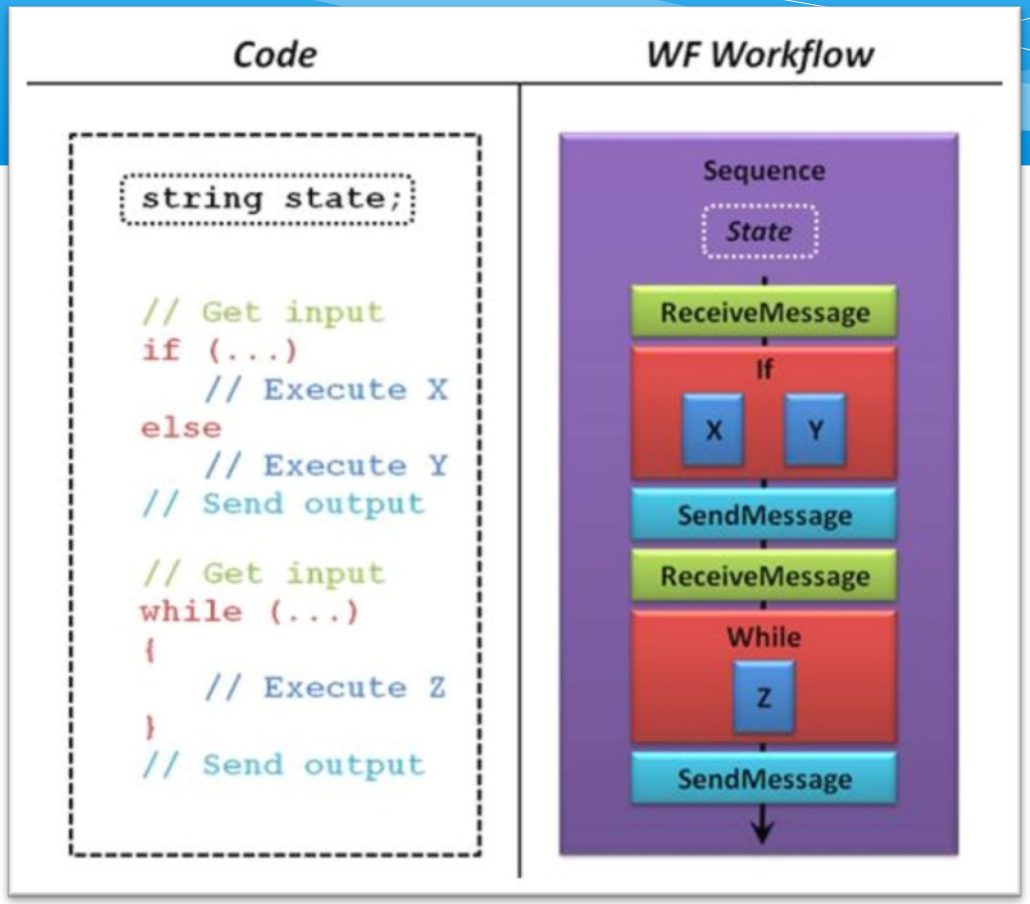
# Windows Workflow Foundation

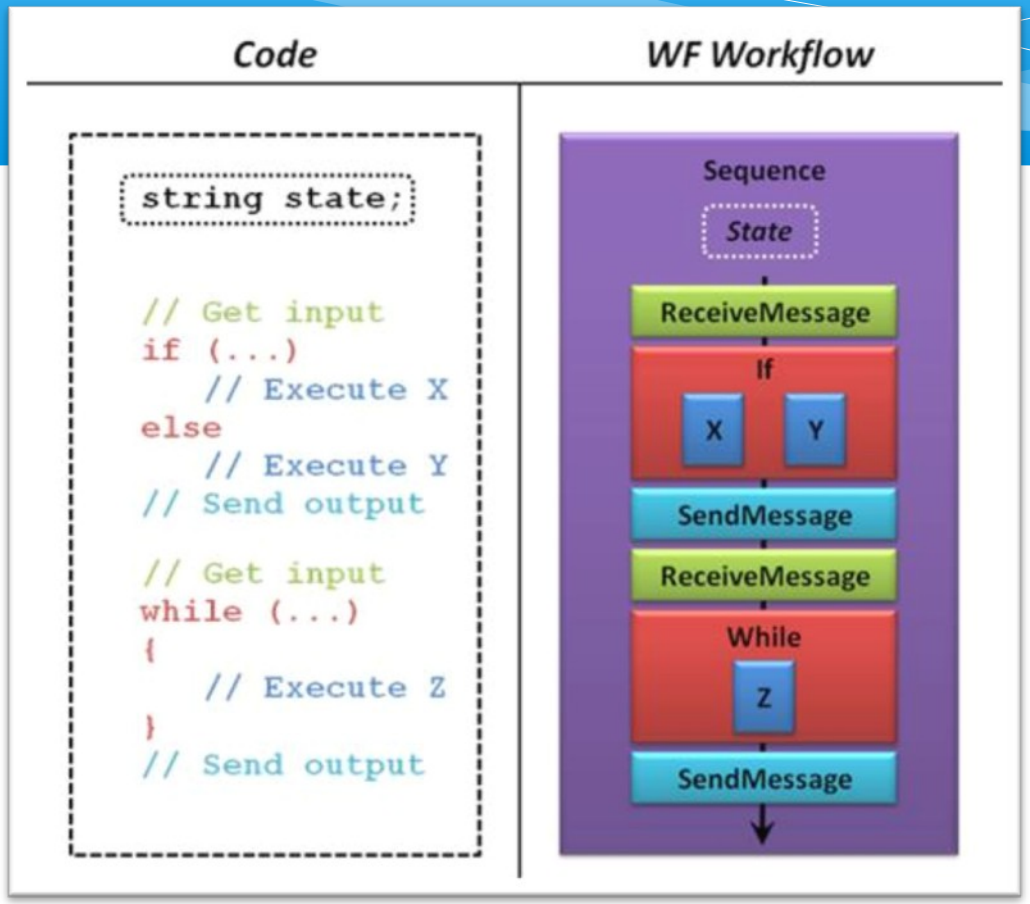
WF został wydany wraz z  NET Framework 3.0 w 2006 r., a następnie zaktualizowany w NET Framework 3.5. Te dwie pierwsze wersje były przydatne, zwłaszcza dla niezależnych dostawców oprogramowania (ISV), ale nie stały się głównym nurtem technologii dla programistów korporacyjnych. Według twórców WF, wraz z wyjściem .NET 4.0 ma się to zmienić. Głównym celem tej najnowszej wersji jest to, aby WF stało się standardowym elementem zestawu narzędzi programowania dla wszystkich użytkowników .NET. (informacje przetłumaczone ze strony MS)

Czym jest WF?

Aplikacje stworzone przy pomocy WF robią zazwyczaj to samo, co zwykłe aplikacje.  Wszystko jednak robi się tu przy użyciu aktywności. Poniższy obrazek przedstawia kod programu oraz to samo wykonane za pomocą WF.



Ponieważ w WF wszystko jest aktywnością, widoczna poniżej aktywność Sequence zawiera cały nasz program, i jak zwykły program, może mieć zmienne i ma jakiś stan.

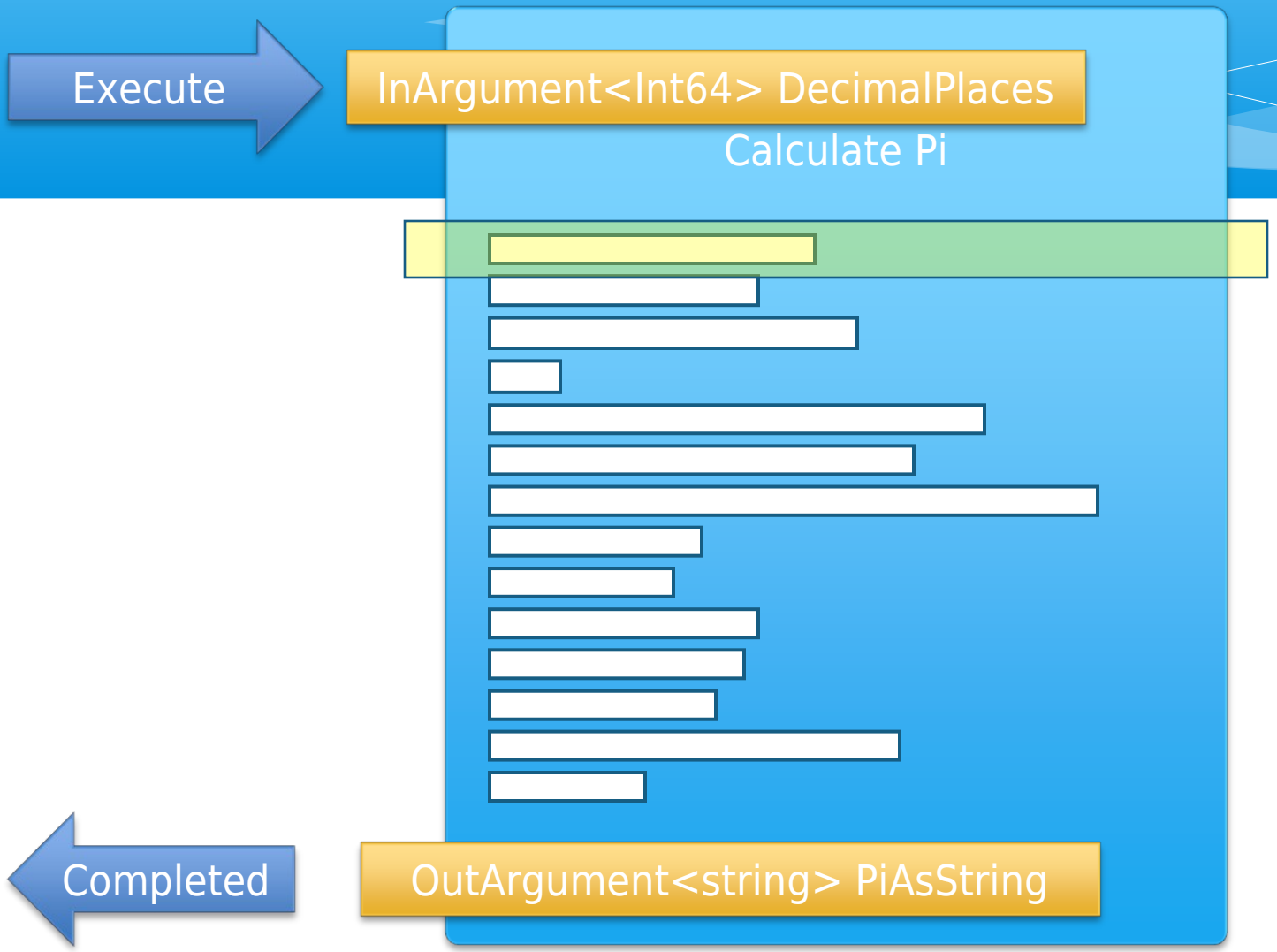


- WF jest aktywnością
  - może być napisany w postaci kodu lub zdefiniowany w XAML – lub dowolnym innym formacie (np w jęz. dynamicznym)
  - Aktywność może zawierać inne aktywności
  - Aktywność może mieć argumenty we i wy (jak również zmienne)
  - Aktywności zwracajace warość mają oddzielne klasy bazowe
    - n.p. CodeActivity<TResult>, NativeActivity<TResult>
- WF jest definicją, wg której można utworzyć wiele instancji
  - Każda instancja ma indywidualne środowisko (zmienne, argumenty)
    - InArgument

# Rodzaje aktywności

Najprostszy przykład – wywołujemy aktywność, która robi to co ma do zrobienia i zwraca jakiś rezultat.

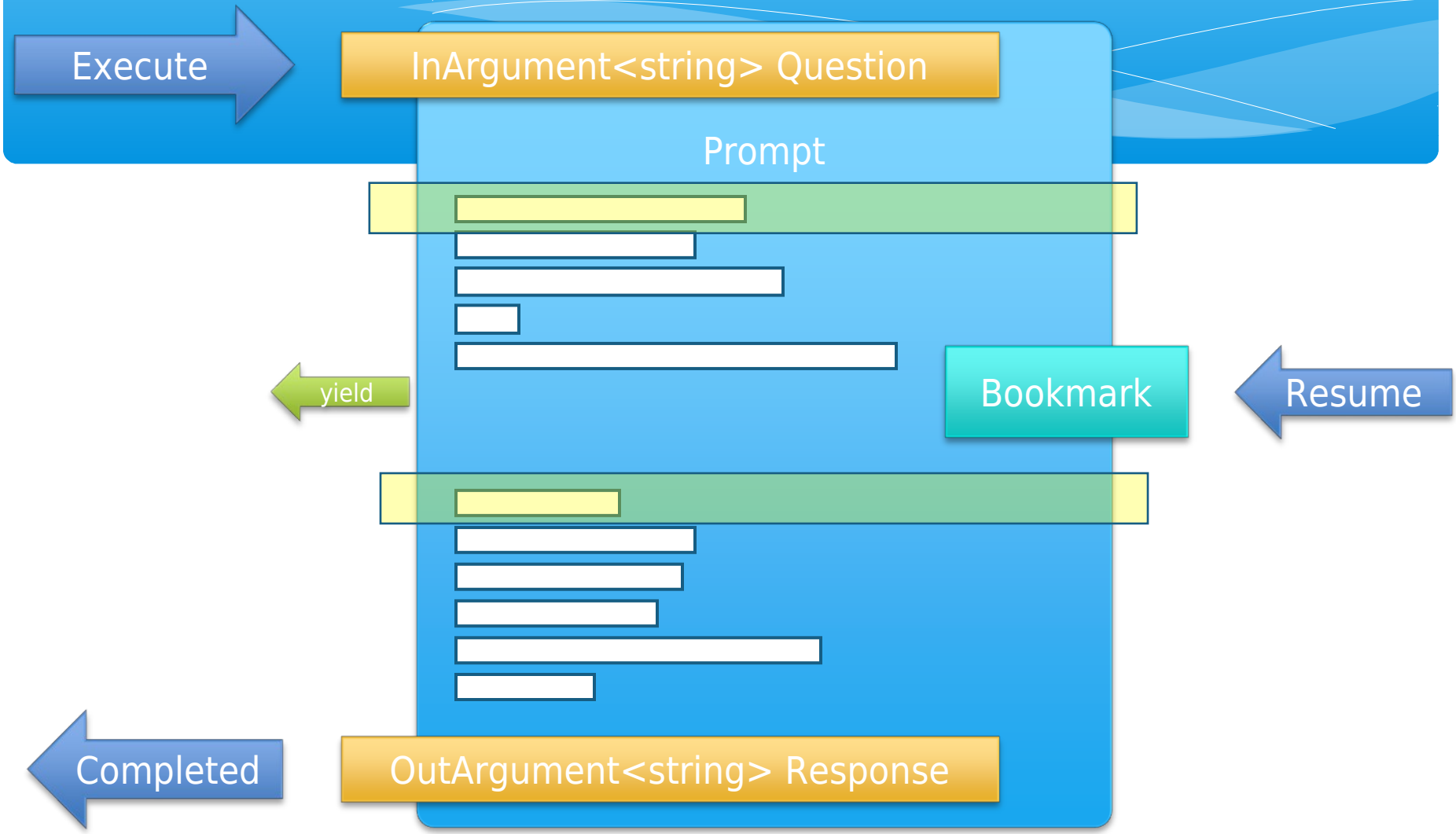
## Atomic execution



# Rodzaje aktywności

Drugi rodzaj – modeluje długoterminowe procesy (np. biznesowe)

Continuation, Long Running, or Reactive Execution



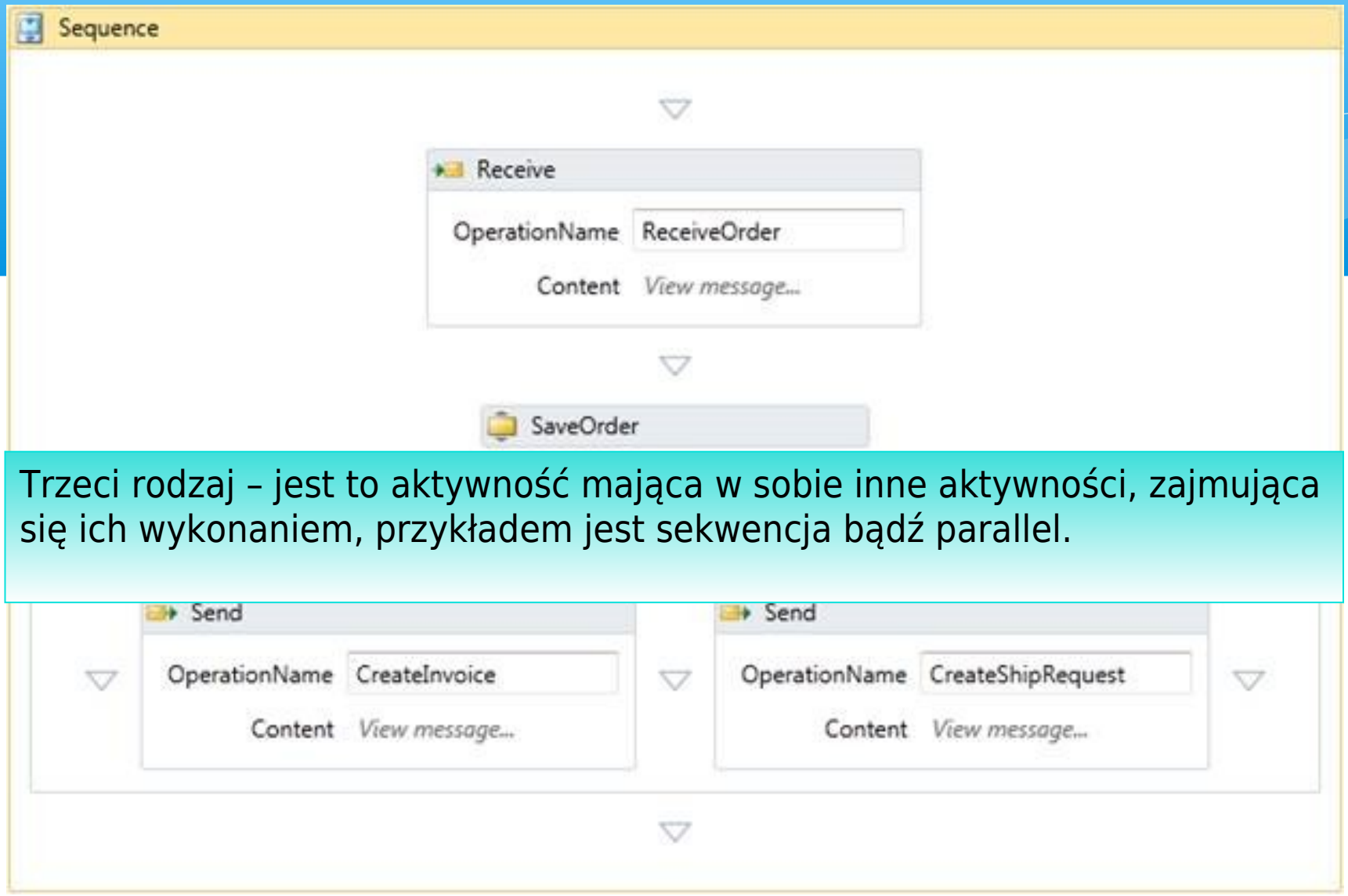
# Rodzaje aktywności

Continuation, Long Running, or Reactive Execution

Zakładka:
-Punkt wznowienia w WF
-Wznowienie wskazuje metodę zwrotną aktywności
-WF runtime zarządza wznowieniem – instancja nie musi być w pamięci

W przypadku, gdy aktywność oczekuje na informacje z zewnątrz, mamy do czynienia z persystencją. Persystencja pozwala wstrzymywać instancję WF, zapisać ją „gdzieś” i wznowić później.

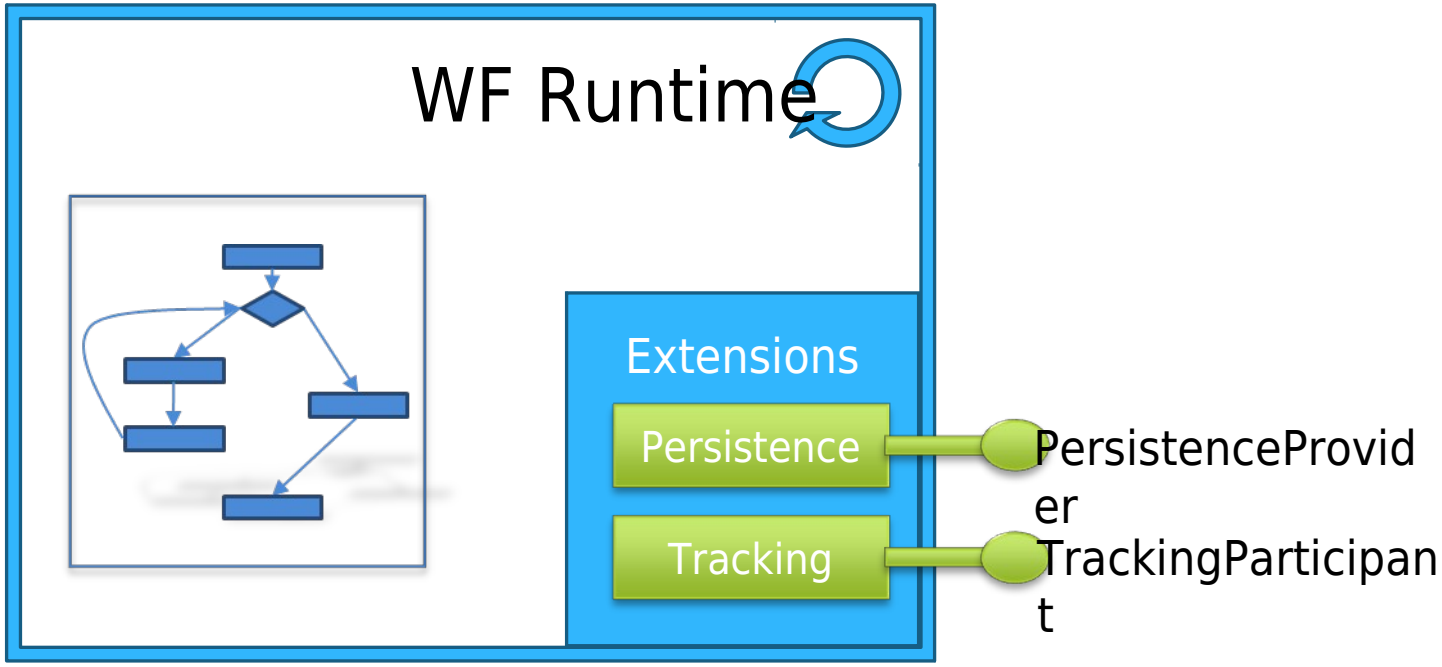
# Rodzaje aktywności



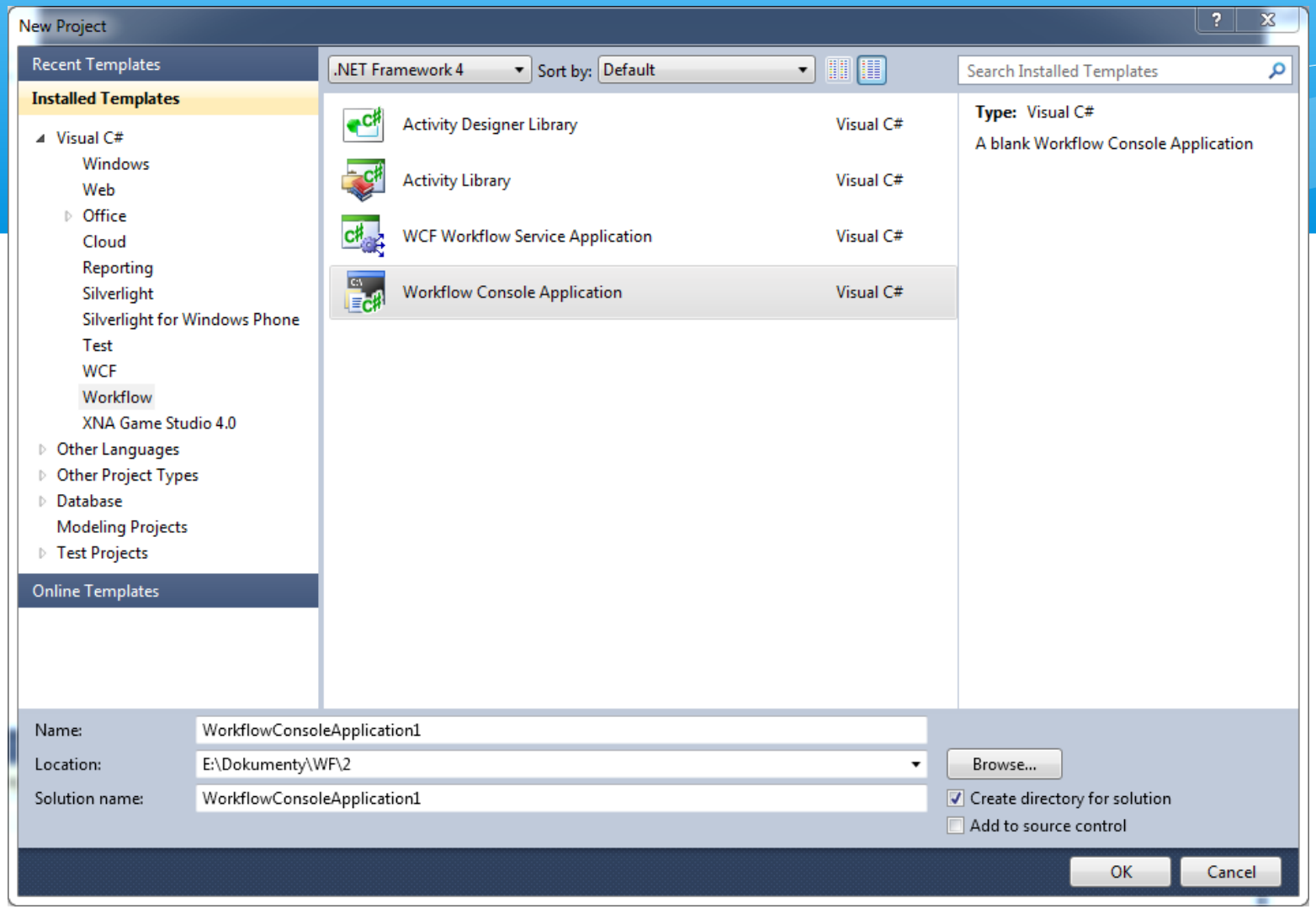

Trzeci rodzaj – jest to aktywność mająca w sobie inne aktywności, zajmująca się ich wykonaniem, przykładem jest sekwencja bądź parallel.

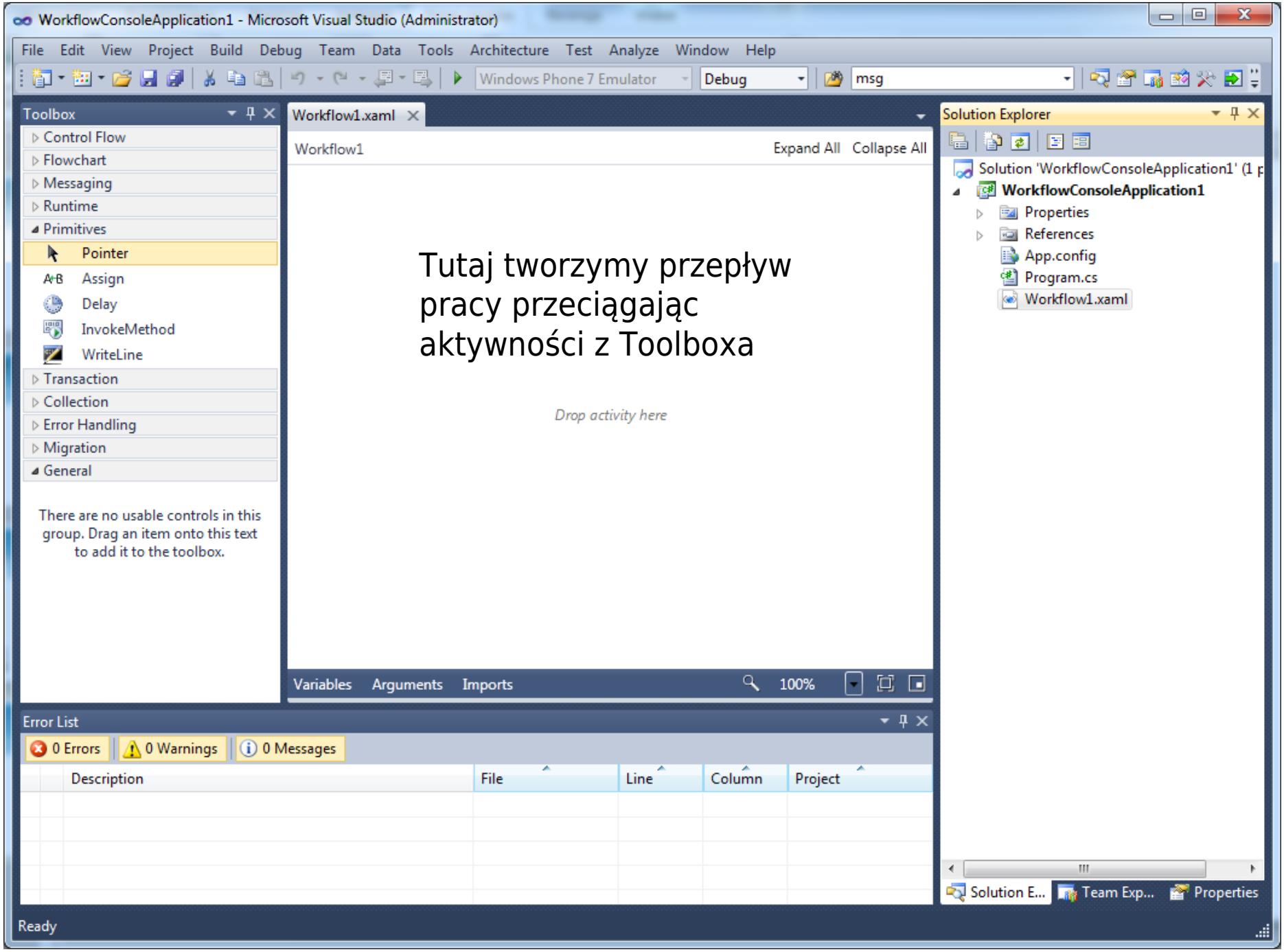
# WF Runtime

WF runtime „widzi” aktywności, aktywności, aktywności (nie Sequence, Parallel, Recurrence).
WF runtime jest „sędzią” pilnuje reguł.
Podczas wykonywania wielu procesów persystuje te, które się nudzą.
Tracking – Runtime podczas wykonywania aplikacji „loguje” pewne informacje, przez co wiemy, w którym miejscu jest nasz proces.

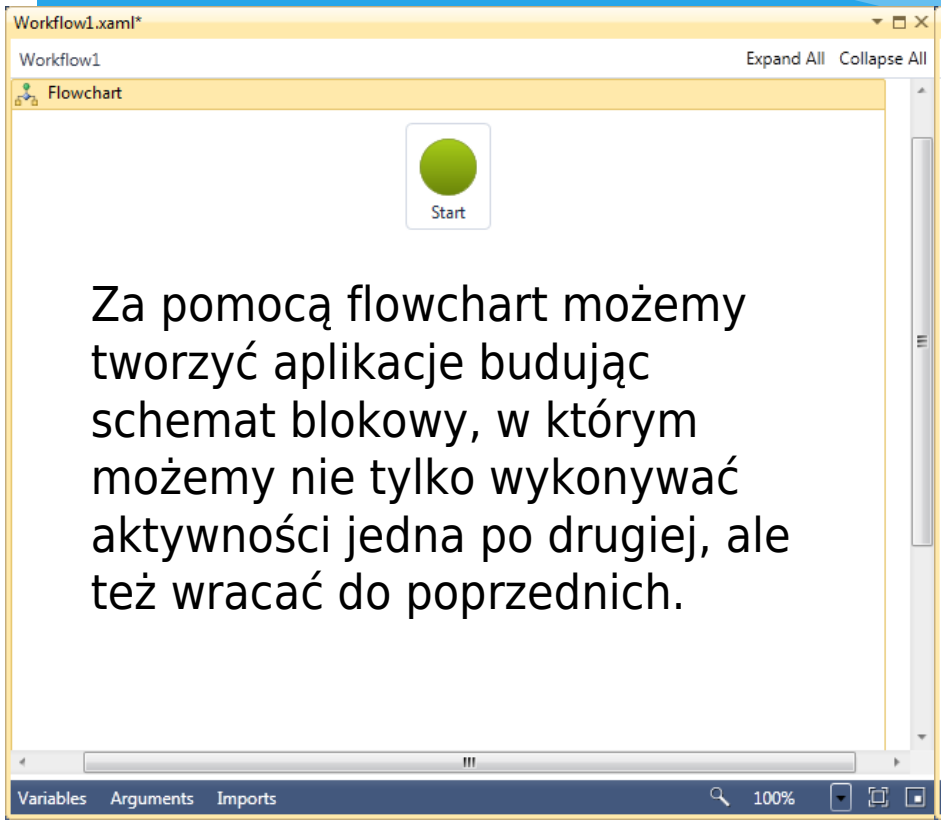


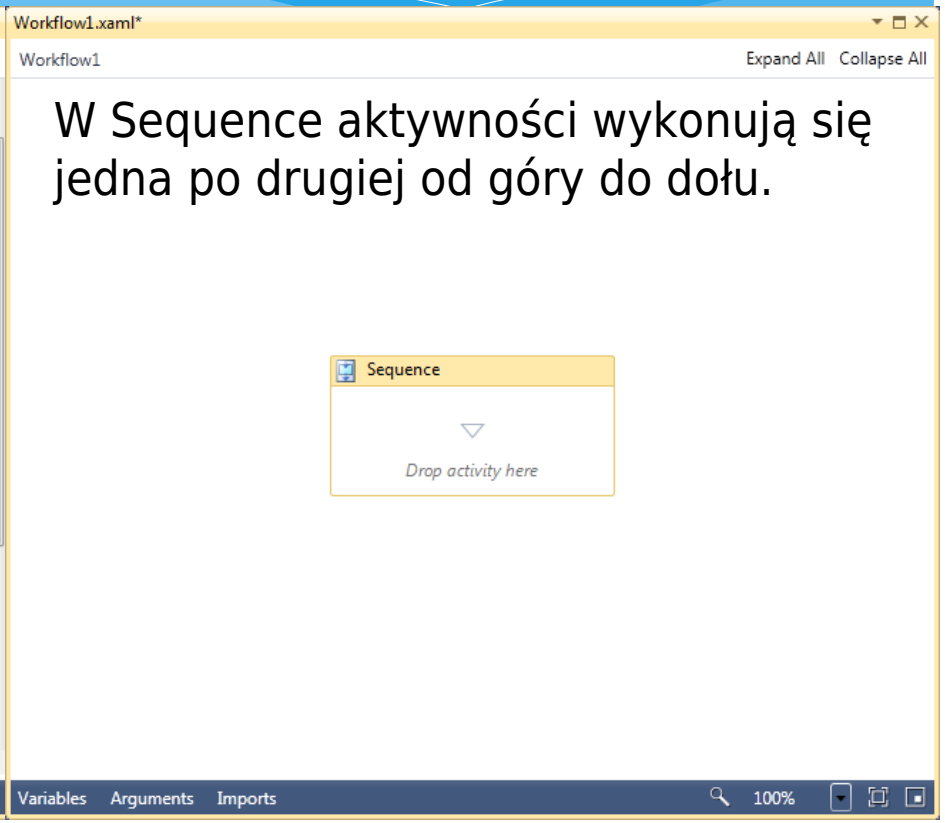
# Jak to wygląda w praktyce?




WorkflowConsoleApplication1 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Windows Phone 7 Emulator
Debug
msg
Toolbox
Control Flow
Flowchart
Messaging
Runtime
Primitives
Pointer
Assign
Delay
InvokeMethod
WriteLine
Transaction
Collection
Error Handling
Migration
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Workflow1.xaml
Workflow1
Expand All
Collapse All
Tutaj tworzymy przepływ pracy przeciągając aktywności z Toolboxa
Drop activity here
Variables
Arguments
Imports
100%
Solution Explorer
Solution 'WorkflowConsoleApplication1' (1 p
WorkflowConsoleApplication1
Properties
References
App.config
Program.cs
Workflow1.xaml
Solution E...
Team Exp...
Properties
Error List
0 Errors
0 Warnings
0 Messages
Description
File
Line
Column
Project
Ready

# W tym momencie mamy do wyboru dwa rodzaje przepływu pracy:

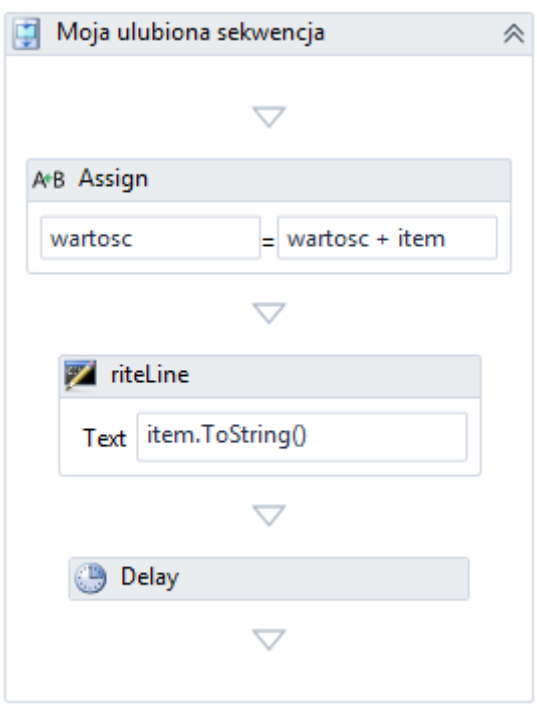



Przepływy pracy można dowolnie w sobie zagnieżdżać, kopiując np. Sequence z jednego przepływu do flowchart z innego.

# Zmienne i argumenty:

Niektórym aktywnościom można zmieniać nazwy. Umożliwia to łatwiejszą identyfikacje.




Do WF można dodawać zmienne i argumenty. Zmienne (Variables) należy przypisać do aktywności złożonej (sequence). Podczas tworzenia zmiennej wybiera się aktywność, do której ma być przypisana. Dzięki temu zmienna jest widoczna w aktywnościach podrzędnych.

W przeciwieństwie do zmiennych, argumenty (Arguments) są widoczne w całym przepływie. Przy tworzeniu wybiera się ich „kierunek” (direction: In, Out, Property). Dzięki temu można przypisać im wartość z zewnątrz przy wywoływaniu WF, bądź też zwrócić je na zewnątrz wraz z końcem pracy WF.

Oczywiście zarówno argumentom jak i zmiennym należy przy tworzeniu wybrać odpowiedni typ.

# Przykładowe aplikacje

W folderze Sequence i Flowchart załączona jest aplikacja wykorzystująca WF do liczenia wyniku wyrażenia 5!

W pliku Program.cs mamy następujący kod:

```
namespace WorkflowConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            WorkflowInvoker.Invoke(new Activity1());
        }
    }
}
```

WorkflowInvoker.Invoke(new NazwaPrzeplywu()); - najprostrzy sposób uruchomienia WF

WorkflowInvoker.Invoke Method

# WF można wywołać w aplikacji na kilka sposobów

W folderze WindowsFormApplication z WF załączona jest aplikacja, w której pokazanych jest kilka sposobów odpalenia WF w Windows Form Application, jak również jak odebrać output i sprawdzić stan WF.

# Tworzenie własnych aktywności

Własne aktywności można tworzyć na dwa sposoby.

Pierwszy z nich to dodanie do projektu nowej aktywności i zbudowanie jej za pomocą innych.

Drugi sposób to napisanie własnej aktywności (Code Activity) za pomocą języka obsługiwanego przez WF (np. C#, VB).

# Tworzenie własnych aktywności

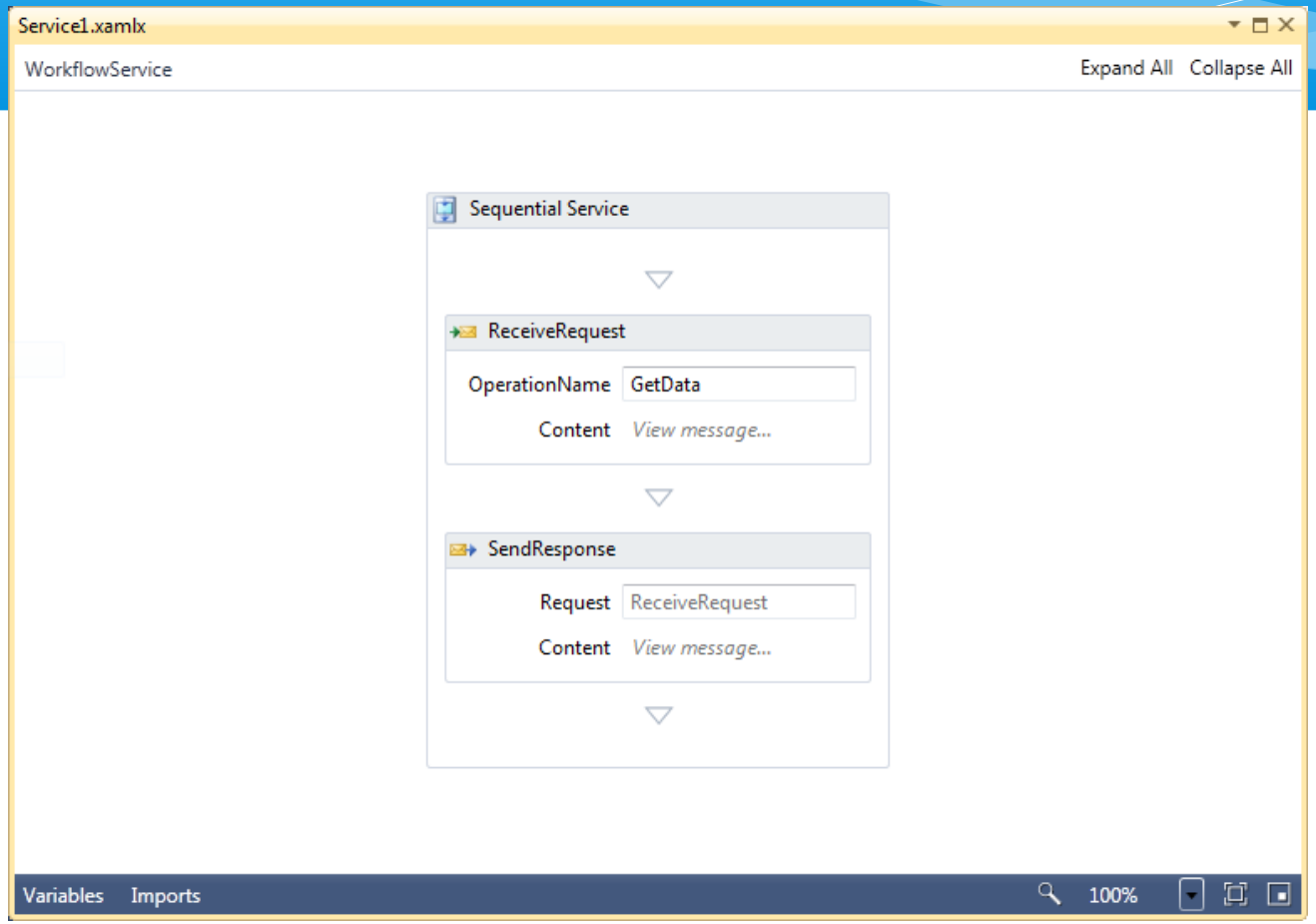
```
...
namespace ActivityLibrary1
{
    public sealed class CodeActivity1 : CodeActivity
    {
        // Define an activity input argument of type string
        public InArgument<string> Text { get; set; }

        // If your activity returns a value, derive from
CodeActivity<TResult>
        // and return the value from the Execute method.
        protected override void Execute(CodeActivityContext context)
        {
            // Obtain the runtime value of the Text input argument
            string text = context.GetValue(this.Text);
        }
    }
}
```

Tak wygląda „świerza” Code Activity.
InArgument to argument wejściowy, domyślnie jest definiowany jeden o nazwie Text.
Wartość Text można zmieniać po dodaniu naszej aktywności do przepływu (flowchart lub sequence), kliknięciu na nią i wpisaniu tego, co chcemy w odpowiednie pole w zakładce Properities.

# Tworzenie własnych aktywności

W folderze Własne Aktywności dołączona jest aplikacja konsolowa, w której utworzone są aktywności – jedna za pomocą Sequence, druga to Code Activity, trzecia podobna do Code Activity to NativeActivity – umożliwia zagnieżdżanie w sobie aktywności w kodzie.

# Kolejny typ aplikacji, którą można utworzyć to WCF Workflow Service Applicaton




Domyślnie usługa ta ma za zadanie przyjmować żądania i wysyłać odpowiedzi, dlatego też po utworzeniu projektu widzimy taki oto przepływ pracy.

# WCF Workflow Service Applicaton

W folderze WCF Workflow Service Applicaton znajduje się prosta aplikacja uruchamiająca usługę przyjmującą dwie liczby i zwracającą ich sumę oraz klienta umożliwiającego korzystanie z usługi.