Evaluating Architectures With Dependency Matrices

Michael Stiefel
Reliable Software, Inc.
www.reliablesoftware.com
development@reliablesoftware.com

How do you know an architecture is good?

Can you compare architectures ?

What does Architecture do?

Structures the software

Supports building the product

Why?

Writing a line of code is…

an act of design

part of a manufacturing process

Architecture as Design…

"Architects! We Don't Need No Stinkin' Architects"

Architecture as Manufacturing

Maximizing Feature Delivery…

Maximizes Economic Value

Theory of Constraints

Design around Bottleneck

View Work as Queues

Given Fixed Capacity, Batch Size Matters

Large Batch Size means…

Longer Time to Integrate

Variability in Results From Stale Requirements

Lower Quality due to Delayed Feedback

Wait Time = Queue Size / Processing Rate

Little's Law

Smaller the pieces of work….

the faster it goes.

Smaller software use cases….

require decoupled systems.

Decoupling starts with architecture.

Separate deploy

Separate rollback

More rapid testing and feedback

Minimize coupling by finding dependencies.

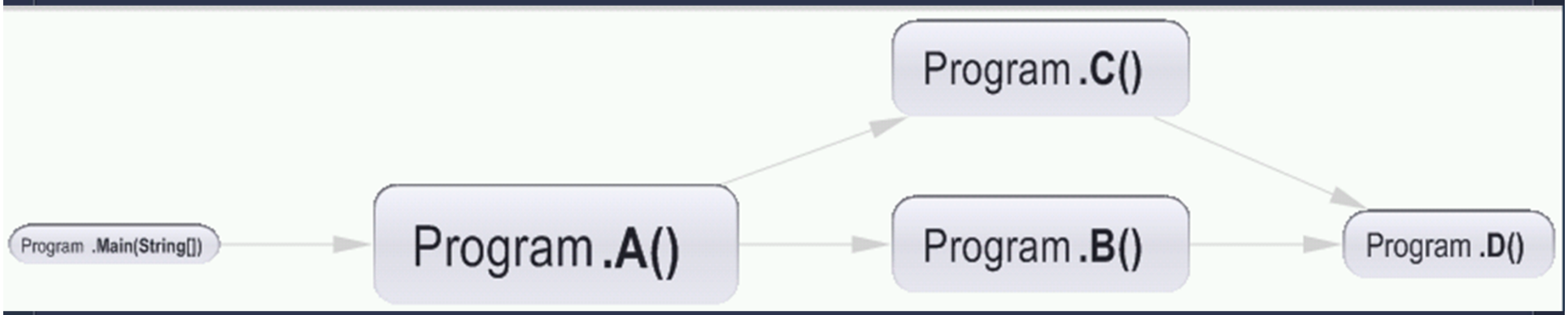
Essential Coupling

Inessential Coupling

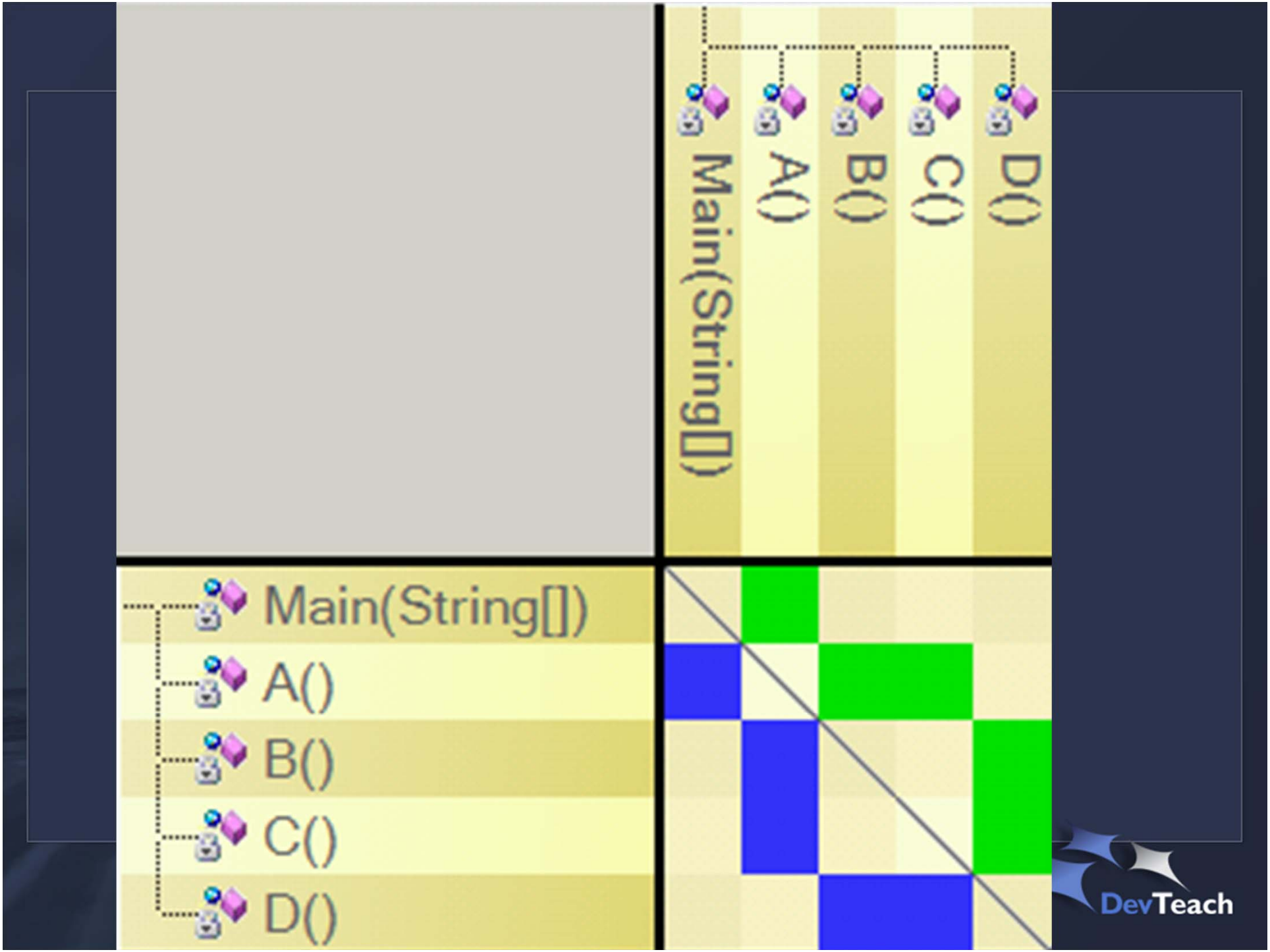
Dependency Structure Matrix

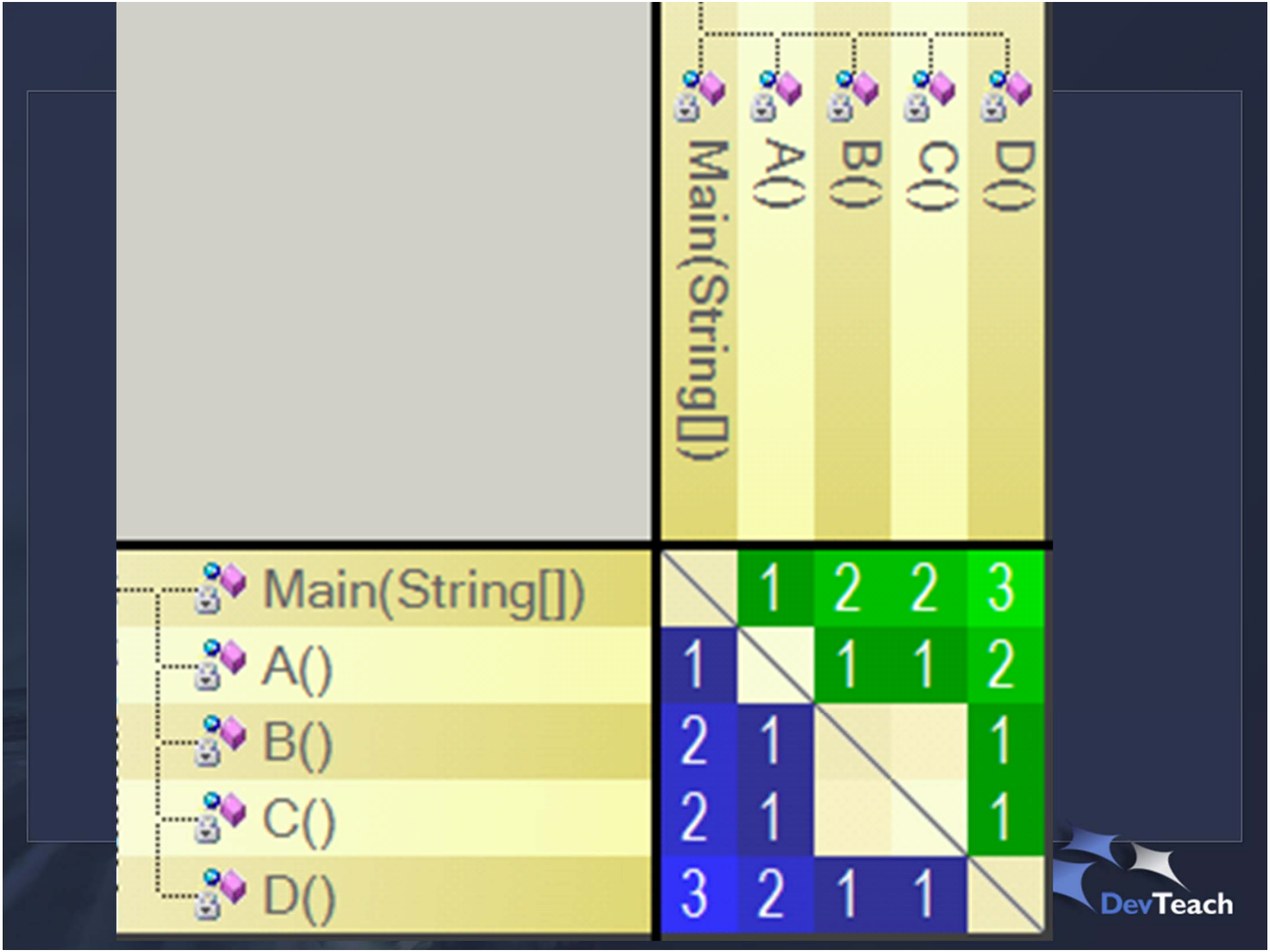
SimpleDependencyMatrix.Program — Main(string[] args)

```
{
    static class Program
    {
        static void Main(string[] args)
        {
            A();
        }

        private static void A()
        {
            Console.WriteLine("A"); B(); C();
        }

        private static void B()
        {
            Console.WriteLine("B"); D();
        }

        private static void C()
        {
            Console.WriteLine("C"); D();
        }

        private static void D()
        {
            Console.WriteLine("D");
        }
```



DevTeach


Main(String[])
A()
B()
C()
D()
Main(String[])
A()
B()
C()
D()

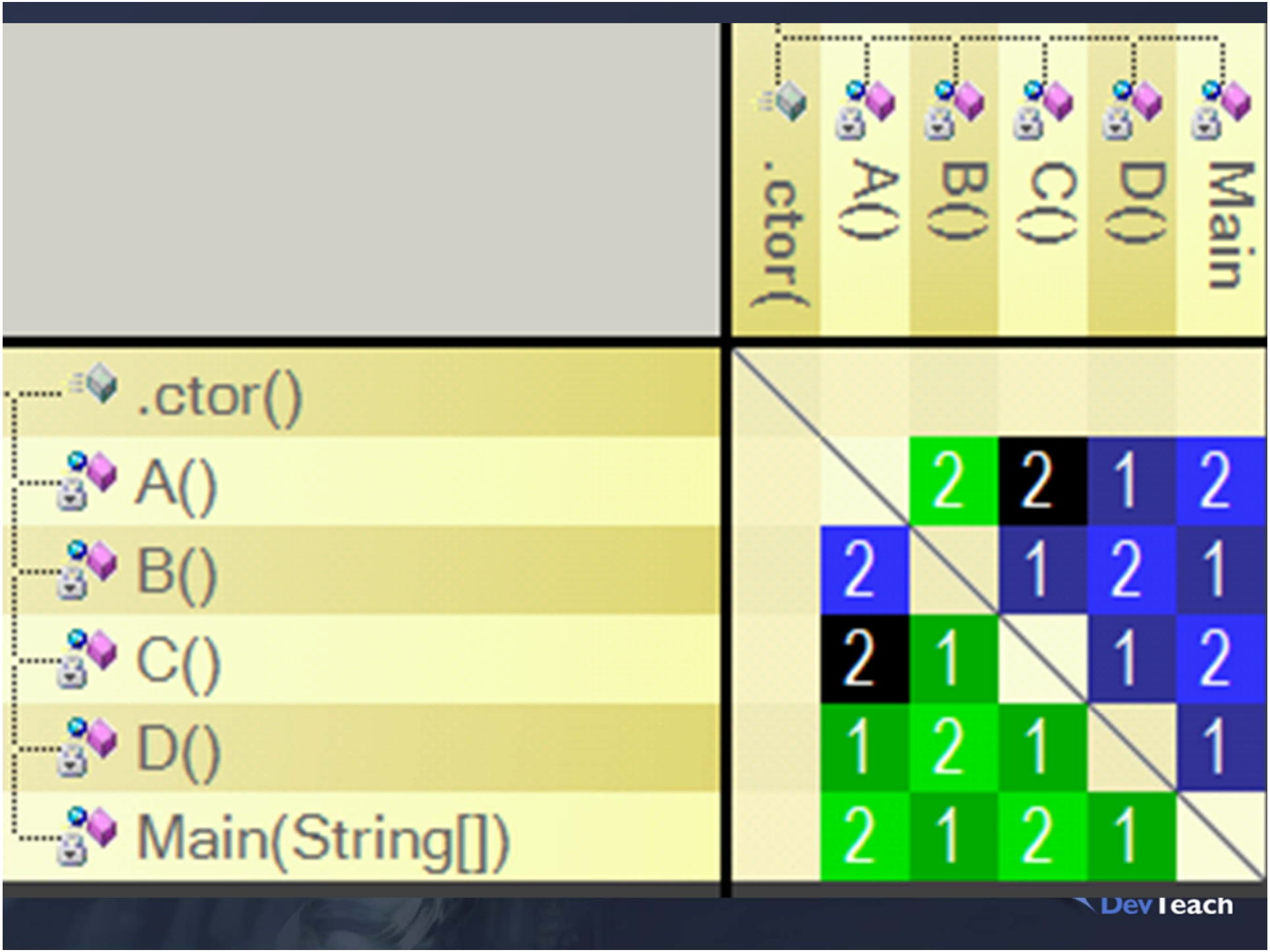

Main(String[])
A()
B()
C()
D()
Main(String[])
A()
B()
C()
D()
1 2 2 3
1 1 1 2
2 1 1
2 1 1
3 2 1 1
DevTeach

NDepend

Architectural Patterns

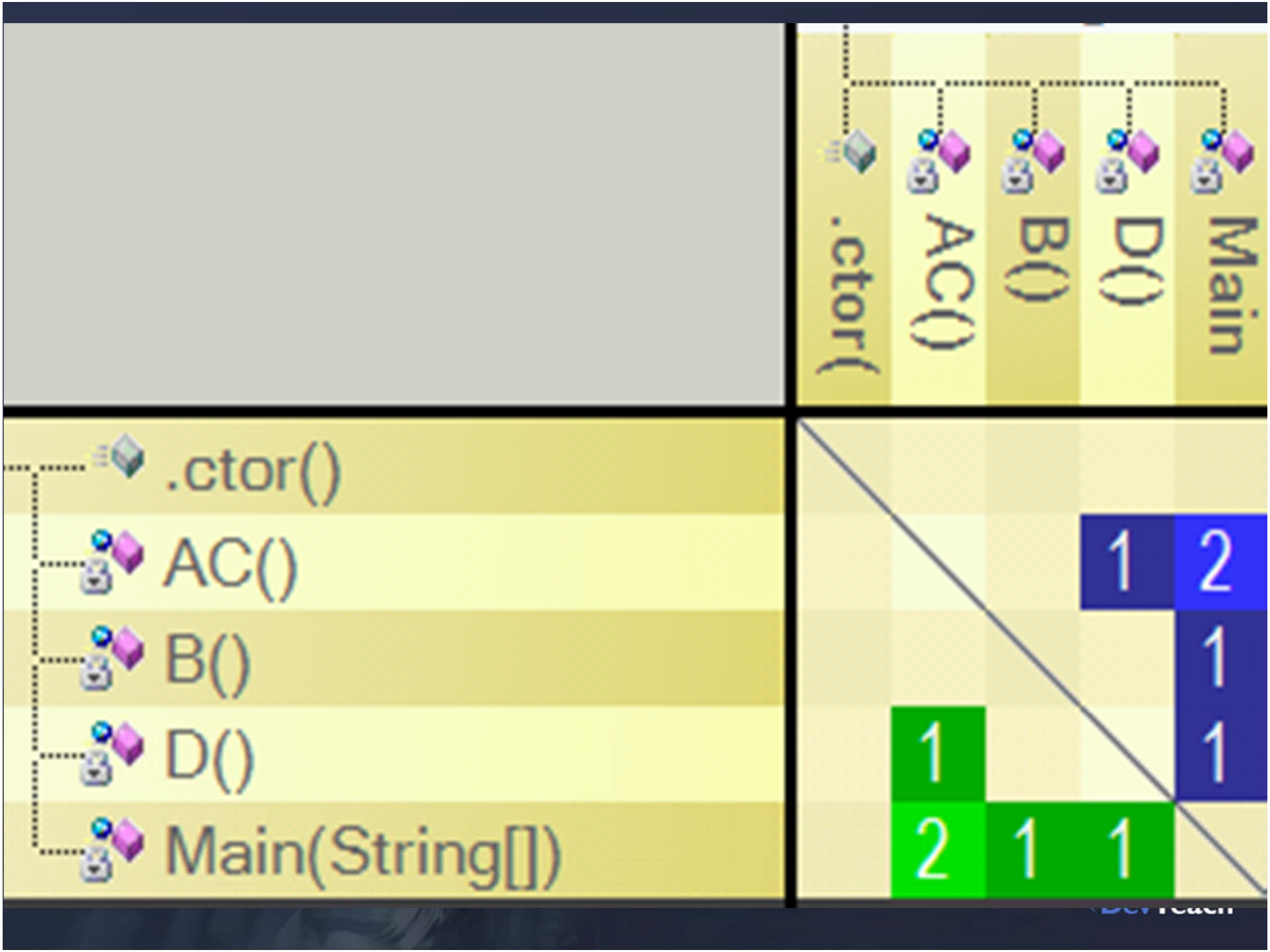
Layer Application (Acyclic Graph)

```
        static void Main(string[] args)
        {
            B();
            D();
        }

        private static void A()
        {
            Console.WriteLine("A"); C();
        }

        private static void B()
        {
            Console.WriteLine("B");
        }

        private static void C()
        {
            Console.WriteLine("C"); A(); B();
        }

        private static void D()
        {
            Console.WriteLine("D"); A(); C();
        }
```


.ctor(
A()
B()
C()
D()
Main
.ctor()
A()
B()
C()
D()
Main(String[])
2 2 1 2
2 1 2 1
2 1 1 2
1 2 1 1
2 1 2 1
DevTeach

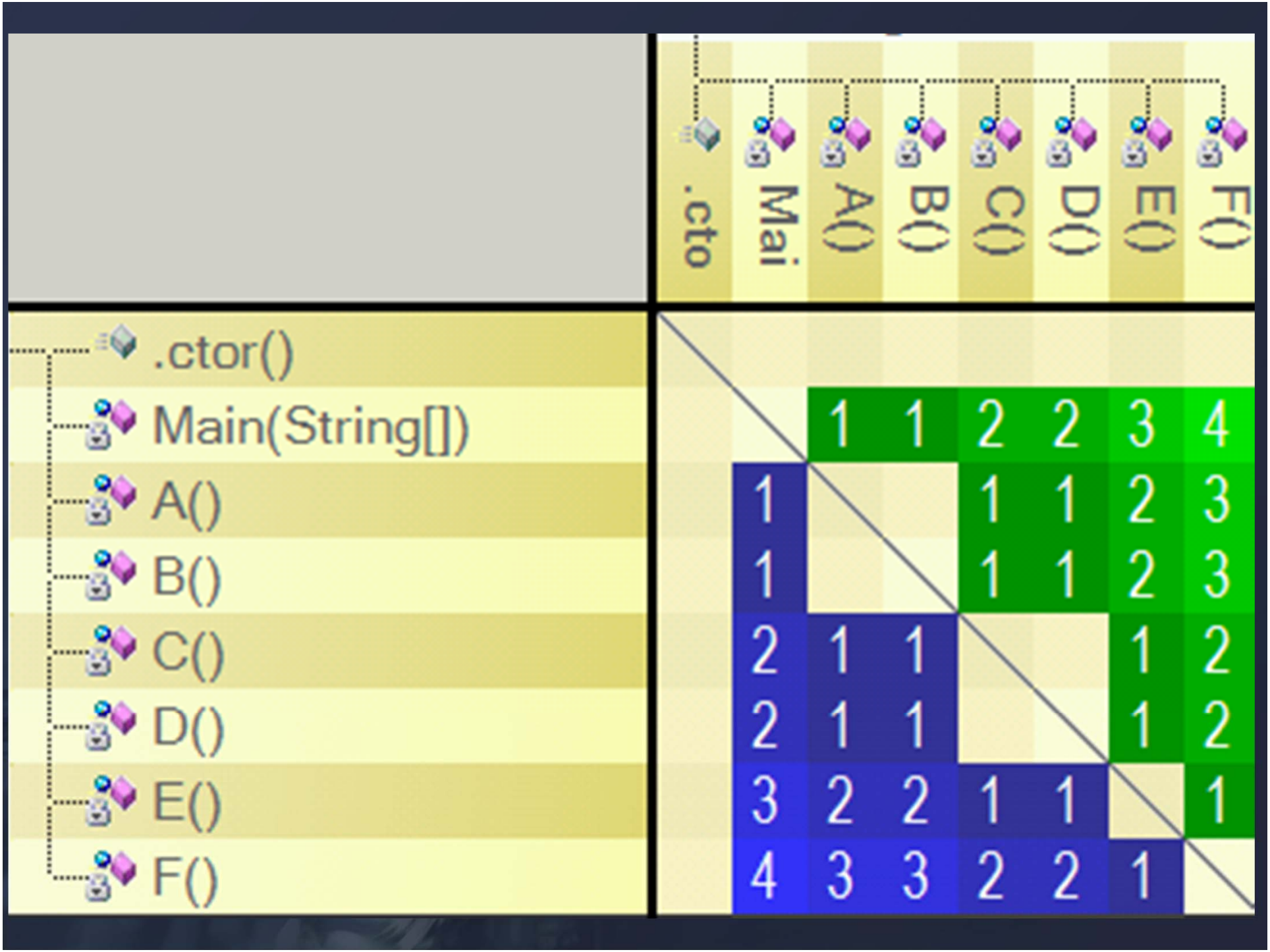
Partitioning Refactor

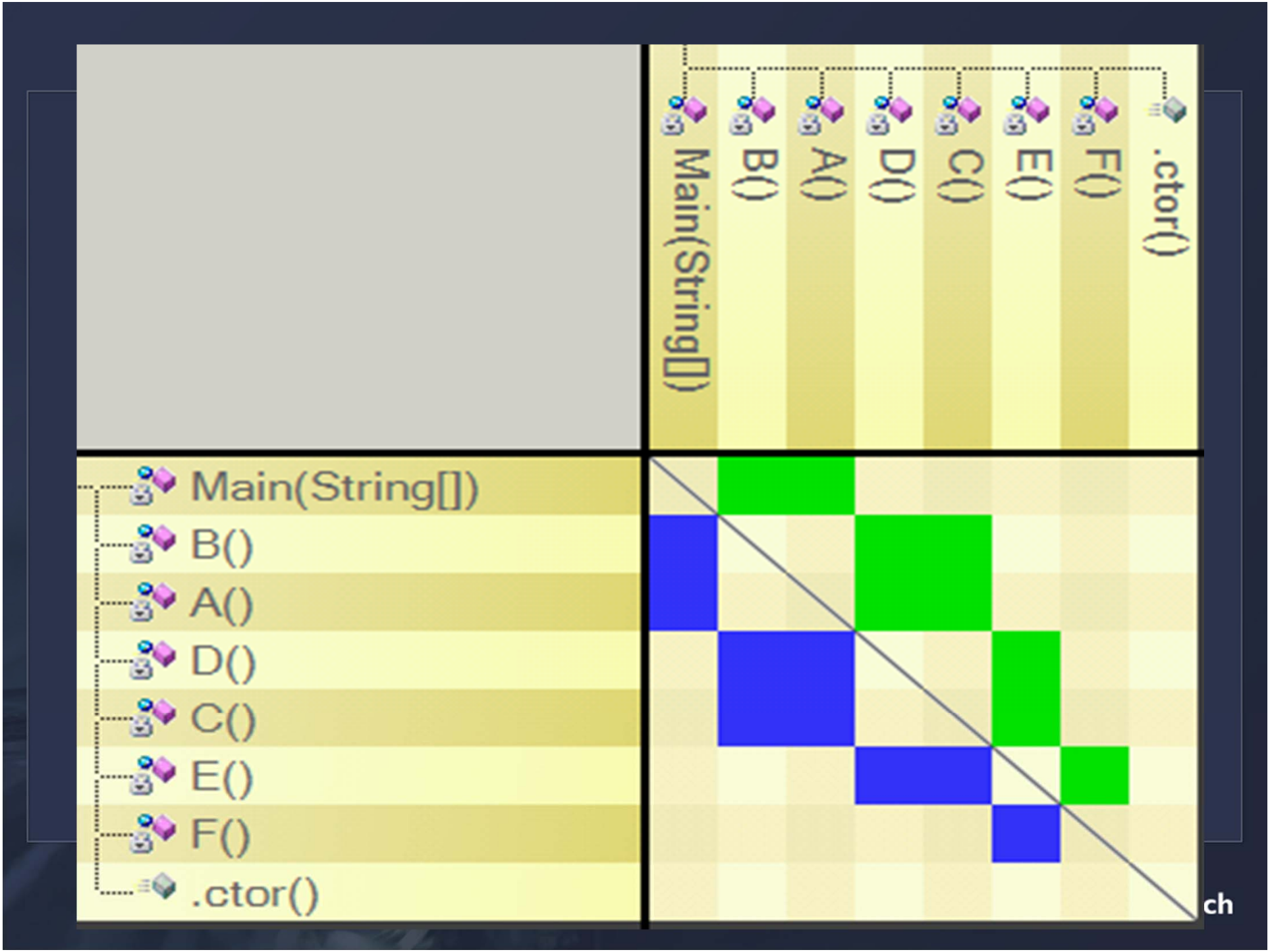
```
static void Main(string[] args)
{
    B();
    D();
}

private static void B()
{
    Console.WriteLine("B");
}

private static void D()
{
    Console.WriteLine("D"); AC();
    Console.WriteLine("B");
}

private static void AC()
{
    Console.WriteLine("AC");
}
```


.ctor(
AC()
B()
D()
Main
.ctor()
AC()
B()
D()
Main(String[])
1 2
1
1
1
2 1 1

Layered System

```
static void Main(string[] args)
{
    A();
    B();
}

private static void A() { C(); D(); }

private static void B() { C(); D(); }

private static void C() { E();}

private static void D() { E(); }

private static void E() { F(); }

private static void F() { }
```

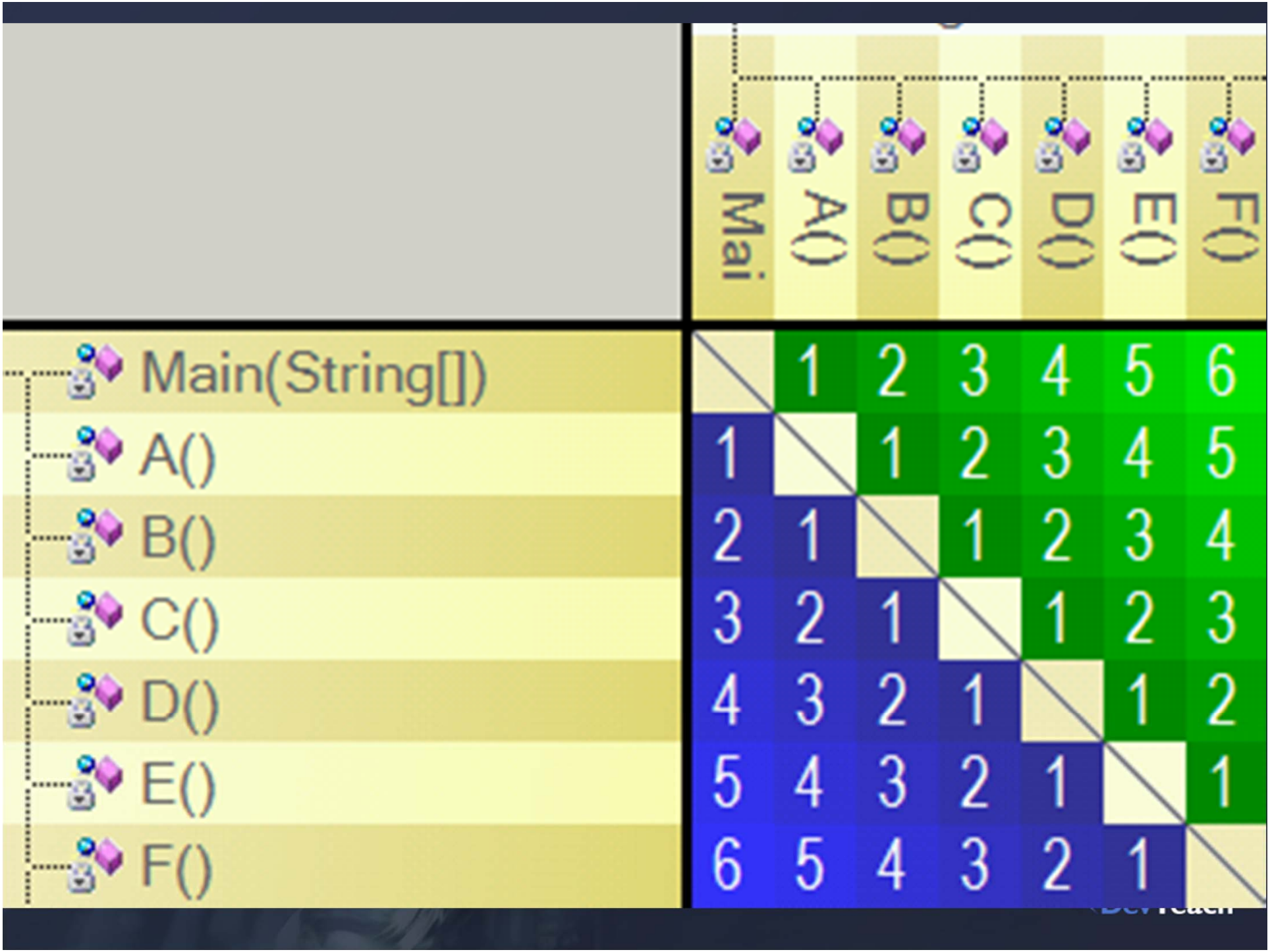

.ctor()
Main(String[])
A()
B()
C()
D()
E()
F()
1 1 2 2 3 4
1 1 1 2 3
1 1 1 2 3
2 1 1 1 2
2 1 1 1 2
3 2 2 1 1 1
4 3 3 2 2 1

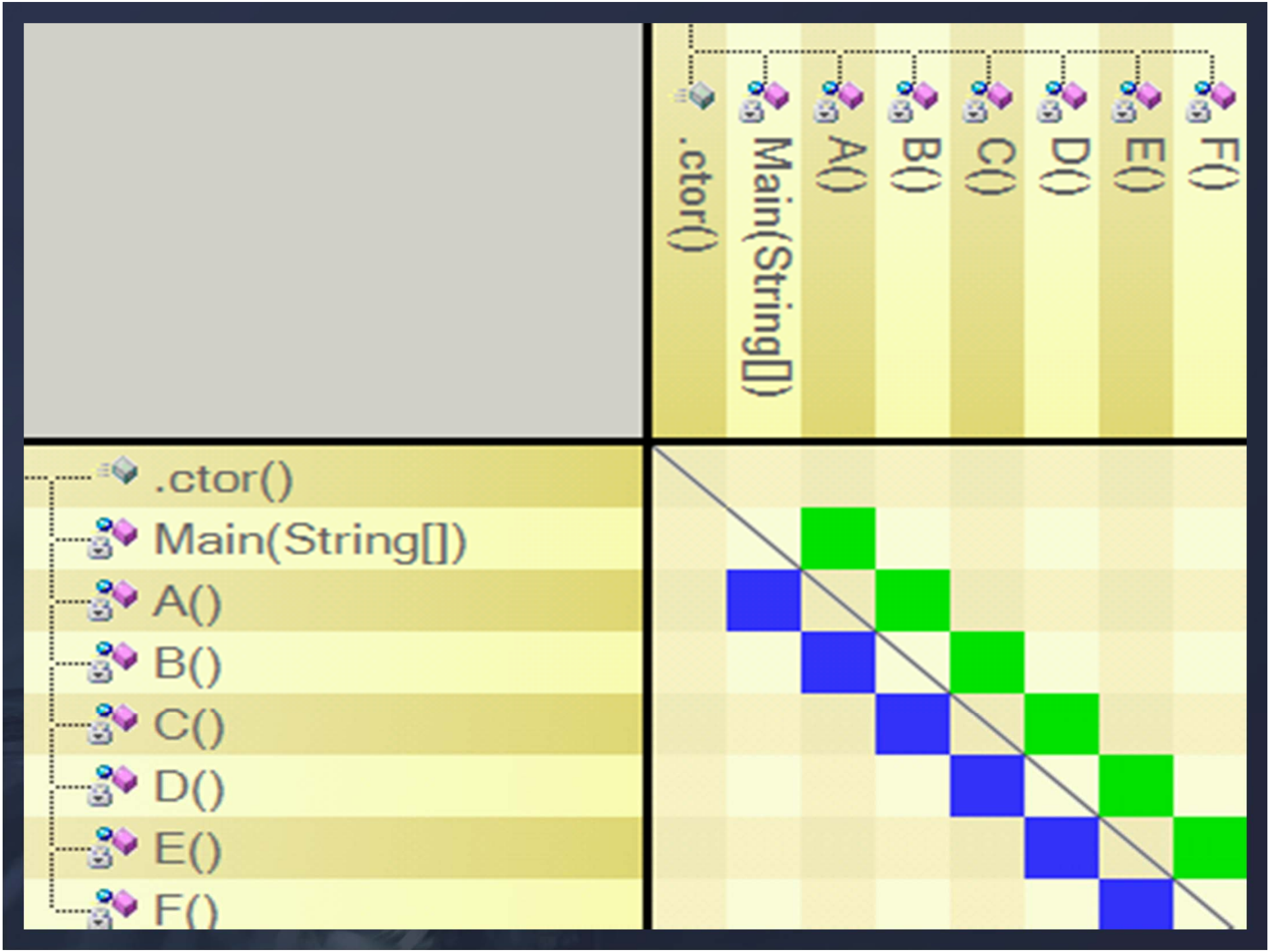

Main(String[])
B()
A()
D()
C()
E()
F()
.ctor()
Main(String[])
B()
A()
D()
C()
E()
F()
.ctor()

Strictly Layered (Hierarchical) System

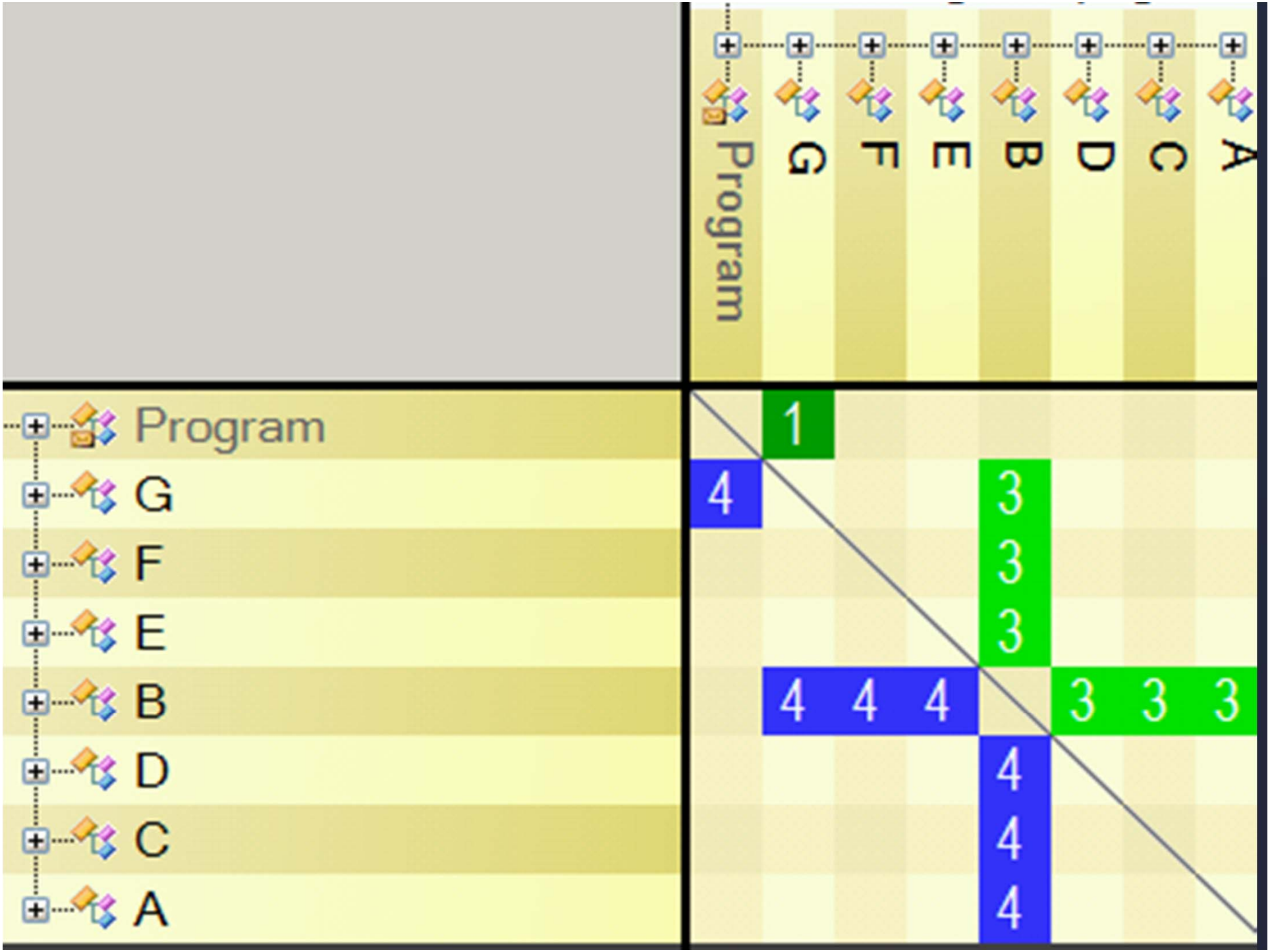
```
{
    class Program
    {
        static void Main(string[] args)
        {
            A();
        }

        private static void A() { B(); }

        private static void B() { C(); }

        private static void C() { D(); }

        private static void D() { E(); }

        private static void E() { F(); }

        private static void F() { }
    }
}
```

DevTeach

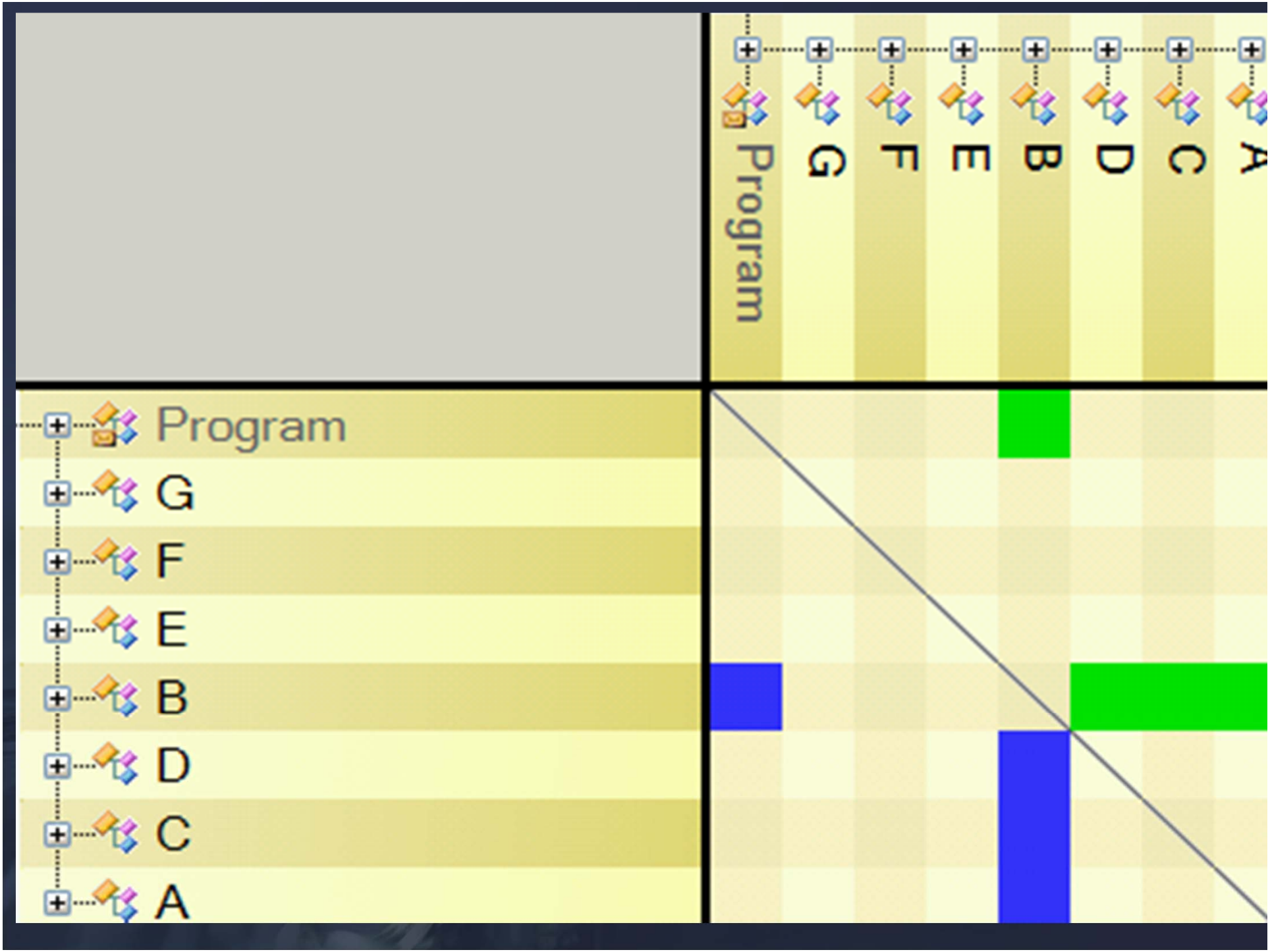

Mai
A()
B()
C()
D()
E()
F()
Main(String[])
A()
B()
C()
D()
E()
F()
1 2 3 4 5 6
1 1 2 3 4 5
2 1 1 2 3 4
3 2 1 1 2 3
4 3 2 1 1 2
5 4 3 2 1 1
6 5 4 3 2 1


.ctor()
Main(String[])
A()
B()
C()
D()
E()
F()

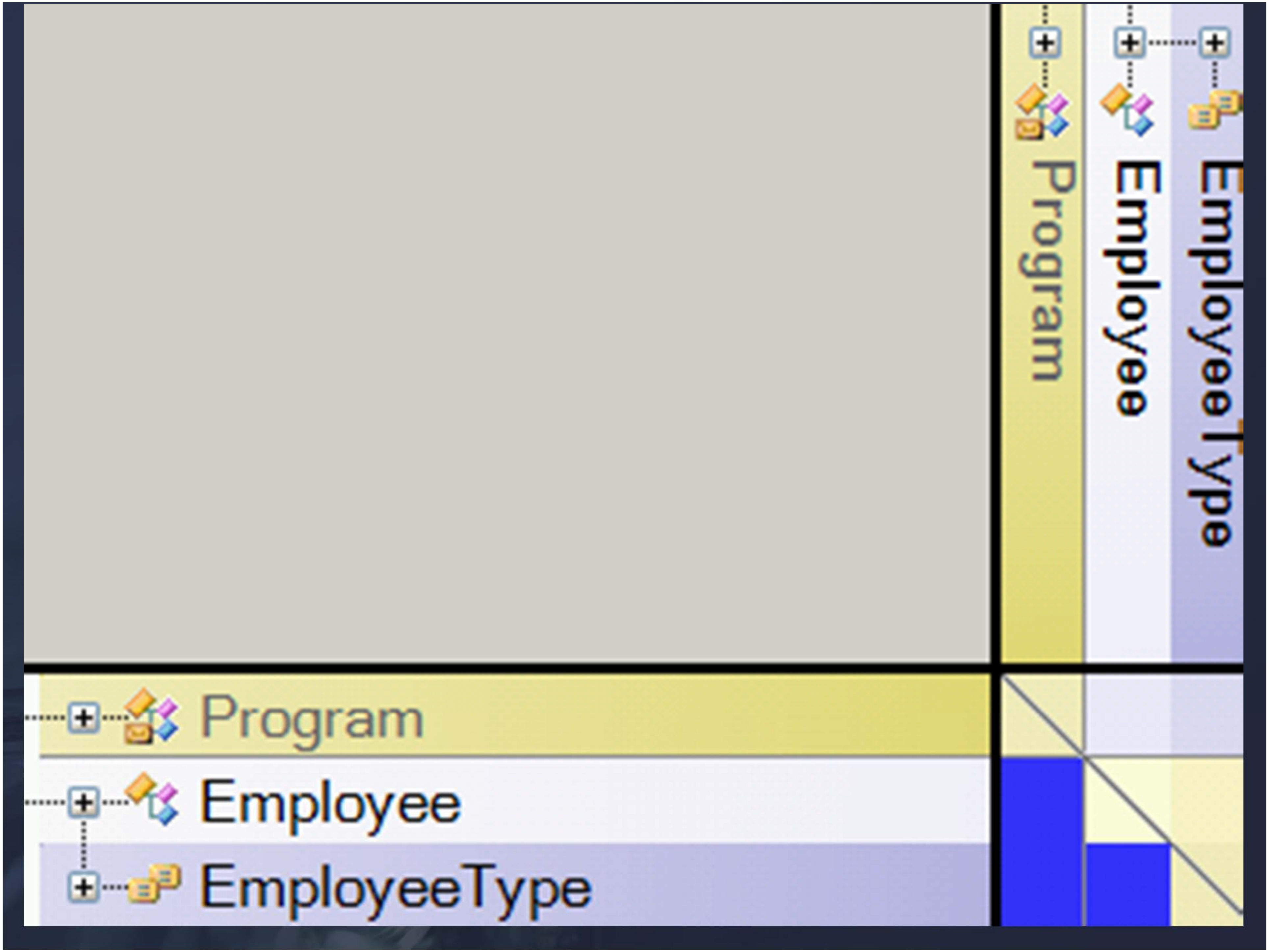
Change Propagator

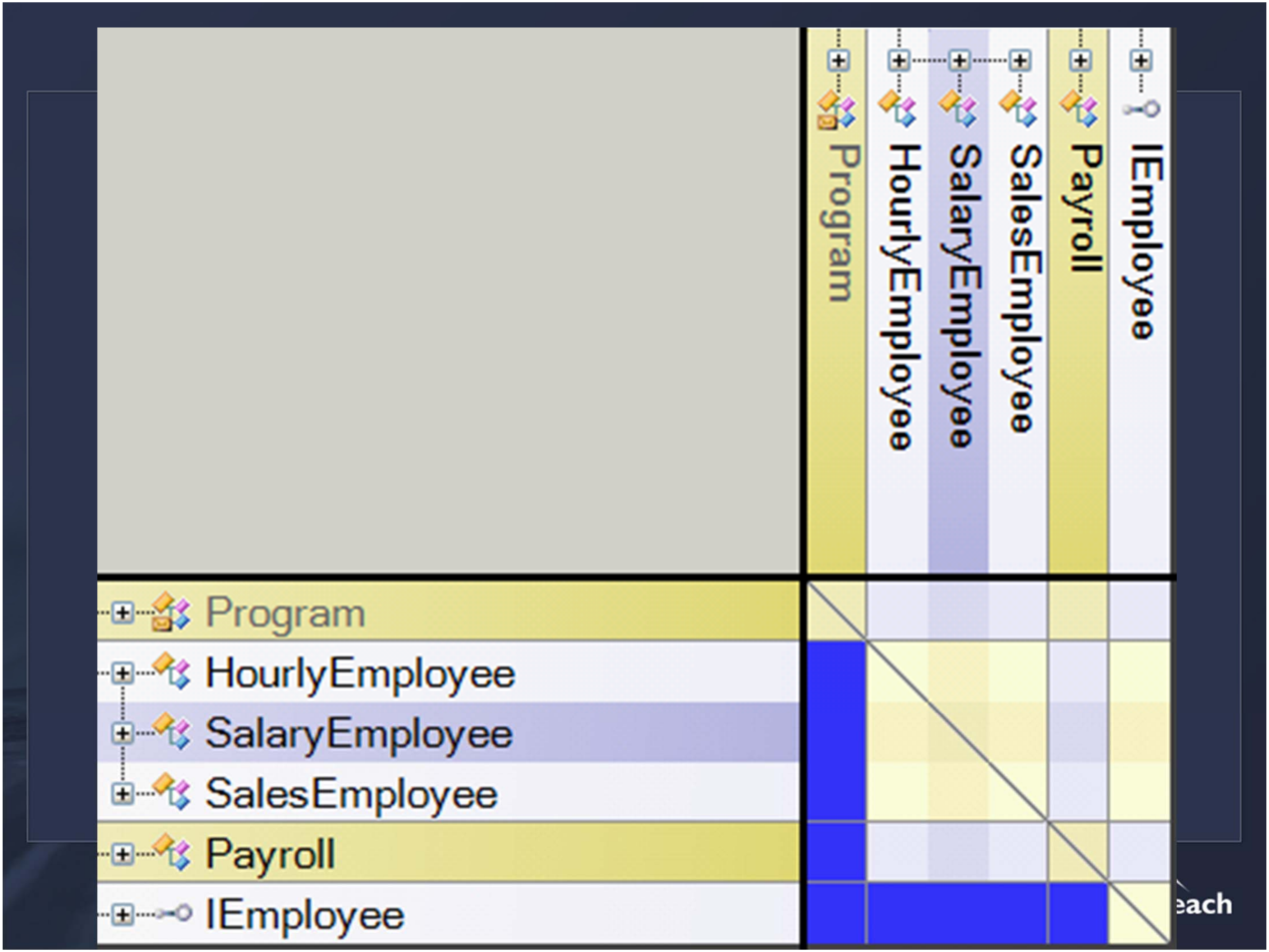

Program
G
F
E
B
D
C
A
Program
G
F
E
B
D
C
A
1
4
3
3
3
4 4 4
3 3 3
4
4
4

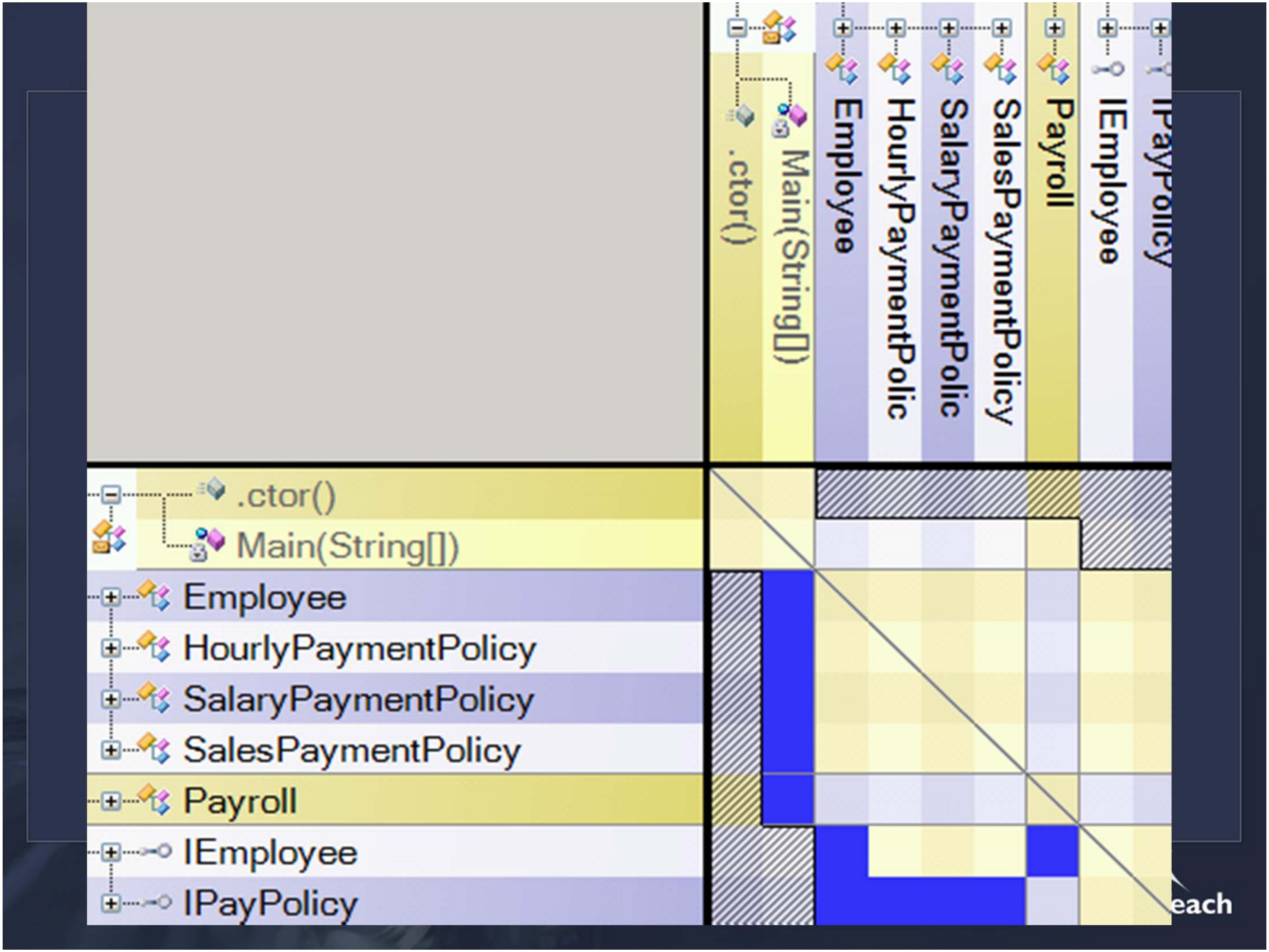
Subsystem


Program
G
F
E
B
D
C
A
Program
G
F
E
B
D
C
A

Simple Refactoring Example


Program
Employee
EmployeeType
Program
Employee
EmployeeType


Program
HourlyEmployee
SalaryEmployee
SalesEmployee
Payroll
IEmployee
Program
HourlyEmployee
SalaryEmployee
SalesEmployee
Payroll
IEmployee



.ctor()
Main(String[])
Employee
HourlyPaymentPolicy
SalaryPaymentPolicy
SalesPaymentPolicy
Payroll
IEmployee
IPayPolicy

Architectural Code Rules

Can Make Rules Break Build

```
warnif count > 0
        from n in Namespaces
        where n.IsUsing("DataLayer")
            && (n.Name == @"UILayer")
         select n
```

```
warnif count > 0 from n in Namespaces
        where n.IsUsing(“Layer3") && (n.Name == “Layer1")
        select n
warnif count > 0 from n in Namespaces
        where n.IsUsing(“Layer4") && (n.Name == "UILayer1")
        select n
warnif count > 0 from n in Namespaces
        where n.IsUsing("Layer5") && (n.Name == "Layer1")
        select n
```

Case Study of Large Project

LL Bean

Source: Eppinger and Browning*, Design Structure Matrix Methods and Applications*

Java

Lattix DSM

Functionally Like NDepend

Domain Independent

Domain Specific

Application Specific

100 Packages

3000 Classes

1 Million Lines of Code

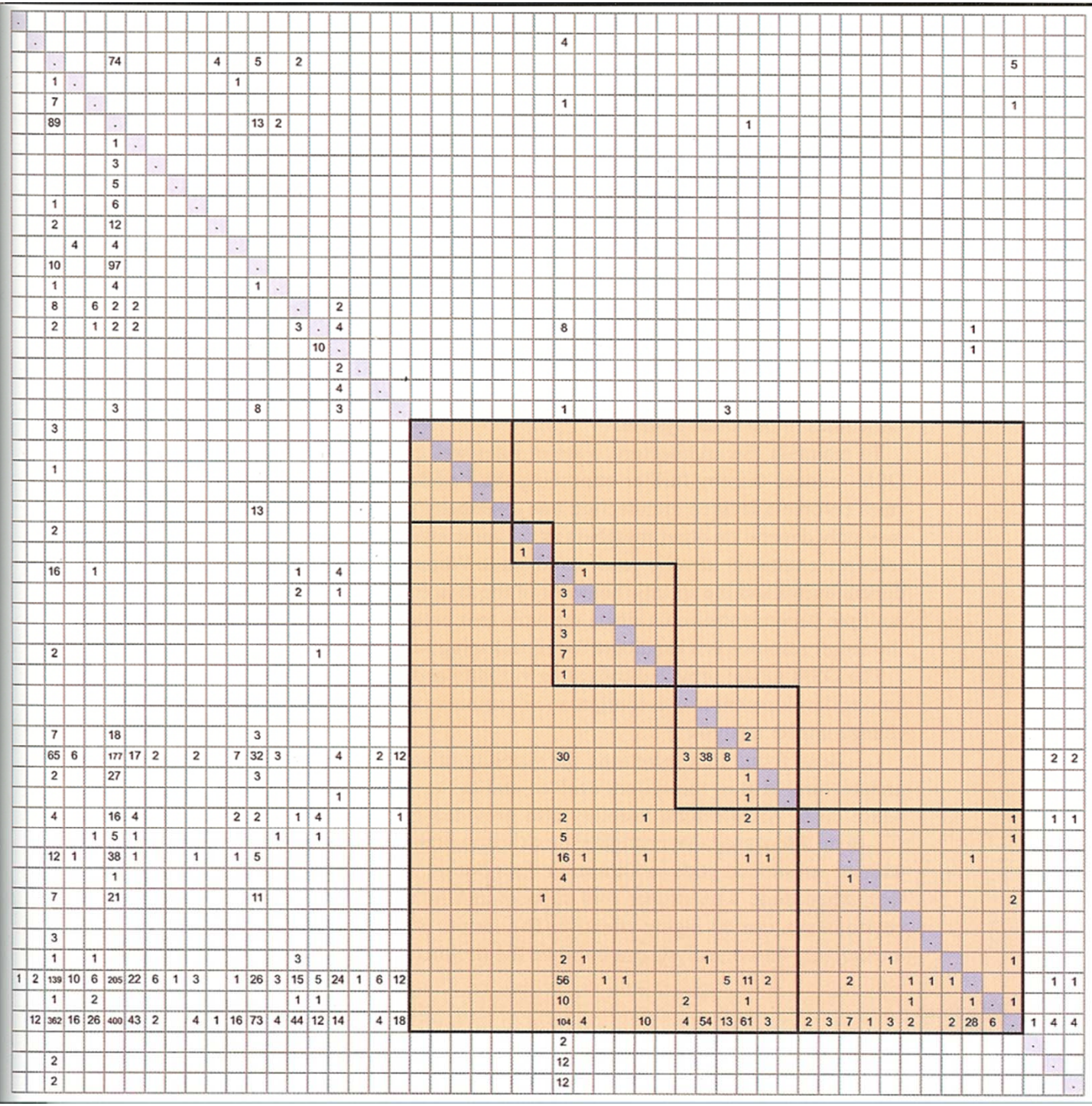
Partitioning in the Large Illustrates Patterns

Misplaced Common Types

Misplaced Inheritable Concrete Classes

Misplaced Catch-All Subsystems

Post Refactoring



Look For Patterns At Every Architecture Level

Maintained Architecture With Rules

Algorithm Search and Big Data

W. Edwards Deming
Theory of Profound Knowledge

System

Variation

Theory

Psychology

When you can measure what you are speaking about, and express it in numbers, you know something about it;

but when you cannot measure it, when you cannot express it in numbers, your knowledge of it is of a meager and unsatisfactory kind;

it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced it to the stage of science.

Lord Kelvin

In brief…

DSMs Provide Large Scale Visualization

Rules Can Be Used to Enforce Architectural Constraints