TechED BoF-12

『プログラミング! プログラミング! プログラミング!
.NET 3.5 時代のコーディング
～これからの実装技術について考えよう～』

プログラミングはどう変わるか?

小島富治雄 (Fujiwo)

Microsoft® tech·ed 2008

今日話したいこと

- .NET 3.5 時代のプログラミングの進化
- プログラミングはどう変わるか 中級編
- LINQ周りを中心に

Agenda

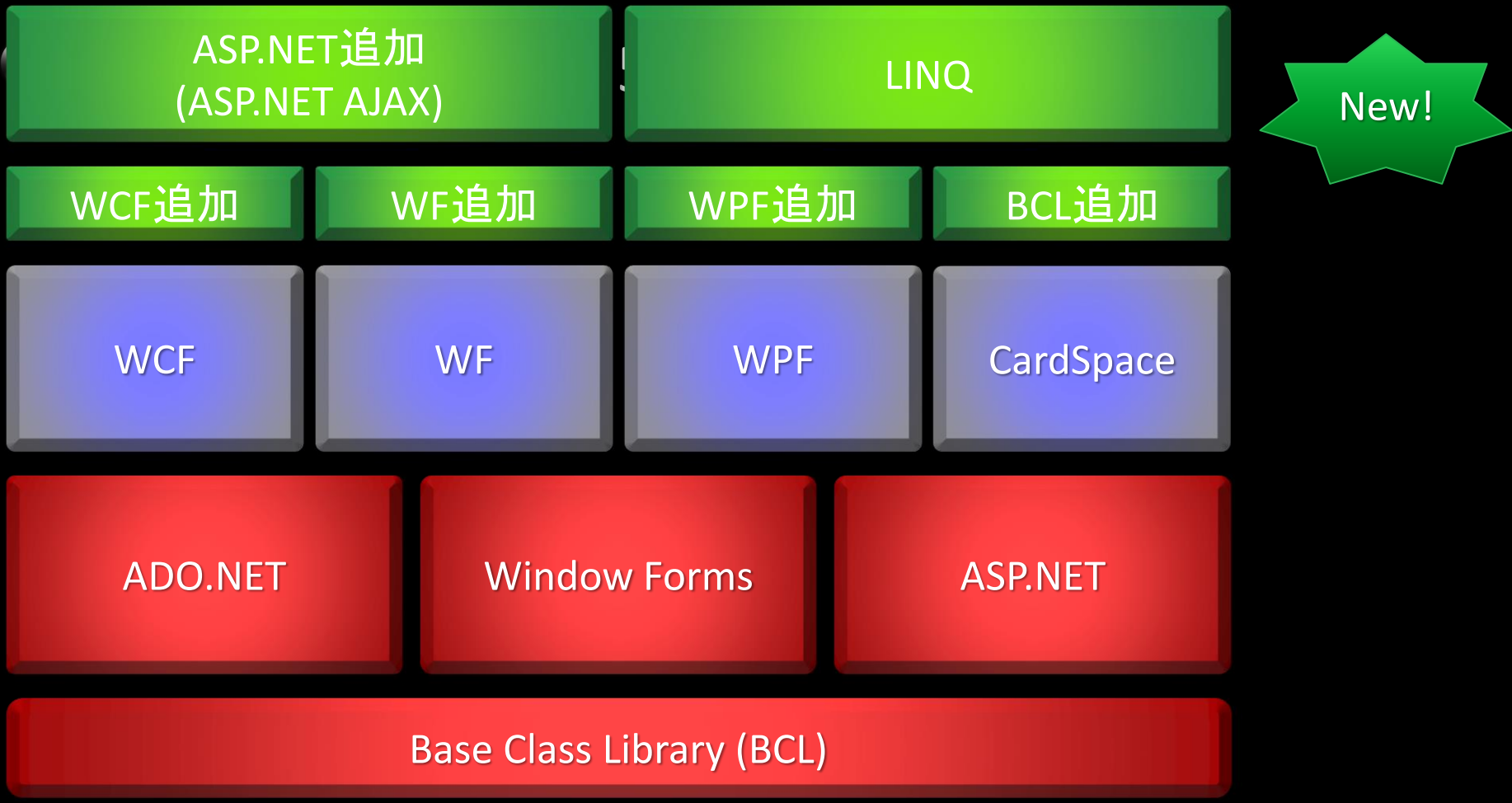
1. .NET の進化
2. Demo
3. 美しいプログラムについて

1. .NET の進化

.NET Framework の進化

	1.0	1.1	2.0	3.0	3.5
ランタイム (CLR)	1.0.3705	1.1.4322	2.0.50727 832		2.0.50727 1433
トピック	ADO.NET ASP.NET WinForm	ASP.NET1.1	ADO.NET 2.0 ASP.NET 2.0 C# 2.0 VB 8.0	WPF WF WCF CardSpace	ASP.NET AJAX LINQ C# 3.0 VB 9.0
VS2002	◎				
VS2003		◎			
VS2005			◎	△	
VS2008			○	○	◎
サポート期限	**2007/07/10**	**2008/10/14**	**2011/01/12**	**2012/04/10**	**----**

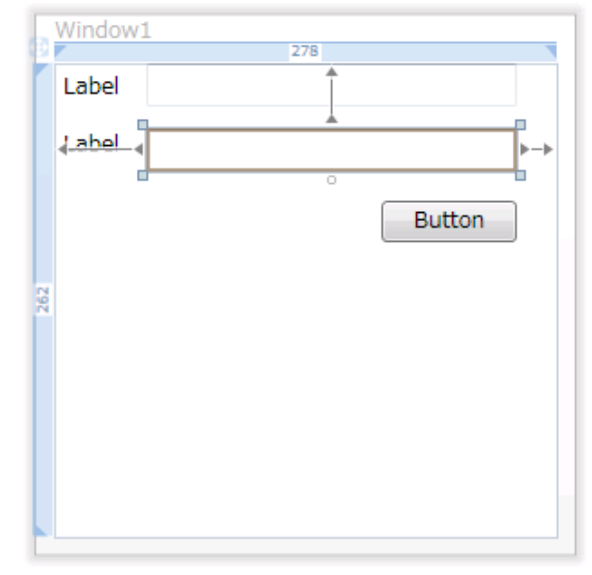
.NET Framework 全体像



ライブラリの拡張・強化!

.NET 3.5 時代のプログラミングで変わること

- IDE の進化
  - 各種図解的言語のデザイナ
    - WPF
    - データ モデル (LINQ to SQL など)
    - ワークフロー



.NET 3.5 時代のプログラミングで変わること

- LINQ (Language-INtegrated Query: 言語に統合されたクエリ)
  - C#3.0/Visual Basic9.0
  - LINQ プロバイダ
    - LINQ to Object
    - LINQ to SQL
    - LINQ to DataSet
    - LINQ to XML

.NET 3.5時代のプログラミングで変わること

更なるマルチ パラダイム化

- 1.0 の頃…
  - 手続き型
  - オブジェクト指向
  - コンポーネント指向
- 2.0 以降
  - ジェネリック
  - 関数型
  - より動的に
  - DSL (Domain Specific Language: ドメイン特化言語) の進化

例. C# 1.0 → 2.0、3.0 によるプログラミングの進化

- オブジェクトへの委譲 → メソッドへの委譲
  - class → delegate → 匿名メソッド (クロージャ) → ラムダ式
- yield による継続
- これらと拡張メソッドによるメソッド チェーン

Anders Hejlsberg

Anders Hejlsberg

Anders Hejlsberg 談: C#の今後について

2006/02/02 at 横浜

Q. 今後C#は、関数型言語として進化していくのか?

A. Yes. C#3.0 や LINQ で導入された「ラムダ式」などの機能は、Haskell や ML などの関数型言語に触発されたものだ。
これらの機能は開発をもっと自由な形にする。設計しているだけでワクワクするような機能だ。C# 3.0というのは、オブジェクト指向言語と関数型言語の「**ハッピーな結婚**」といってよいものになるだろう。

2. Demo

世界のなべあつ

「3の倍数と3の付く数字だけアホになります」

例. 世界のなべあつ

- 手続き型パラダイムなべあつ
- マルチパラダイムなべあつ
- LINQ なべあつ

手続き型なべあつ

手続き型なべあつ

「3の倍数と3の付く数字だけ
アホになり、
5の倍数だけ
犬っぽくなります」

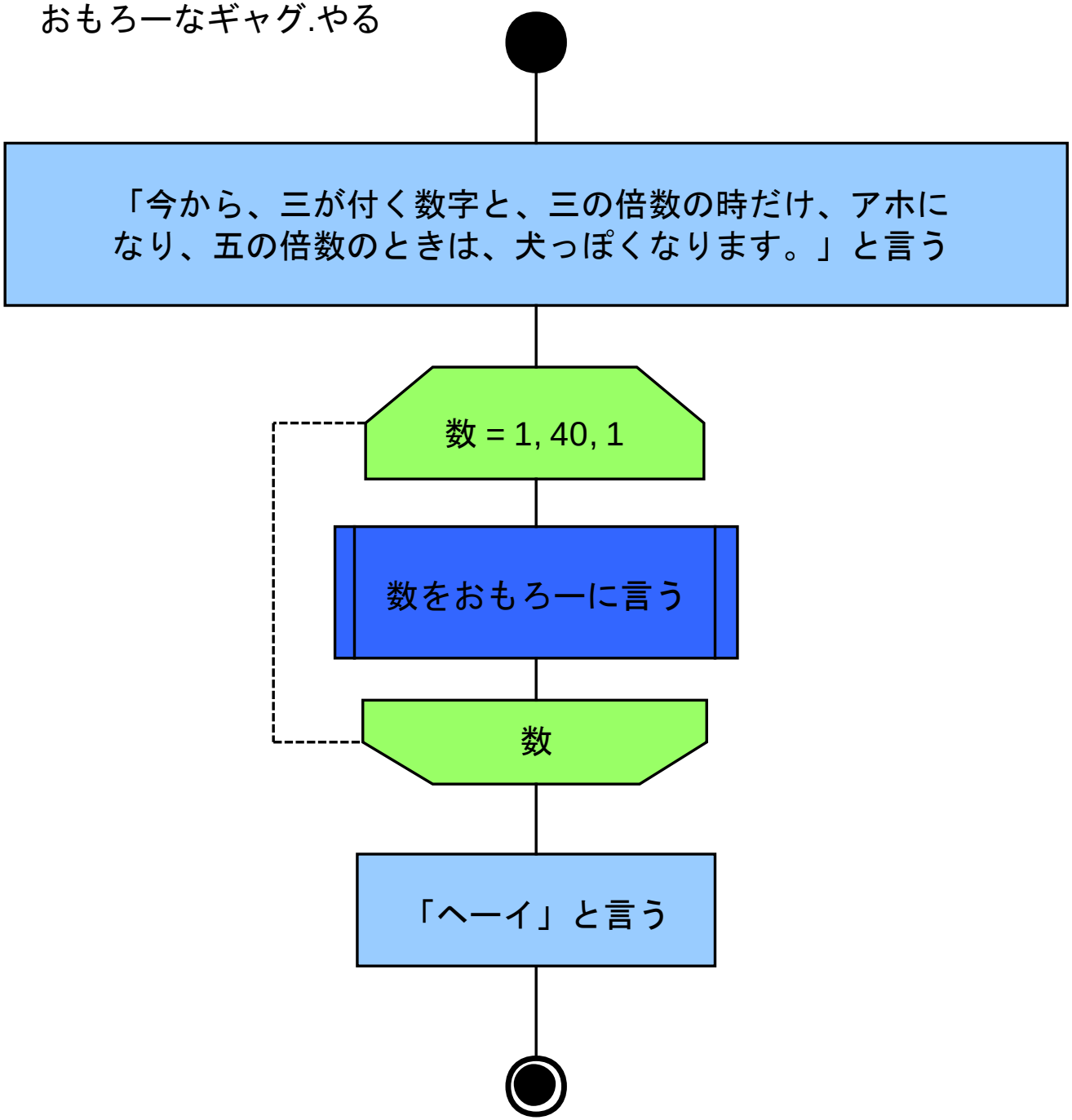
手続き脳で考えてみる

1. 順次実行、条件分岐、繰り返し処理の組み合わせで考える
2. 処理の順序を考える
3. 各繰り返しの中で何をするか考える
4. 更により小さい粒度の処理で、同様に考える

手続き脳で考えてみる

1. 1から40まで繰り返す
2. 繰り返しの中で、各の数についておもろーに言う
   - 数をおもろーに言う
     - 三が付くかまたは三の倍数で五の倍数のときはあほっぽく犬っぽく言う
     - 三が付くかまたは三の倍数のときは、あほっぽく言う
     - 五の倍数のときは、犬っぽく言う
     - それ以外のときは、普通に言う

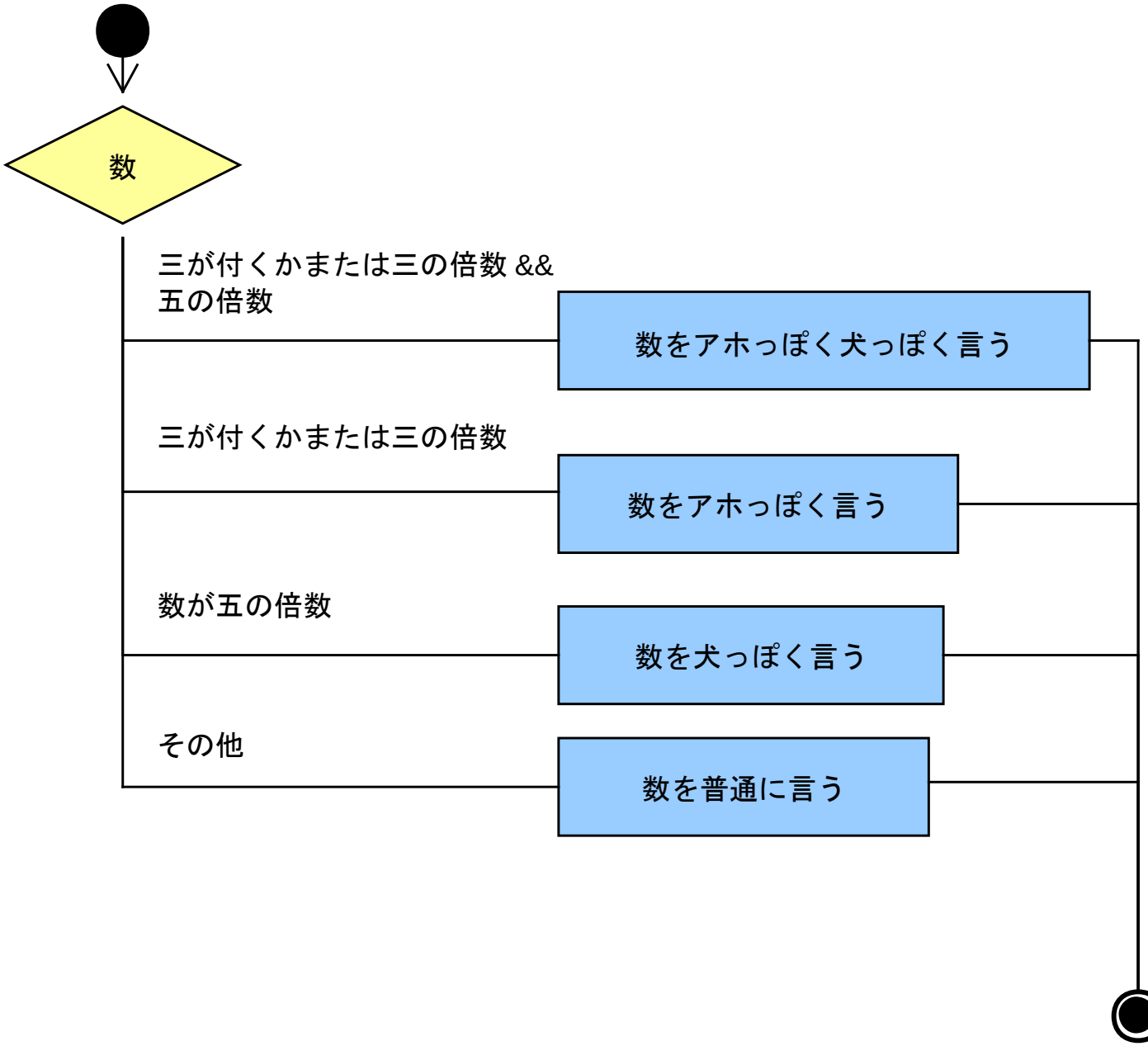
おもろーなギャグ.やる



手続き型
なべあつ
フローチャート1

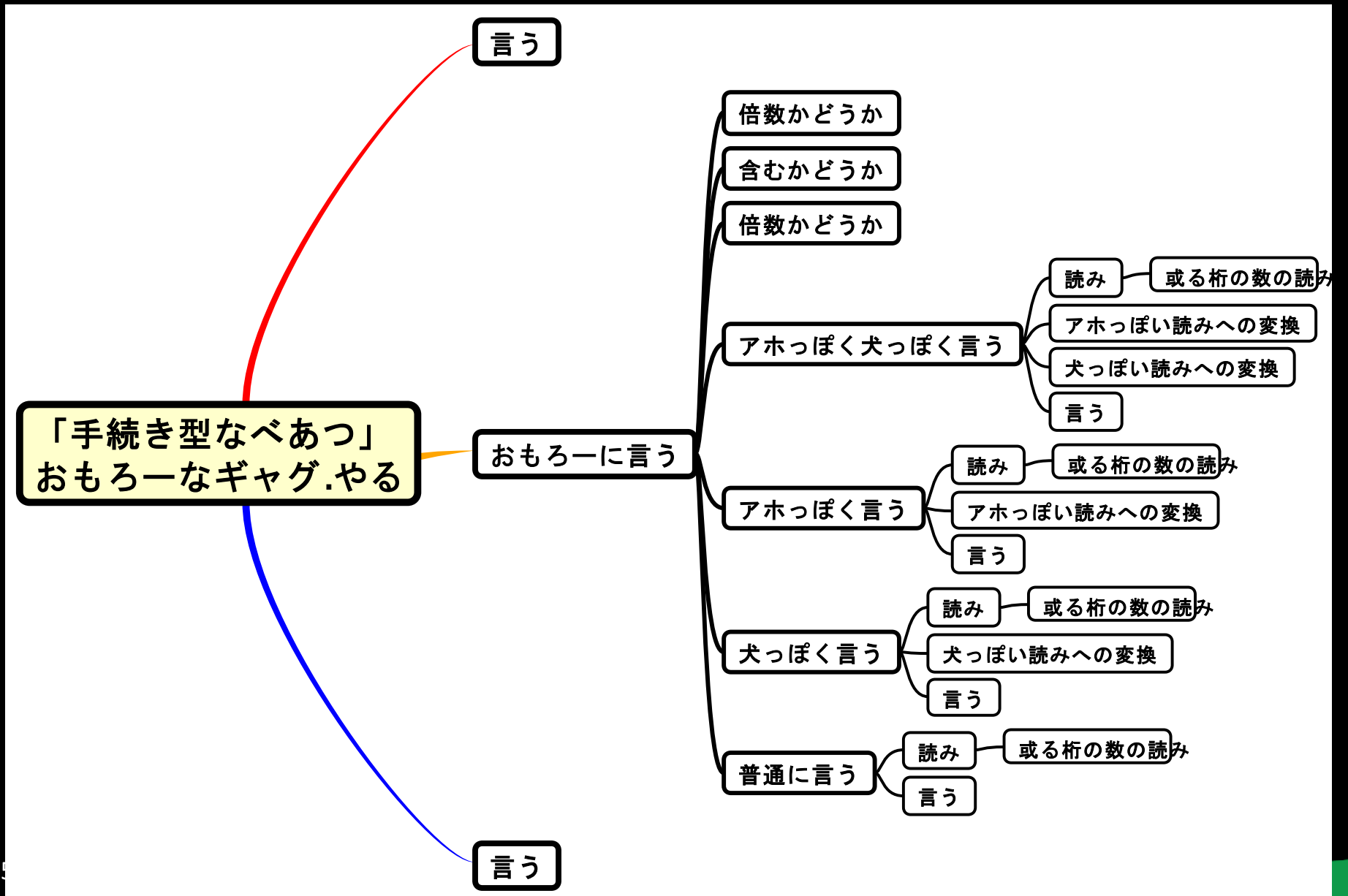
手続き型なべあつフローチャート2

数をおもろーに言う



Demo

手続き型なべあつの分割の様子



マルチパラダイム
なべあつ

マルチパラダイム

- 手続き型
- オブジェクト指向
- (アスペクト指向)
- ジェネリック
- 関数型
- 図解型

手続き型とは別の考え方

1. 「1～40の数の集合」を
2. 「集合に対する『おもろーな加工』をするフィルタ」に通したものが
3. 「集合を出力するもの」のデータだ

new おもろー()

シーケンシャルな.次の範囲の数(1, 40)

列挙可能 *(IEnurarable)* な何か

(列挙可能な何かを加工する汎用的な)
フィルタ!

遅延評価!

そのそれぞれに次の変換をし
(数 => おもろーに読む(数))

列挙可能 *(IEnurarable)* な何か

データバインド!

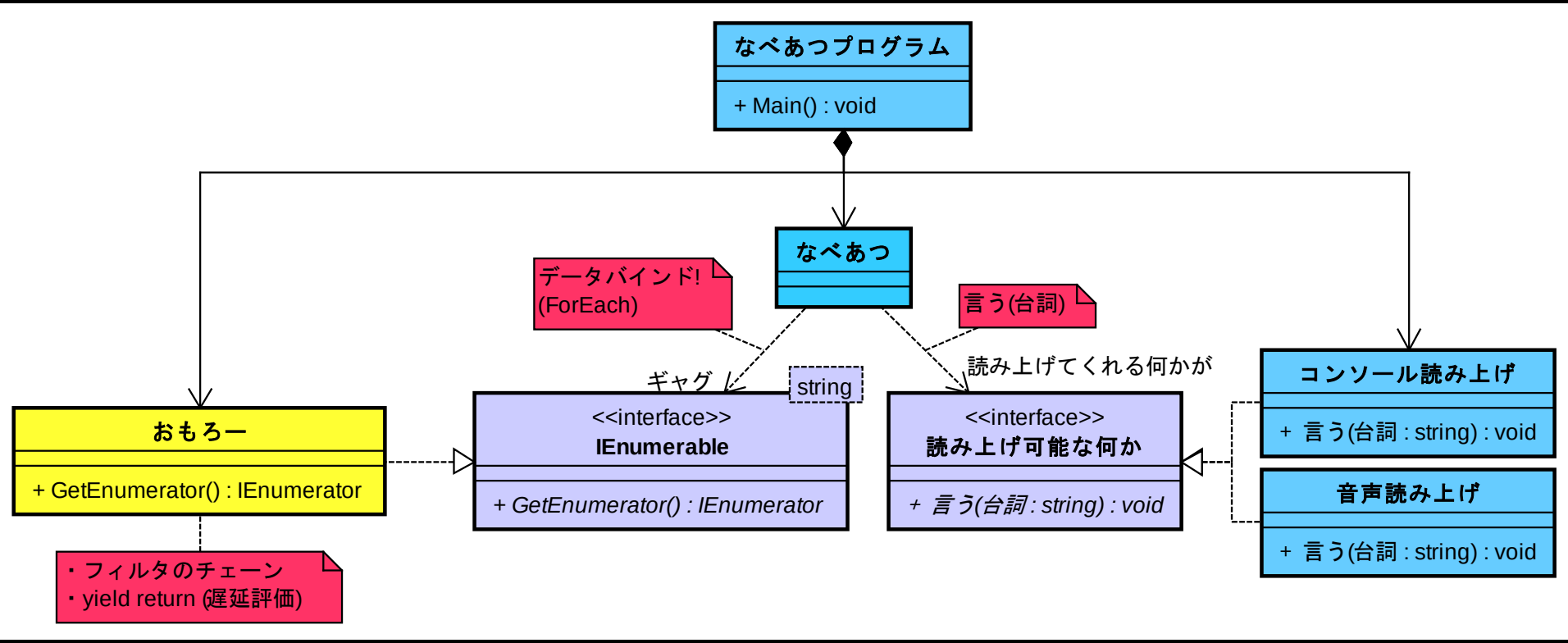
ギャグ

new なべあつ()

読み上げてくれる何か

列挙可能な何かを
出力できる何か

ニューなべあつ


なべあつプログラム
+ Main() : void
なべあつ
データバインド!
(ForEach)
言う(台詞)
ギャグ
string
読み上げてくれる何かが
おもろー
+ GetEnumerator() : IEnumerator
<<interface>>
IEnumerable
+ GetEnumerator() : IEnumerator
<<interface>>
読み上げ可能な何か
+ 言う(台詞 : string) : void
コンソール読み上げ
+ 言う(台詞 : string) : void
音声読み上げ
+ 言う(台詞 : string) : void
・フィルタのチェーン
・yield return (遅延評価)

関心の分離

- 分離される関心が違う
  - 手続き
  - オブジェクト
  - 汎用アルゴリズム
  - 汎用データ構造
- 手続きだけ、オブジェクトだけ、では分散する関心が多い

Demo

LINQで書き換え

new おもろー()

Enumerable.Range(1, 40)

IEnurarable

(列挙可能な何かを加工する汎用的な)
フィルタ!

Select(数 => おもろーに読む(数))

遅延評価!

IEnurarable

ギャグ

データバインド!

new なべあつ()

読み上げてくれる何か

列挙可能な何かを出力できる何か

Demo

参考: Linq to Object

```
var books = from    aBook in bookList
            where   aBook.ISBNコード == "BBBBBBB"
            orderby aBook.タイトル
            select  new { タイトル = aBook.タイトル, 価格 = aBook.価格 };
books.ToList().ForEach(item => Console.WriteLine(item));
```

```
bookList.Where  (aBook => aBook.ISBNコード == "BBBBBBB")
        .OrderBy(aBook => aBook.タイトル)
        .Select (aBook => new { タイトル = aBook.タイトル, 価格 = aBook.価格 })
        .ToList().ForEach(item => Console.WriteLine(item));
```

BookList

列挙可能 *(IEnurarable)* な何か

Where (aBook => aBook.ISBNコード == "BBBBBBB")

列挙可能 *(IEnurarable)* な何か

OrderBy(aBook => aBook.タイトル)

列挙可能 *(IEnurarable)* な何か

Select (aBook
new { タイトル = aBook.タイトル, 価格 = aBook.価格 })

列挙可能 *(IEnurarable)* な何か

ForEach(item => Console.WriteLine(item))

(列挙可能な何かを加工する汎用的な)
フィルタ群!

遅延評価!

列挙可能な何かを
何かする何か

3. 美しいプログラムについて

「美しいプログラム」とは？

「美しくないプログラム」とは？

Martin Fowler

Bad Smell『不吉な匂い』

- 「リファクタリング」を必要とするコード

- 重複したコード
- 長すぎるメソッド
- 巨大なクラス
- 多すぎる引数
- 変更の発散
- 変更の分散
- 属性、操作の横恋慕
- データの群れ
- 基本データ型への執着
- スイッチ文
- パラレル継承
- 怠け者クラス
- 疑わしき一般化
- 一時的属性
- メッセージの連鎖
- 仲介人
- 不適切な関係
- クラスのインタフェース不一致
- 未熟なクラスライブラリ
- データクラス
- 相続拒否
- コメント

きれいなコードとは何か?

- *Robert C. Martin*
  - 「いいコードとは、読みやすいコードだ」
- *Ward Cunningham*
  - 「いいコードとは、期待したコード (what you expect) だ」

Robert C. Martin

- 「きれいなコードを書こう。
  それがプロフェッショナルだ」

美しいプログラム

- *Ease to Change*
(変更容易性)
- *Ease to Test*
(検証容易性)

「美しいソースコードのための七箇条」

「美しいソースコードのための七箇条」

1. 意図を表現
2. 単一責務
3. 的確な名前
4. Once And Only Once
5. 的確に記述されたメソッド
6. ルールの統一
7. Testable

意図を表現

「プログラムの任意の部分で」

- 意図 (関心事) 以外のことが書かれていないこと
- 意図 (関心事) が書き尽くされていること
- ＝高凝集 (high cohesion)

分割が鍵

分割攻略 (Divide-and-Conquer) について復習

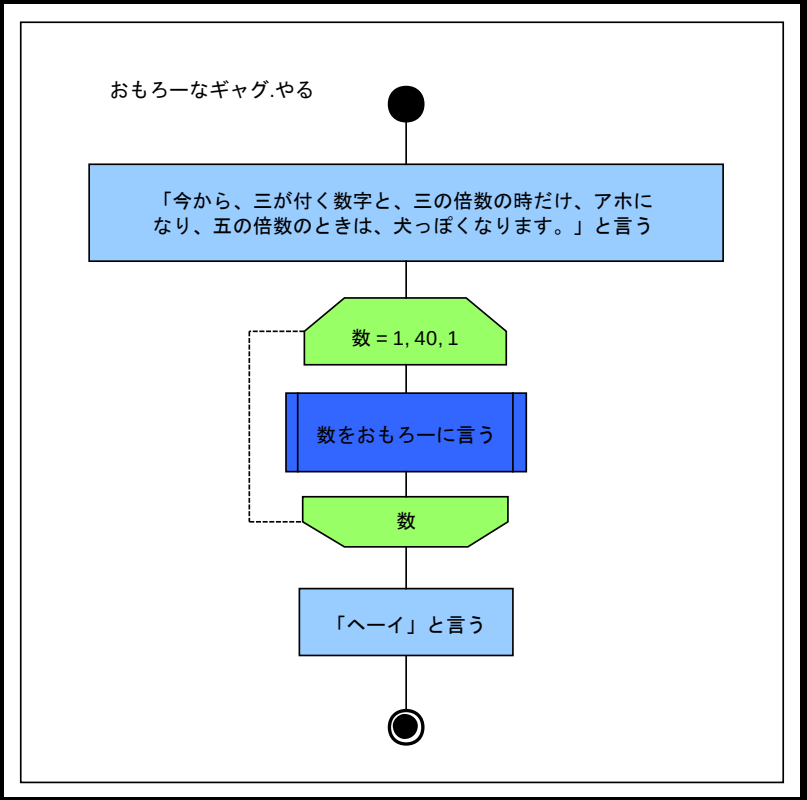
1. プログラム開発は複雑さとの戦い
2. 問題はサイズが小さいほうが簡単に解ける
3. もし大きなサイズの問題を、いくつかのより小さなサイズの問題に分割でき、それぞれを解けば良い状態にできれば、その方が容易に解ける
4. 小さな問題に分けてそれぞれに解を与える

分割攻略 (Divide-and-Conquer) について復習

- 「分かる」とは「分けられる」ということ
  - 「これとこれは違う問題と言えるようになる」こと
  - 「これであるものとないものの境界を知る」こと
- どう分けるか? が重要
  - 低結合 (low coupling) & 高凝集 (high cohesion)

どう関心を分離したいか

- ひとつのパラダイムだけでは、様々な分離のケースに対応が困難
  - 手続き型パラダイムは処理単位での分割



新たなパラダイムに関する態度

新たな概念の習得

→自分の既知の概念と結び付ける

これを安易にやってしまうと、思考停止を招くので要注意

「分かったつもり」

新たなパラダイムに関する態度

- 「それって結局×××のことだよね」
- 「それって昔からやってきたことで、別に新しくないじゃん」
- 「そんなの×××でもできるじゃん」
- 「結局現場じゃ使えないし使わない」
- 例. プログラミング言語Cの説明を受けて
  - 「そんなのアセンブリ言語でも『全部』できるじゃん」
  - 「アセンブリ言語でできることでできないことがあるから、現場じゃ使えないよ」

→ 要パラダイム シフト!

美しいプログラム

例.

- 「何でも手続きで書くのが美しいのか?」
- 「オブジェクト指向ですべてうまくいく?」
- 「何でもC#で書くのが美しい? 」
- 「なんでもかんでも XML で」
- 「モデルは全て UML で」

→ 適材適所

- 手続きは手続き
- ワークフローはワークフローデザイナで
- UIはUIデザイナで
- データはデータのデザイナで

ひとつのパラダイムに捕らわれない

「無執無着」

『一枚の葉にとらわれては木は見えん。一本の木にとらわれては 森は見えん。どこにも心を留めず 見るともなく全体を見る。それがどうやら「見る」ということのようだ。』

- バガボンド (井上雄彦/講談社) 第4巻より

新たなパラダイムに関する態度

例. LINQ

「どう使おうか? 別に要らないよね?」

ではなく

「書きたかったように
やっとできるようになった」

手続き的 or 宣言的

```
// 手続き的
for (int i = 0; i < 10; i++)
    何かする();
// 宣言的
10.回(何かする);
```

C# の記述:

手続き的

```
var textBlock = new TextBlock();
textBlock.FontSize = 18;
textBlock.Text = "Hello";
textBlock.SetValue(Canvas.LeftProperty , 150);
textBlock.SetValue(Canvas.TopProperty ,  50);
```

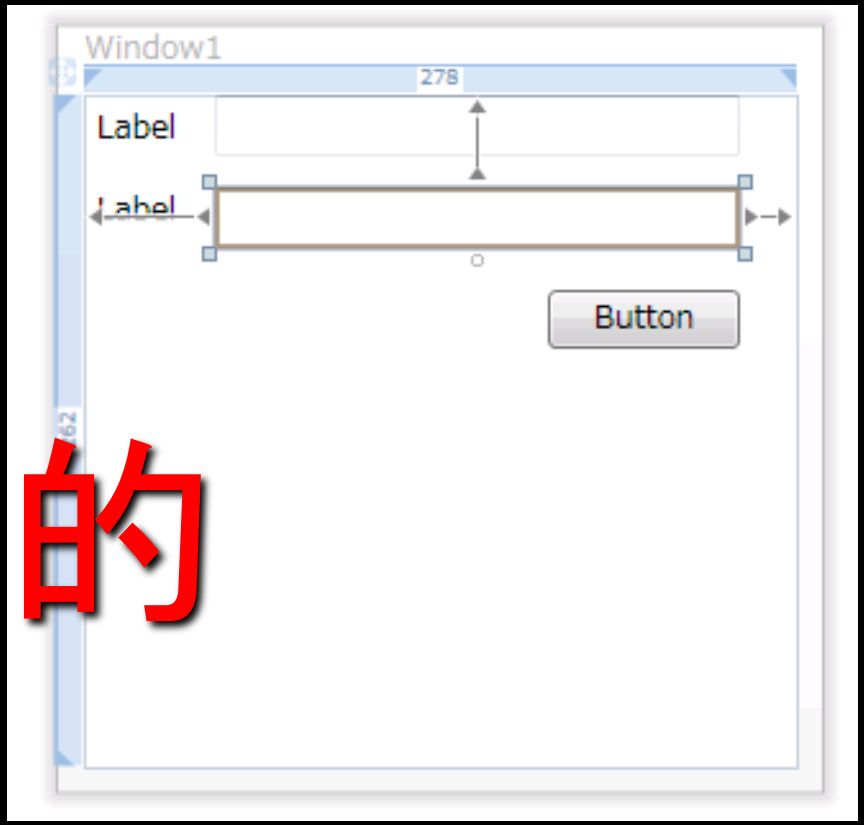
XAMLの記述:

宣言的

```
<TextBlock FontSize="13"
  Text="Hello" Canvas.Top=“50"
  Canvas.Left=“150"/>
```

VS2008
デザイナ
による記述:

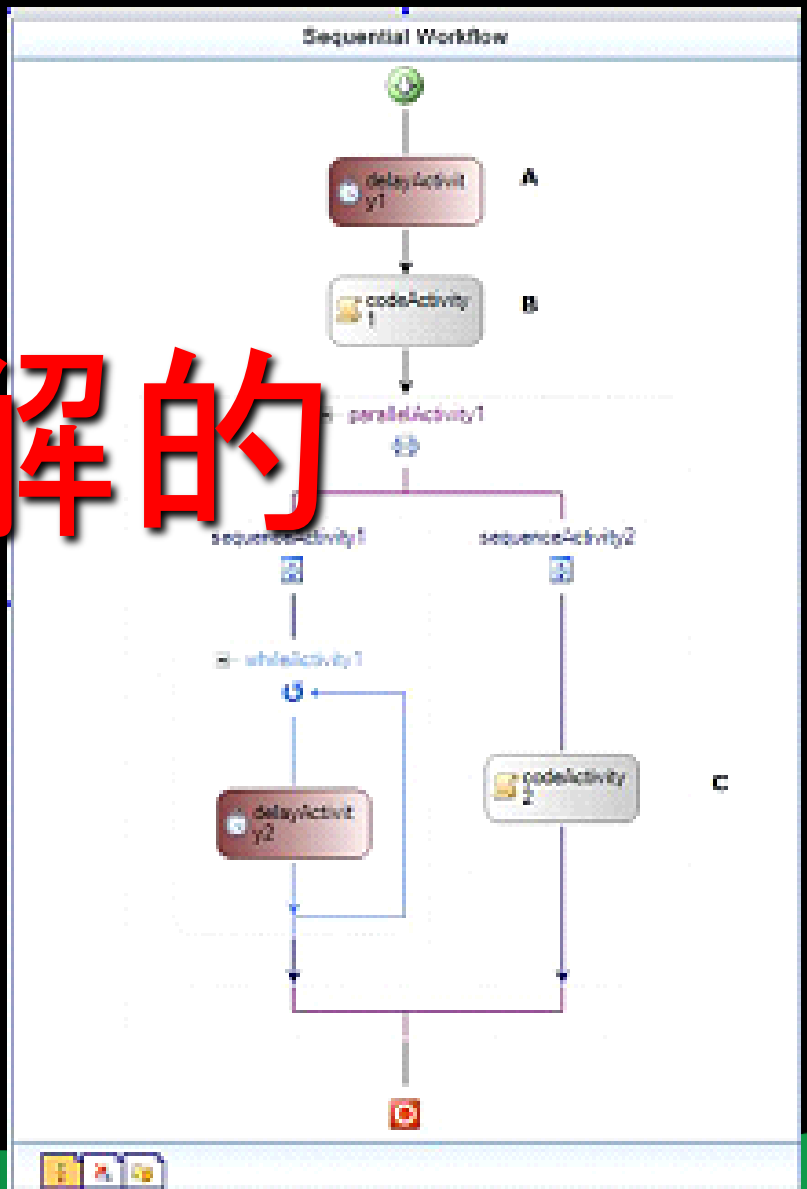
図解的



C# (手続き的記述) や XAML (宣言的記述) と
比較して意図以外のノイズが少ない

VS2008 と WF

図解的



VS2008 で EDM (Entity Data Model)

他の言語に学ぼう

- Haskell
  - 純粋な関数型プログラミング言語
  - 副作用の除去
  - ラムダ式
  - 遅延評価
- F#
  - 関数型プログラミングを強化した .NET 向けマルチパラダイム言語
- 各種DSL

F# 版:

```
let (|Mul|_|) m n =
  if n % m = 0 then Some n else None;;
let (|Col|_|) m n =
  let f =
    Seq.exists ((=) m)
    << Seq.unfold (fun n -> if n > 0 then Some (n % 10, n / 10) else None)
  in
  if f n then Some n else None;;
let nabeatsu =
  let ahoppokusuru s = "^" + s + "!" in
  let (|Aho|_|) =
    function Mul 3 n | Col 3 n -> Some n
           | _ -> None
  in
  let inuppokusuru s = s + "わう〜ん" in
  let (|Inu|_|) =
    function Mul 5 n -> Some n
           | _ -> None
  in
  function Aho _ & Inu n -> ahoppokusuru << inuppokusuru << int_to_string <| n
         | Aho n        -> ahoppokusuru << int_to_string <| n
         | Inu n        -> inuppokusuru << int_to_string <| n
         | n -> int_to_string n;;

let _ =
  seq { 1..40 }
  |> Seq.map nabeatsu
  |> Seq.iter (printf "%s ");
  System.Console.WriteLine();
  System.Console.WriteLine("へーイ!\n");;
```

Microsoft

Microsoft® tech·ed Japan | 2008

Be a part of the experience.