Towards upgradable ERP systems

Sebastien Vaucouleur
vaucouleur@itu.dk
IT University of Copenhagen

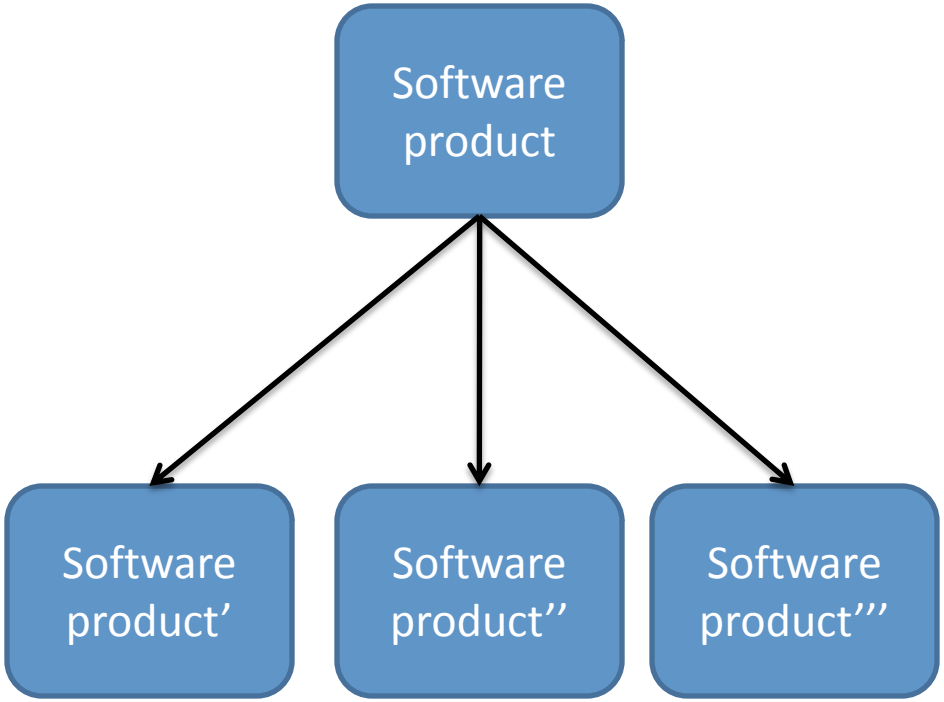
Context and contributors

- Evolvable software products project
- Software development group
- IT University of Copenhagen
- Microsoft Dynamics, DHI Denmark

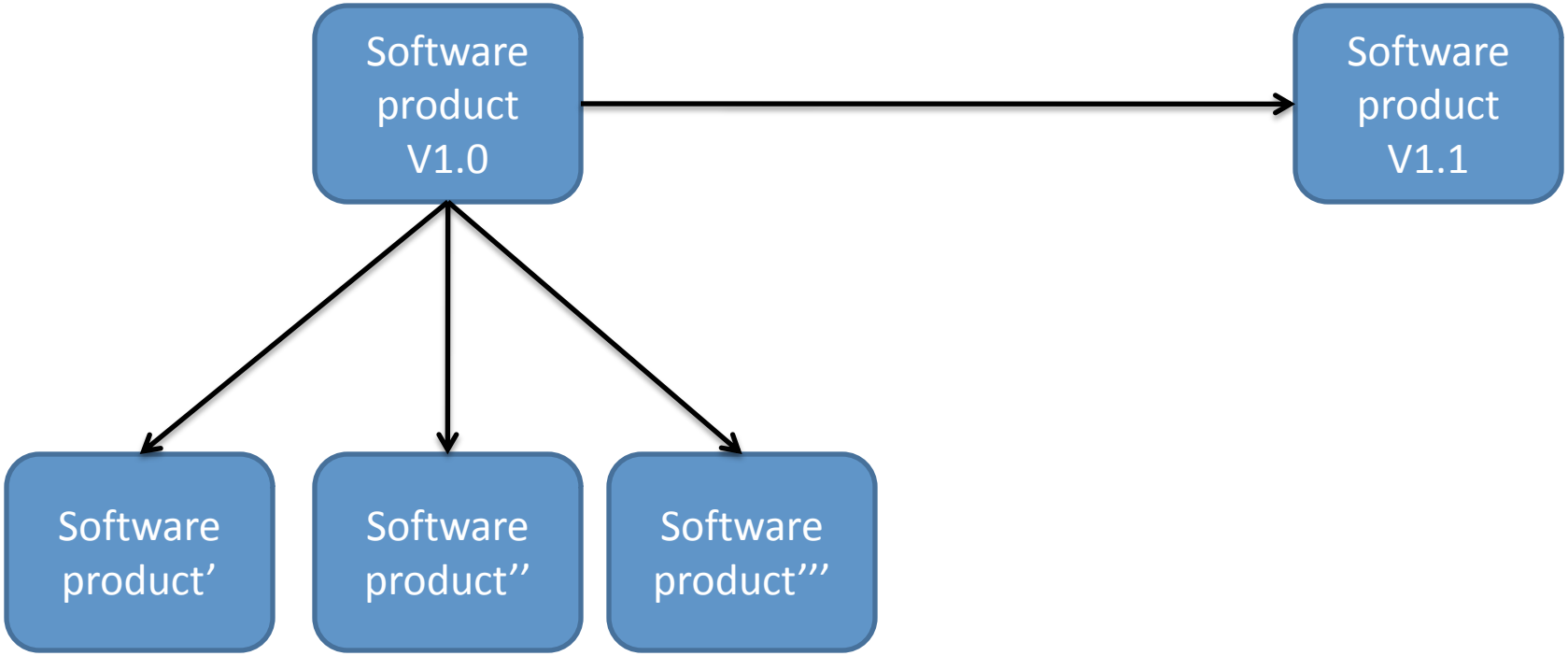
- Core ideas developed with A. Cisternino (University of Pisa)
- Contributions from my colleagues @ SDG group

Software Products

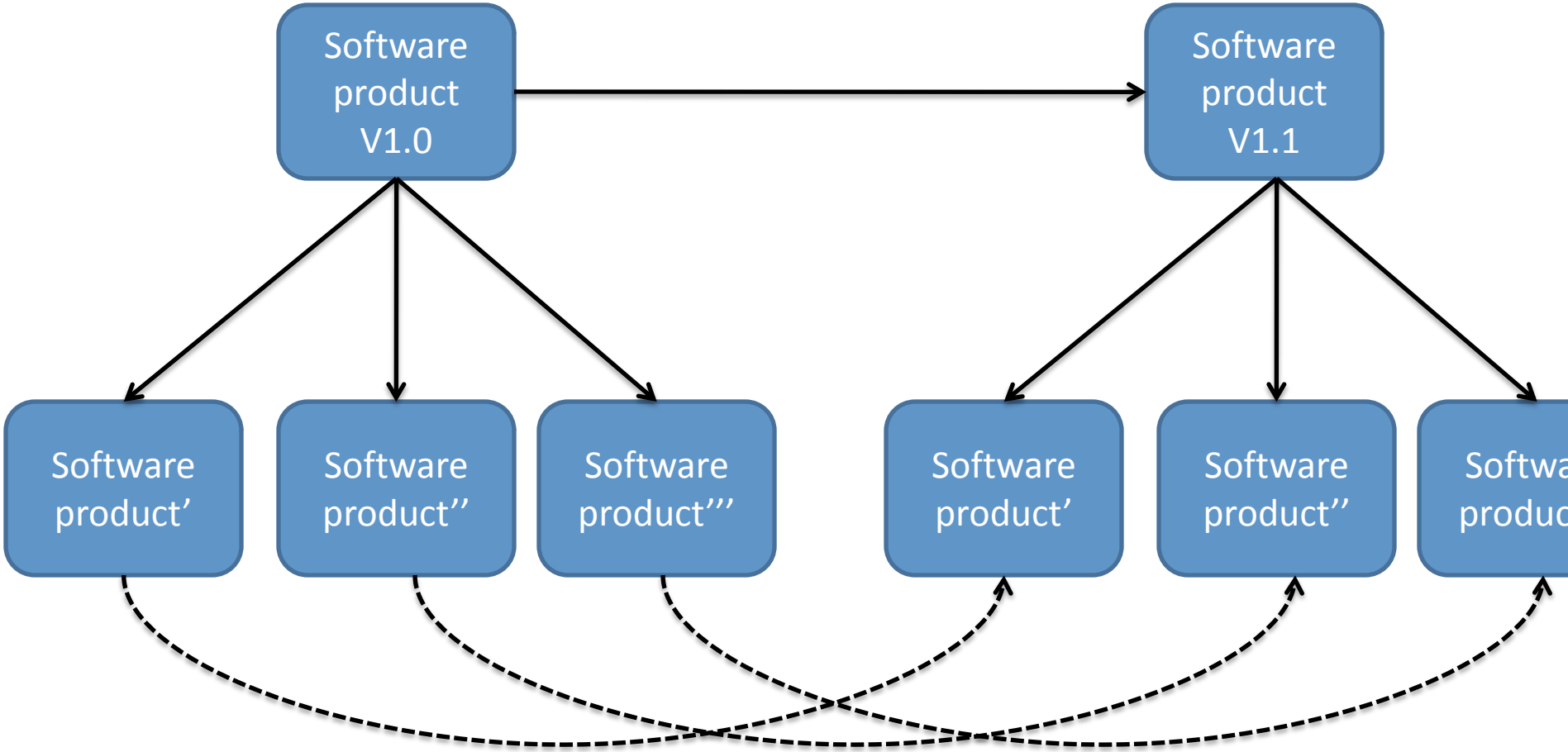
Software Products
(1st Problem)



Evolvable Software Products



Evolvable Software Products (2nd Problem)



Empirical grounds: Microsoft Dynamics Enterprise Resource Planning Systems

"Customizations that must be carried over from one version of enterprise software to the next are the biggest **technology headache and ROI killer** that CIOs face in upgrades"
[Beatty et al. 2006, Communication of the ACM].

- Y.Dittrich, S. Vaucouleur.
  "Customization and Upgrade of ERP systems".
  TR-2008-105

- Y.Dittrich, S. Vaucouleur.
  "Practices around Customization of Standard Systems".
  ICSE 2008

- P.Sestoft, S. Vaucouleur.
  "Evolvable Software Products",
  Springer LNCS Advances in Software Technology 2008

Summary of empirical results

- Online survey, 45 answers (world-wide)
- 3 Head-to-head interviews with partners (DK)

- Partners answers, summary
  1. Customizations cannot be anticipated
  2. Partners' staff typically have modest CS education
  3. Time to market is more important than correctness

Extremely simple running example

```
[…]
account1.Debit(x);
account2.Credit(x);
[…]
```

Running example: desired customization

```
[…]
account1.Debit(x * TAX);
account1.Debit(x);
account2.Credit(x);
[…]
```

Many instances

```
[…]
account1.Debit(x);
account2.Credit(x);
[…]
account3.Debit(y);
account4.Credit(y);
[…]
account5.Debit(z);
account6.Credit(z);
```

In-place modifications

```
[…]
account1.Debit(x * TAX);
account1.Debit(x);
account2.Credit(x);
[…]
account3.Debit(y * TAX);
account3.Debit(y);
account4.Credit(y);
[…]
account5.Debit(z * TAX);
account5.Debit(z);
account6.Credit(z);
```

Procedural abstraction

```
public virtual void Transaction(Account a, Account b, double x)
{
    a.Debit(x);
    b.Credit(x);
}
```

Named abstraction

```
public virtual void Transaction(Account a, Account b, double x)
{
    a.Debit(x);
    b.Credit(x);
}
[…]
Transaction(account1,account2,x);
[…]
Transaction(account2,account3,y);
[…]
```

Event solution

[…]
Some_ code;
Some_other_code;
Raise_transaction_event;
Yet_some_other_code;
Yet_Yet_some_other_code;
Raise_transaction_event;
More_ code;
[…]

The crystal ball assumption

- Decomposition into modules requires anticipation of likely changes (Parnas 70's)
- If A and B are likely to change together, put them in the same module

Problem/Solution space

Solutions	Customization without anticipation	Resilience to upgrades
In-place modifications	Yes	No

Problem/Solution space

Solutions	Customization without anticipation	Resilience to upgrades
In-place modifications	Yes	No
Usual abstraction-based techniques	No	Yes

Problem/Solution space

Solutions	Customization without anticipation	Resilience to upgrades
In-place modifications	Yes	No
Usual abstraction-based techniques	No	Yes
Our goal	Yes	Yes

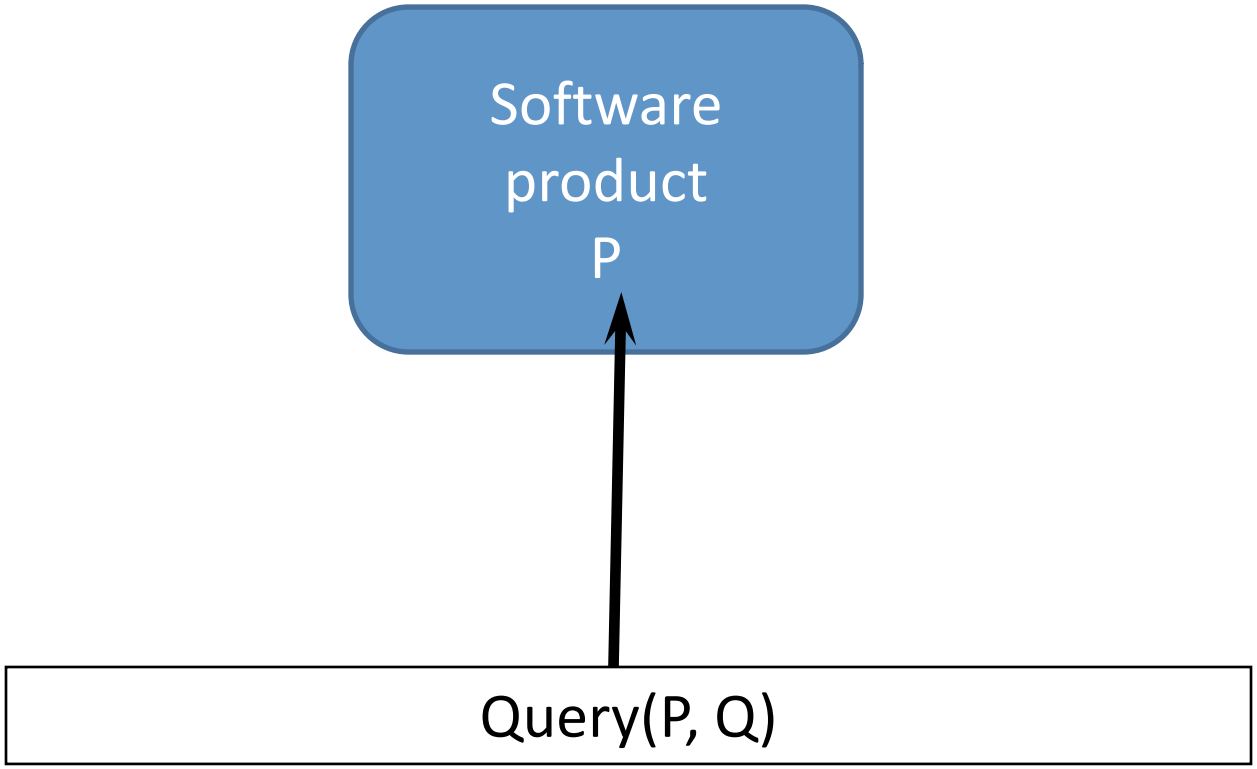
- Useful abstractions requires anticipation
- In-place modifications are sensitive to upgrades

Our approach: Glass-box

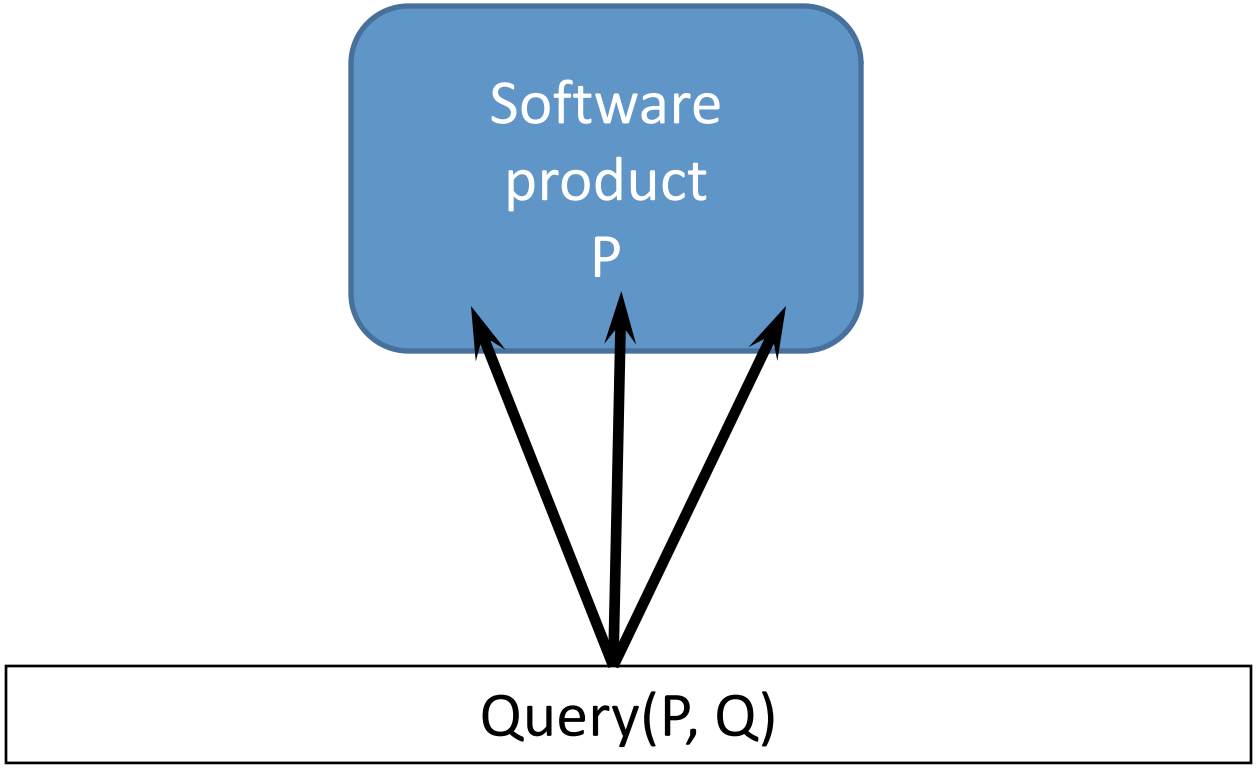
Partners write code queries

Query(P, Q)

Result = { CodeFragment }



{ CodeFragment1, CodeFragment2,CodeFragment3…}



Query language: Code query by example

Easy for non computer science experts

```
a.Debit(x);
b.Credit(x);
```

Typed formal arguments

```
public void DebitCredit(Account a, Account b, double x) {
  a.Debit(x);
  b.Credit(x);
}
```

Query methods defined using .NET attributes

```
[Query]
public void DebitCredit(Account a, Account b, int x) {
  a.Debit(x);
  b.Credit(x);
}
```

Named query method (abstraction)

```
[Query("Transaction")]
public void DebitCredit(Account a, Account b, int x) {
  a.Debit(x);
  b.Credit(x);
}
```

Query class

```
public class Partner1 {
  [Query(“Transaction”)]
  public void DebitCredit(Account a, Account b, int x) {
    a.Debit(x);
    b.Credit(x);
  }
}
```

Existing Software Product (no markers)

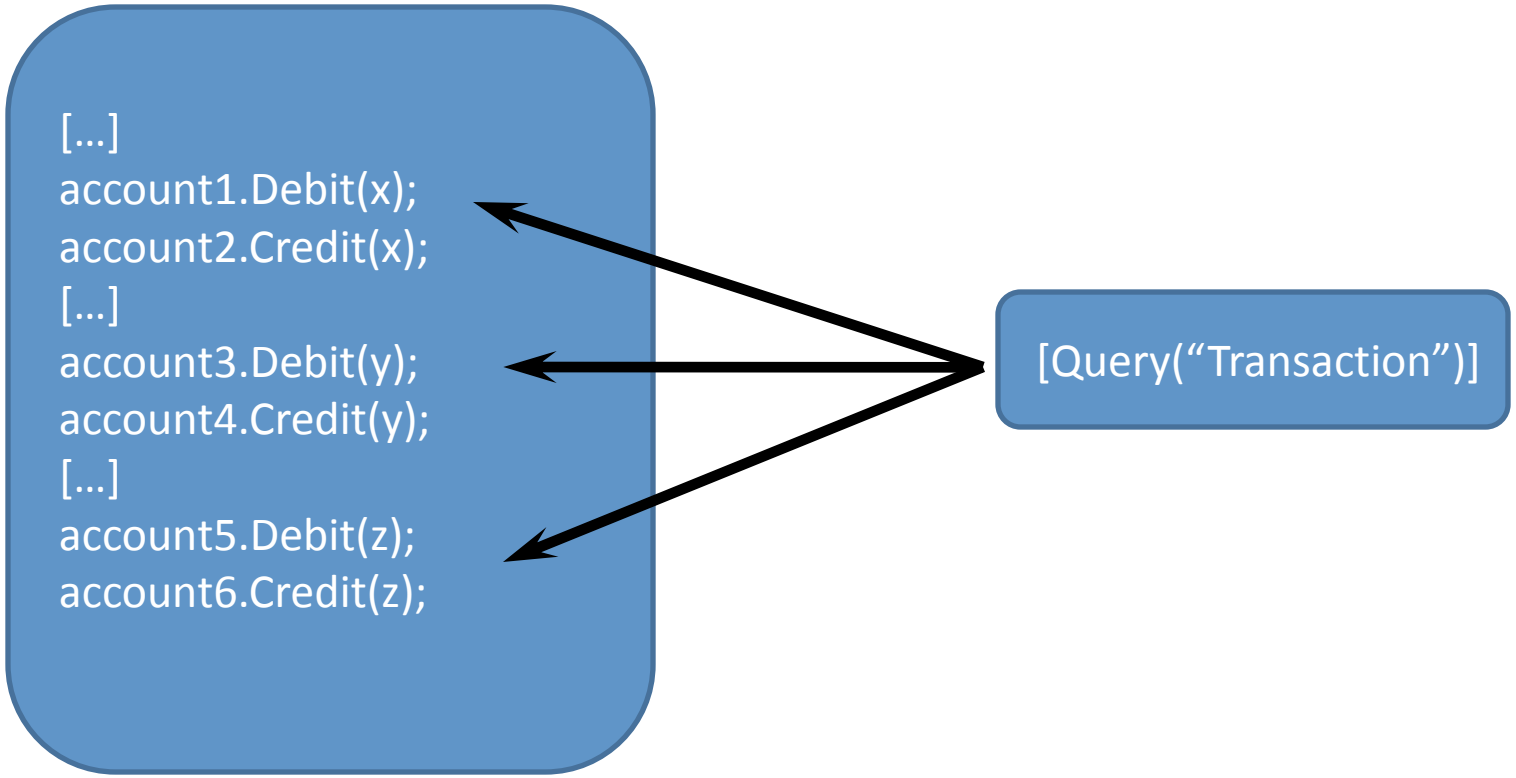
```
[…]
account1.Debit(x);
account2.Credit(x);
[…]
account3.Debit(y);
account4.Credit(y);
[…]
account5.Debit(z);
account6.Credit(z);
```

A set of queries

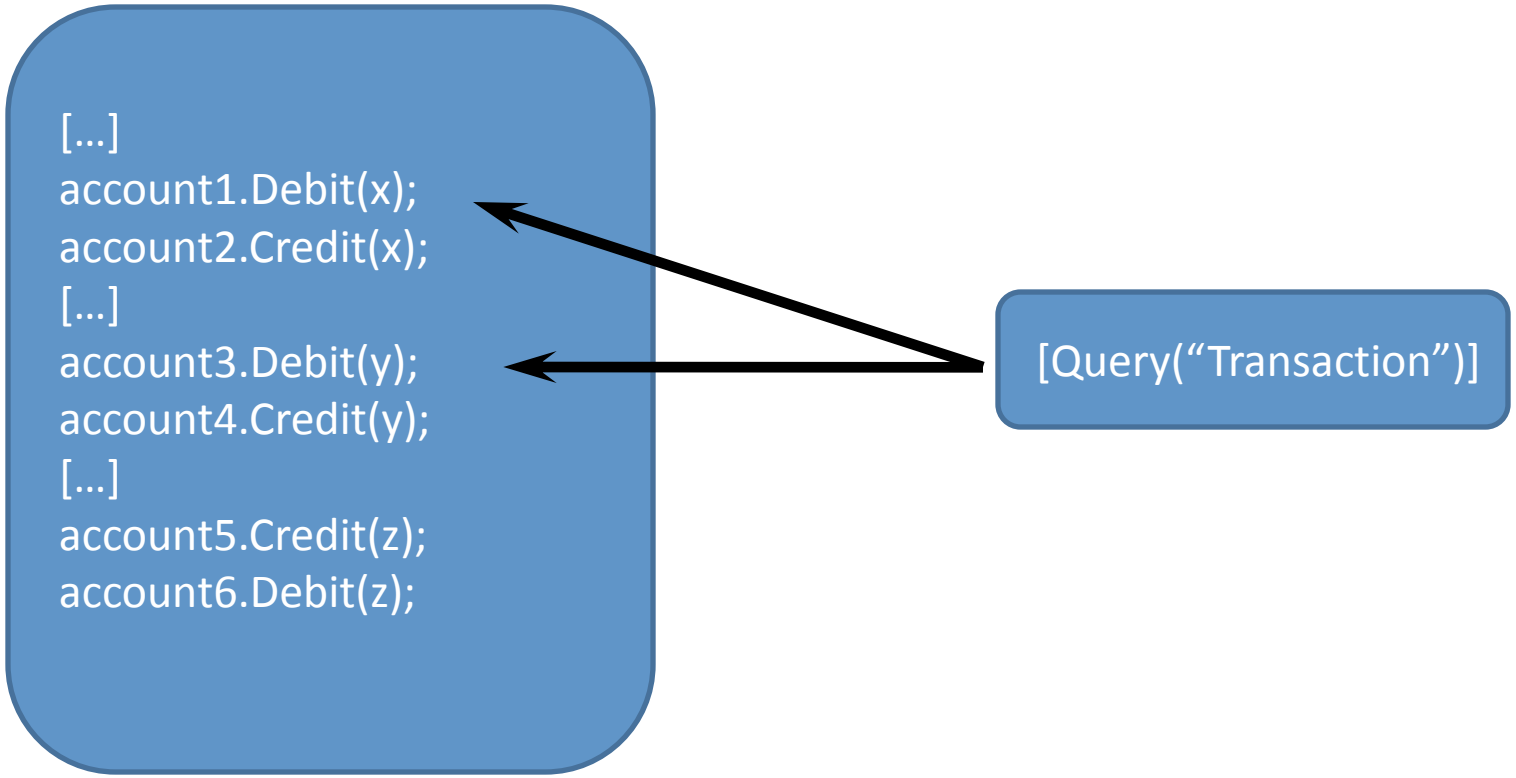
```
[…]
account1.Debit(x);
account2.Credit(x);
[…]
account3.Debit(y);
account4.Credit(y);
[…]
account5.Debit(z);
account6.Credit(z);
```

```
[Query("Transaction")]
```

Match: SoftwareProduct x Query -> CodeFragment



Debit; Credit | Credit;Debit



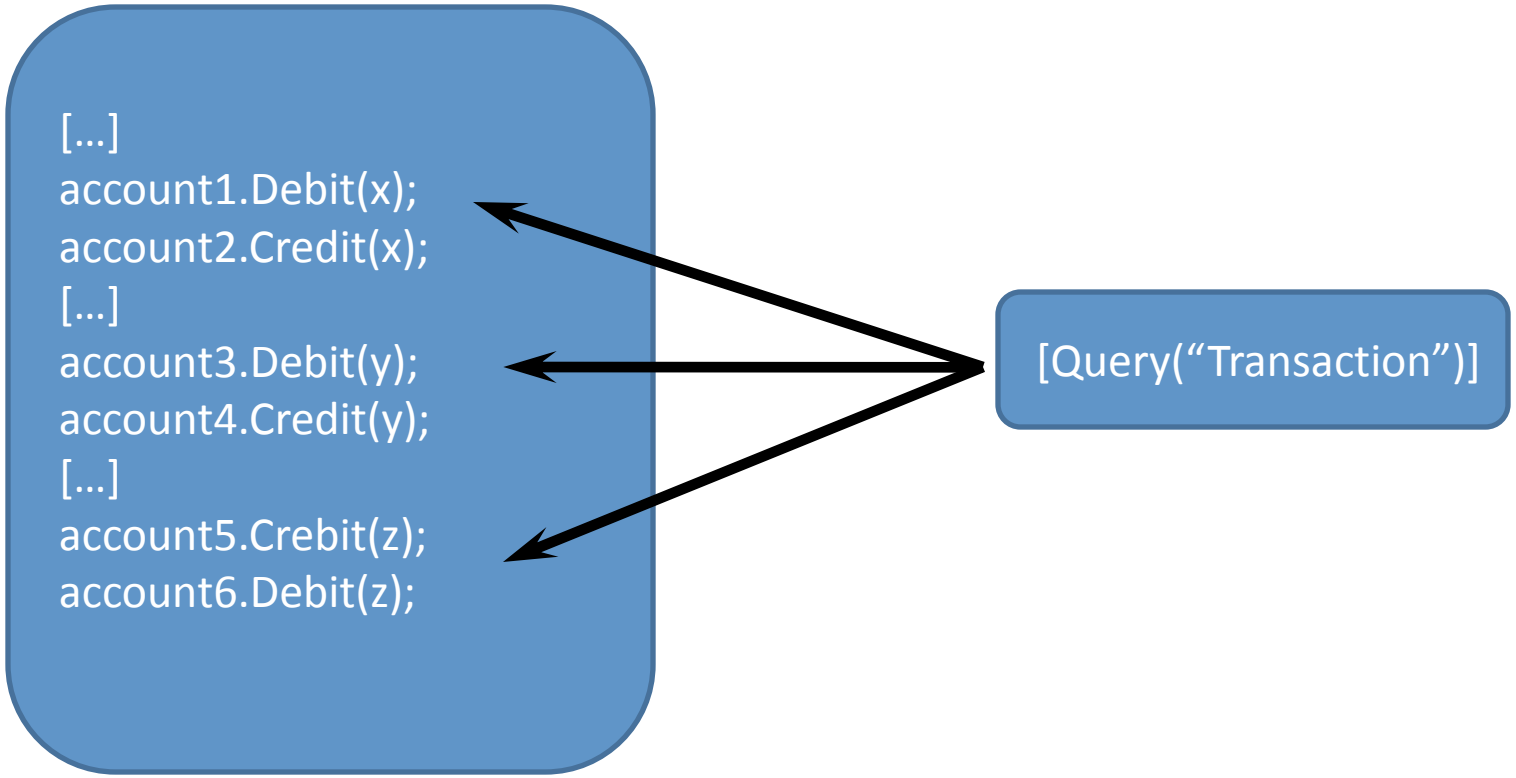
Back to the code query

```
public class Partner1 {
  [Query(“Transaction”)]
  public void DebitCredit(Account a, Account b, double x) {
    a.Debit(x);
    b.Credit(x);
  }
}
```

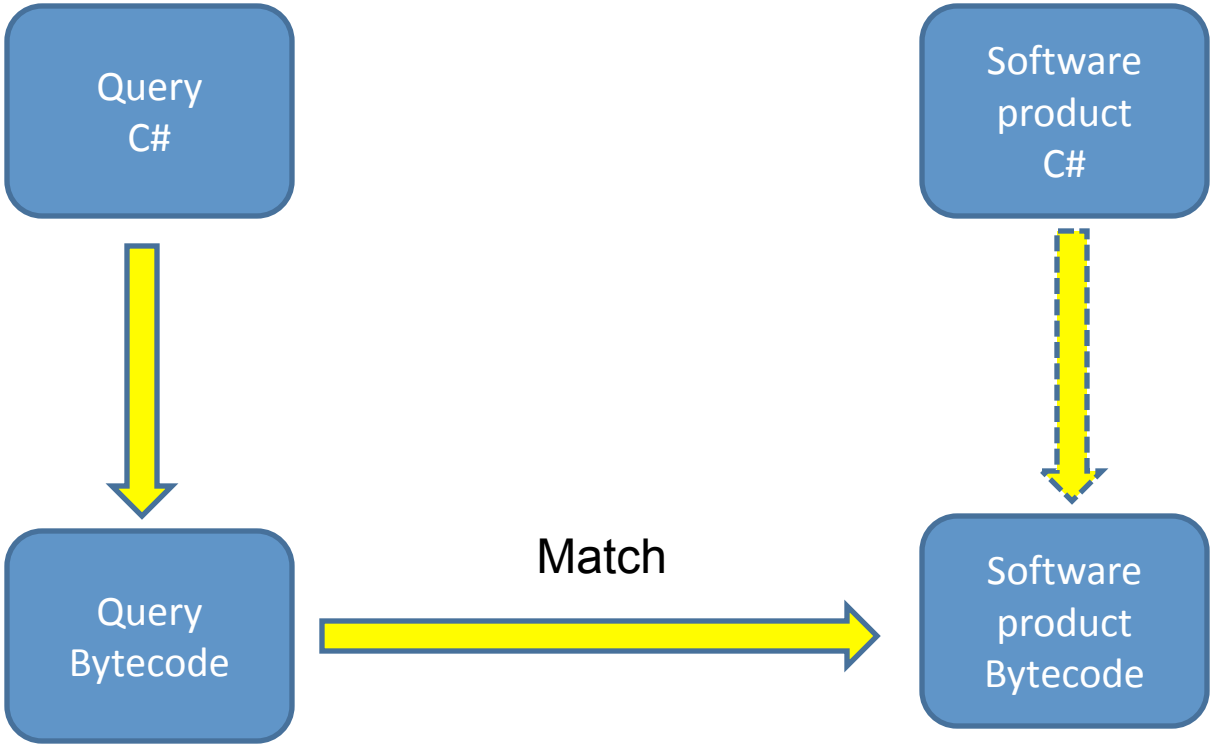
Add a new query method

```
public class Partner1 {
  [Query(“Transaction”)]
  public void DebitCredit(Account a, Account b, double x) {
    a.Debit(x);
    b.Credit(x);
  }
 [Query(“Transaction”)]
  public void CreditDebit(Account a, Account b, double x) {
    a.Crebit(x);
    b.Debit(x);
  }
}
```

Match

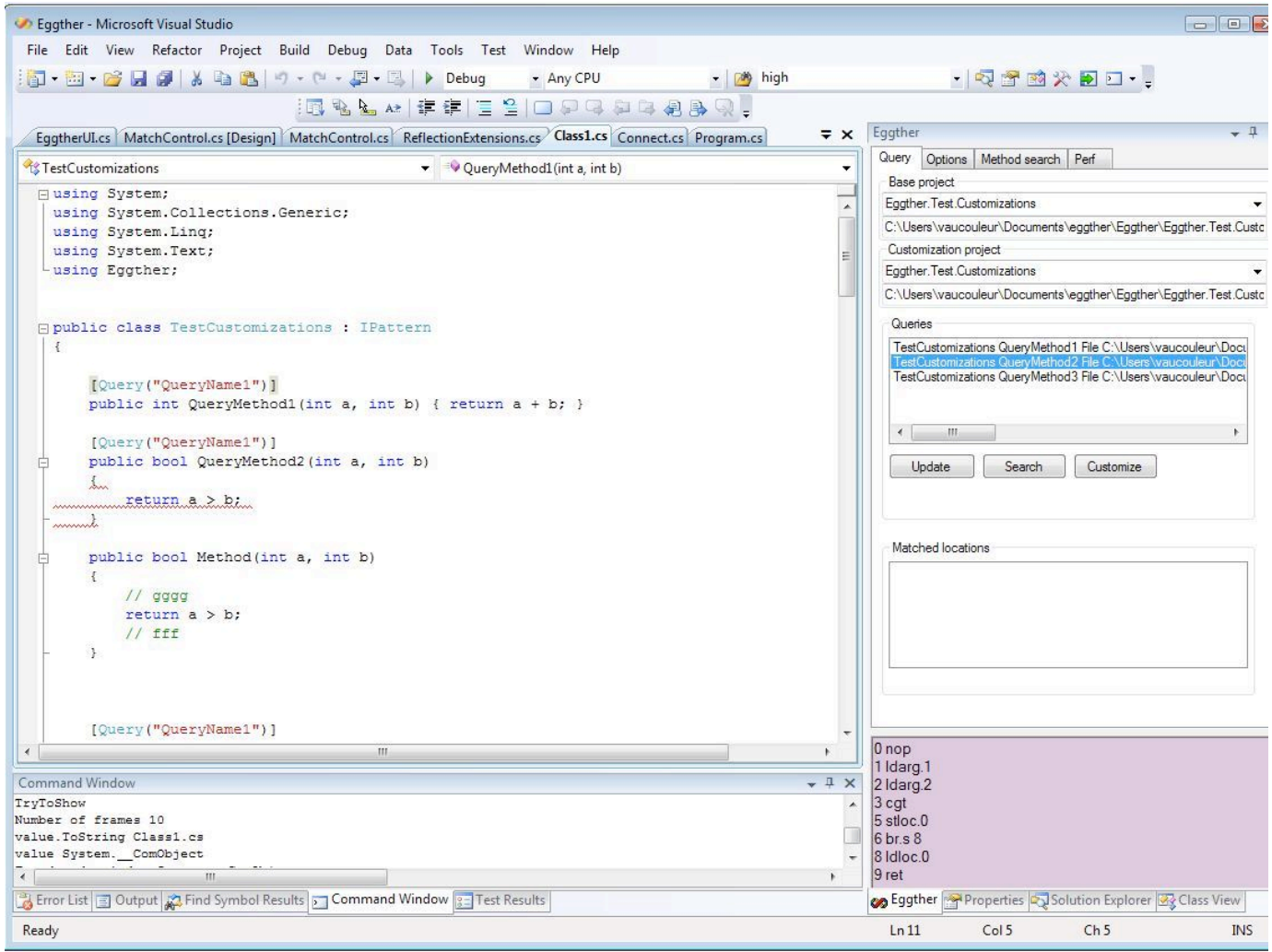

[…]
account1.Debit(x);
account2.Credit(x);
[…]
account3.Debit(y);
account4.Credit(y);
[…]
account5.Crebit(z);
account6.Debit(z);
[Query(“Transaction”)]

Implementation approach: Query

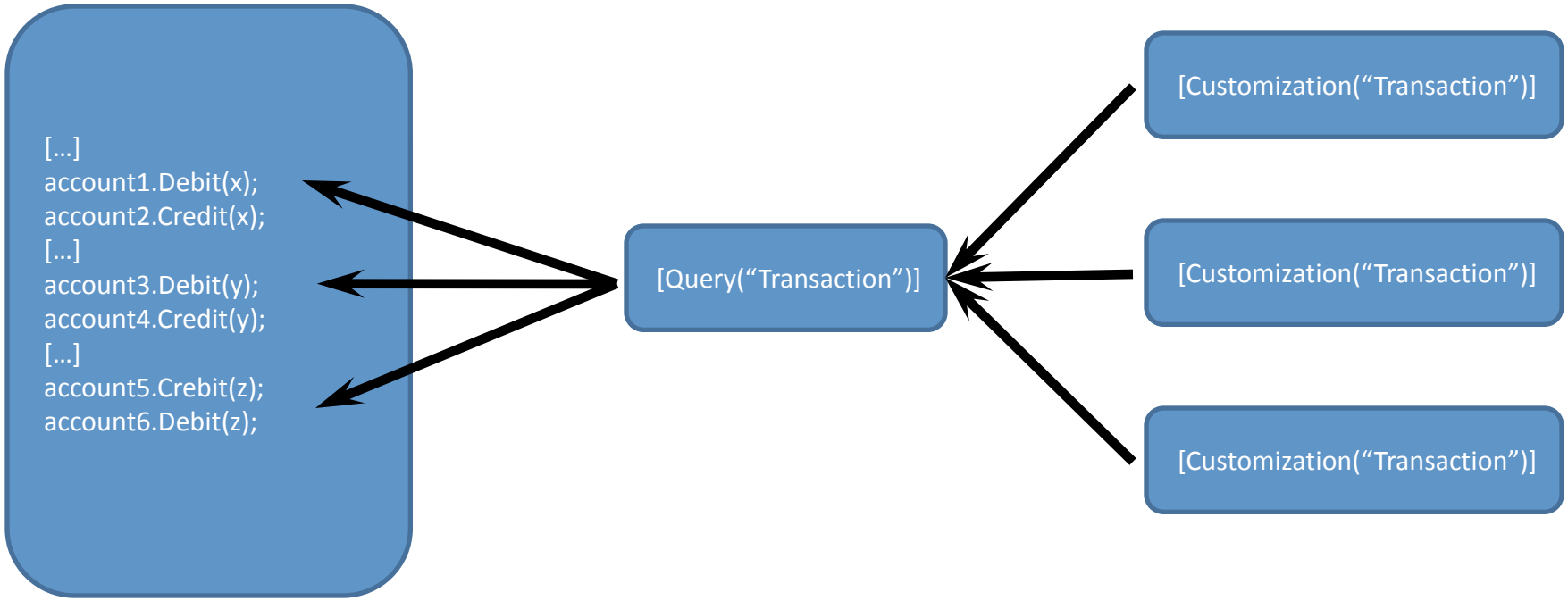


Tool support Visual Studio plugin

- Show result of code queries (at design time)



Part 2: Customization



Customization

[Customization(“Transaction”)]

```
[Customization("Transaction")]
 static public void Tax(….) {

 }
```

```
public class CustomizationDenmark {
  [Customization(“Transaction”)]
   static public void Tax(….) {

   }
}
```

```
  [Customization("Transaction")]
   static public void Tax(….) {

   }
}
```

Binding

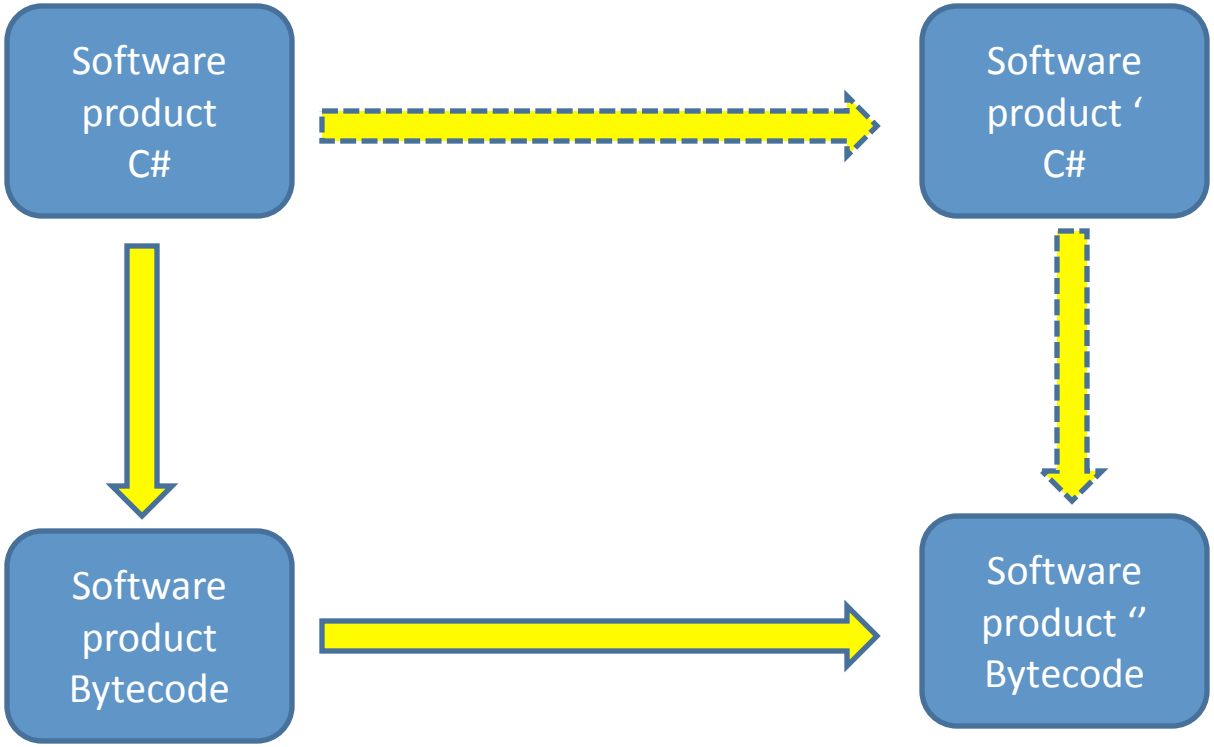
```
[Customization("Transaction")]
 static public void Tax(Account a, Account b, double x) {
    a.Debit(x * TAX);
 }

[Query("Transaction")]
public void DebitCredit(Account a, Account b, double x) {
  a.Debit(x);
  b.Credit(x);
}
```

```
[…]
CustomizationDenmark.Tax(a, b, x);
a.Debit(x);
b.Credit(x);
[…]
```

Upon upgrade

- Developers re-apply code queries
  - Code query find the new locations
  - Developer inspect the results & modify the queries to fit the new version

Implementation approach: Customization



- Pros
  - Can use standard C# compilers
    - Fast
    - Supported by Microsoft (language evolution)
  - Can use standard IDE (Visual Studio)
    - Syntax highlighting, Refactoring tools, etc.
    - Static typing at design time within IDE
  - Support for multiple languages
- Cons
  - Bytecode instrumentation can be tricky
  - Some information is lost during compilation
  - Problem with context sensitive optimizations?

Thanks, Questions ?

http://eggther.org