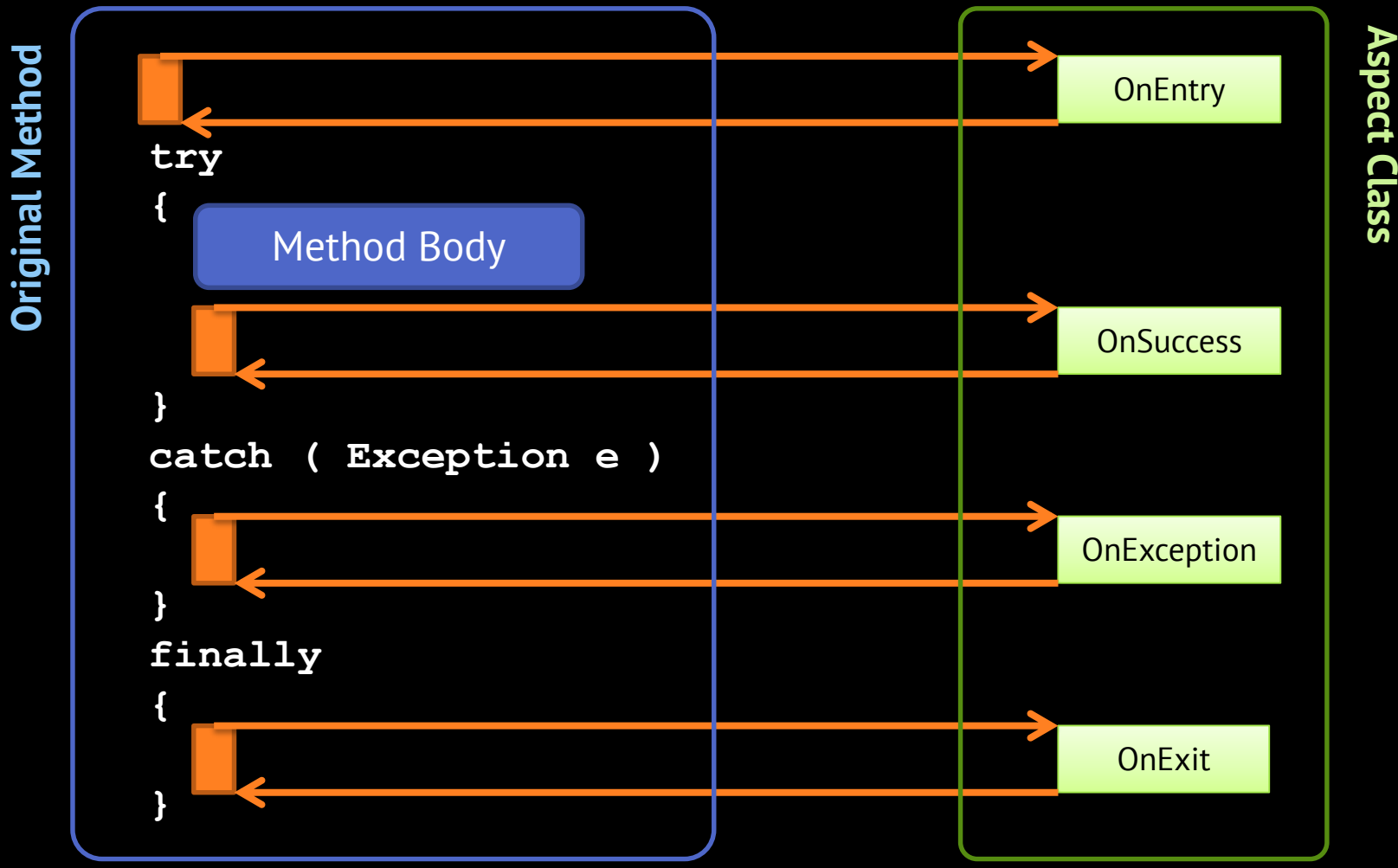
- Relationship browsing
- Syntax Tree
- Constraints

SDK

Aspect Primitives

- Method-Level
  - Try/Catch/Finally/Success
  - Intercept
- Field- and Property-Level
  - Intercept
  - Validate
- Parameter-Level
  - Validate
- Event-Level
  - Intercept
- Type-Level
  - Introduce Interface
  - Introduce Member
  - On Initialized
- Assembly-Level
  - Introduce resource
- Everywhere
  - Introduce custom attributes

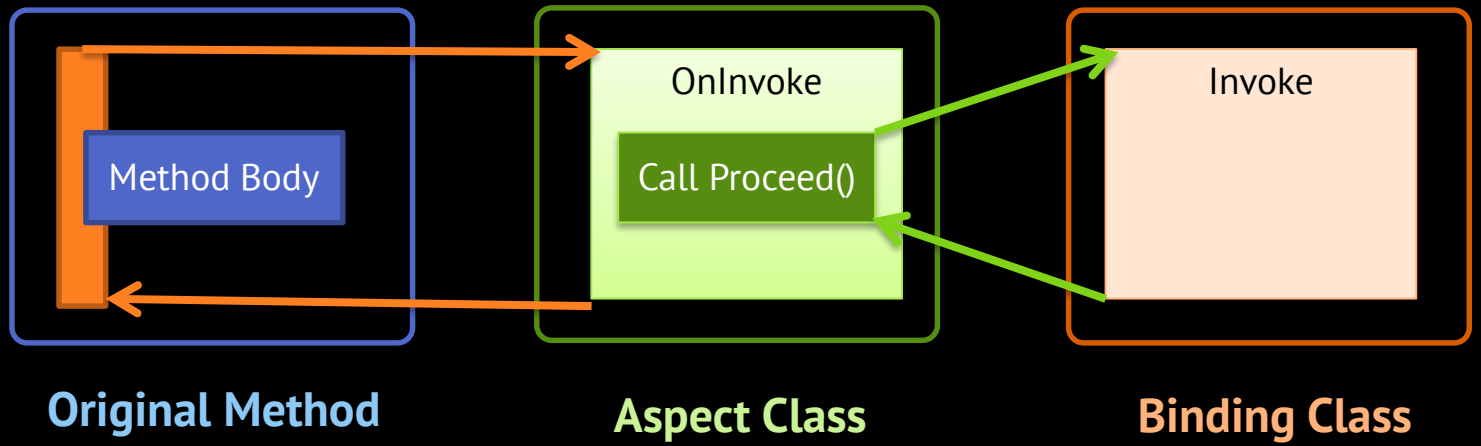
OnMethodBoundaryAspect



LAB 1

Logging

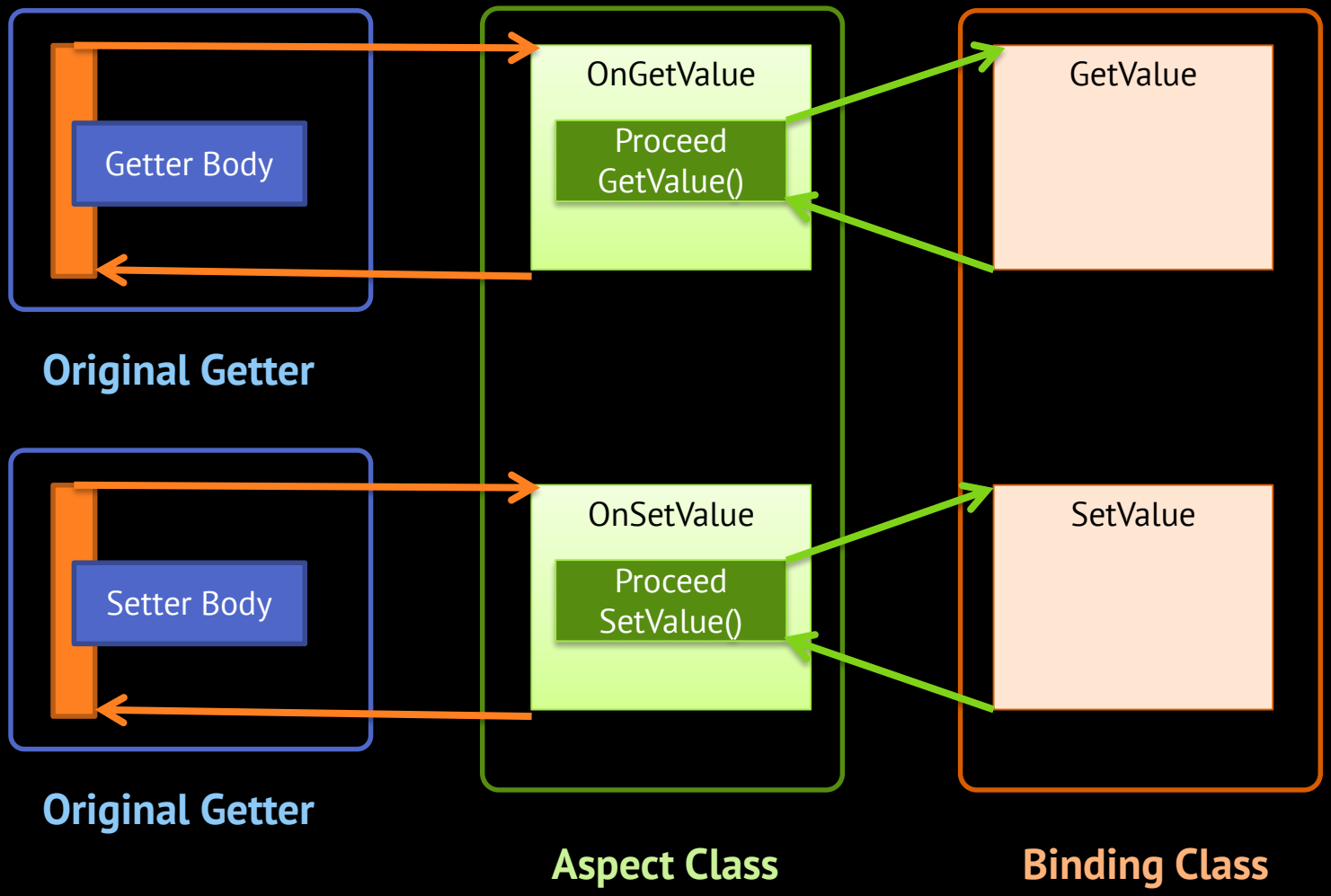
MethodInterceptionAspect



LAB 2

Auto Retry

LocationInterceptionAspect



LAB 3

Normalizing string values

Applying Aspects to Code

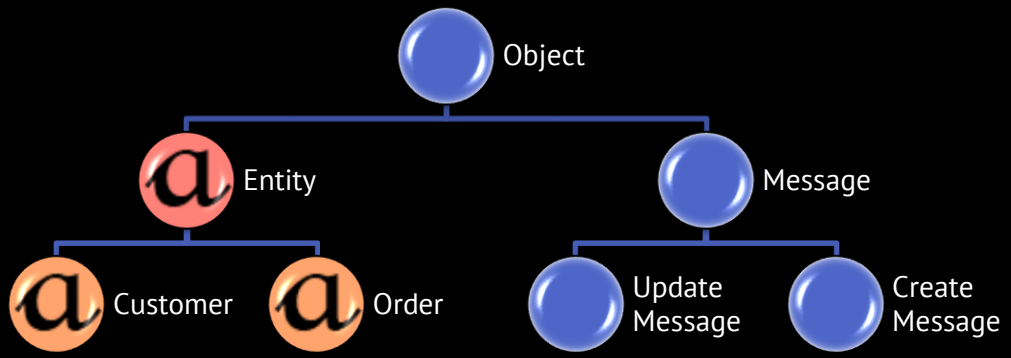
Applying Aspects to Code
Attribute Multicasting

- Class: MulticastAttribute
- Default: Apply to all. Filter by:
  - Name (wildcard/regular expression)
  - Modifier (public/private/protected, virtual, …)
  - Type of elements of code

```
[ExceptionDialog(AttributeTargetMembers = "On*Click")]
public partial class ContactControl : UserControl
{
```

Applying Aspects to Code
Attribute Inheritance

- Interfaces
- Classes
- Virtual Methods
- Assemblies (!)




```
[Serializable]
[AspectRoleDependency(AspectDependencyAction.Order, AspectDependencyPosition.After, StandardRoles.DataBinding)]
[MulticastAttributeUsage(MulticastTargets.Class,Inheritance = MulticastInheritance.Strict)]
public sealed class UndoableAttribute : InstanceLevelAspect
{
```

- or -

```
[Undoable(AttributeInheritance = MulticastInheritance.Strict)]
public abstract class Entity
{
```

LAB 5

Multicasting

Applying Aspects to Code
Aspect Provider

- Add aspects dynamically
- Arbitrary complexity (read external files, …)
- Create object graphs of aspect instances

```
class CustomAspect : TypeLevelAspect, IAspectProvider
{
    public IEnumerable<AspectInstance> ProvideAspects(object targetElement)
    {
        Type targetType = (Type)targetElement;

        return from method in targetType.GetMethods(BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public)
               where method.IsDefined(typeof(ObfuscationAttribute), true)
               select new AspectInstance(method, new CheckLicenseAspect());

    }
}
```

Examples of language extensions

- Composition (parent/child)
- Threading models
- Caching
- Serialization
- Security

Section 6

Case Study: Authorization

Authorization: the Model

- Role-Based Model
  - **Action** requires **Permission**
  - **Policy** assigns **Permission** to **Role**
  - **Subject** belongs to **Role** for **Entity**

Responsible?

Developer

Administrator

User / Manager

Our Challenge:
to make it *easy* for the developer and *safe* in case of human error.

Permission API

- Default permissions:
  - Read: property/field getters
  - Write: property/field setters

  Pessimistic design:
  Fail fast, fail soon, and avoid security holes.

- Overriding permission:

  [RequiresPermission]
  on fields, properties, methods and parameters.

  Simple design:
  A single attribute for all kinds of declarations.

Authorization: Implementation

- RequiresPermission: the API
- Implementation:
  - LocationAuthorizationAspect
  - MethodAuthorizationAspect
  - ApplyDefaultPermissionsAttribute

There is *much* more…

- Aspect Framework
  - Robust Aspect Composition
  - Composite Aspects
- Architecture Framework
- Patterns Libraries
  - Model
  - Threading
  - Diagnostics
  - Caching

…

…

Summary

- Compilers must do more to raise abstraction level
- Patterns must become first-level language constructs
- There are (imperfect) solutions today

Copyright © by Arild Vågen

Let's Get Started!

- Go to www.postsharp.net/download.
  Get your free student license:
  http://bit.ly/postsharp-student
- All examples of this demo are on:
  samples.postsharp.net.
- Complete documentation on:
  doc.postsharp.net..