IRA A. FULTON SCHOOLS OF engineering

school of computing, informatics, & decision systems engineering

CSE446 / CSE598
Software Integration and Engineering

Unit 2

Software Development by Composition and Integration

Lecture 2-2

Workflow-Based Software Development Concepts

Dr. Yinong Chen

Unit 2 Outline

- 2-1 Enterprise Architecture and Business Process
- 2-2 **Workflow Foundation 1: Concepts**
- 2-3 Workflow Foundation 2: Case Study
- 2-4 BPEL (Business Process Execution Language)
  - WSDL in BPEL
  - BPEL constructs and BPEL Process Definition
- 2-5 A Case Study of BPEL Application
- Stateful Services
- 2-6 Development Frameworks Supporting BPEL
  - Oracle SOA Suite and BizTalk
- 2-7 Message-Based Integration
- 2-8 Web Caching and Recommendation

Unit Test 2

Lecture 2-2 Outline

- Previous lecture focused on the architecture design and management
- Workflow for Architecture and Process Integration
  - Cover the Gap between Architecture (CSE360) and Development (Programming Courses)
  - Key Ideas of Workflow-based Application Development
  - Workflow Foundation Constructs and Activities
    - Creating WF Workflow Application
    - Creating WF Workflow Service
    - Event-Driven Approach and State Machine in WF

Project 2
Assignment 3

What is Workflow?

- Workflow is a new solution to an old problem: Integrating, managing, and supporting business process.
- Workflow offers a new model for the division of labor between people and computer:
  - People do only those that computers cannot: What we want.
- Workflow better matches the business logic that customers require, and thus the customers can better understand the solution offered by the software engineers / architects.
- Workflow better separates the tasks of software architects and programmers.
- Service-Oriented Architecture makes workflow easier.

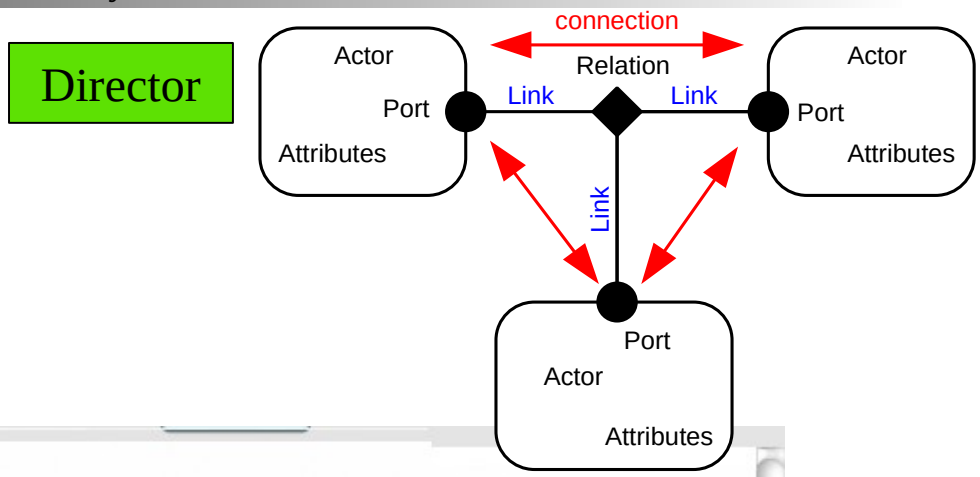
Kepler Project

https://kepler-project.org/

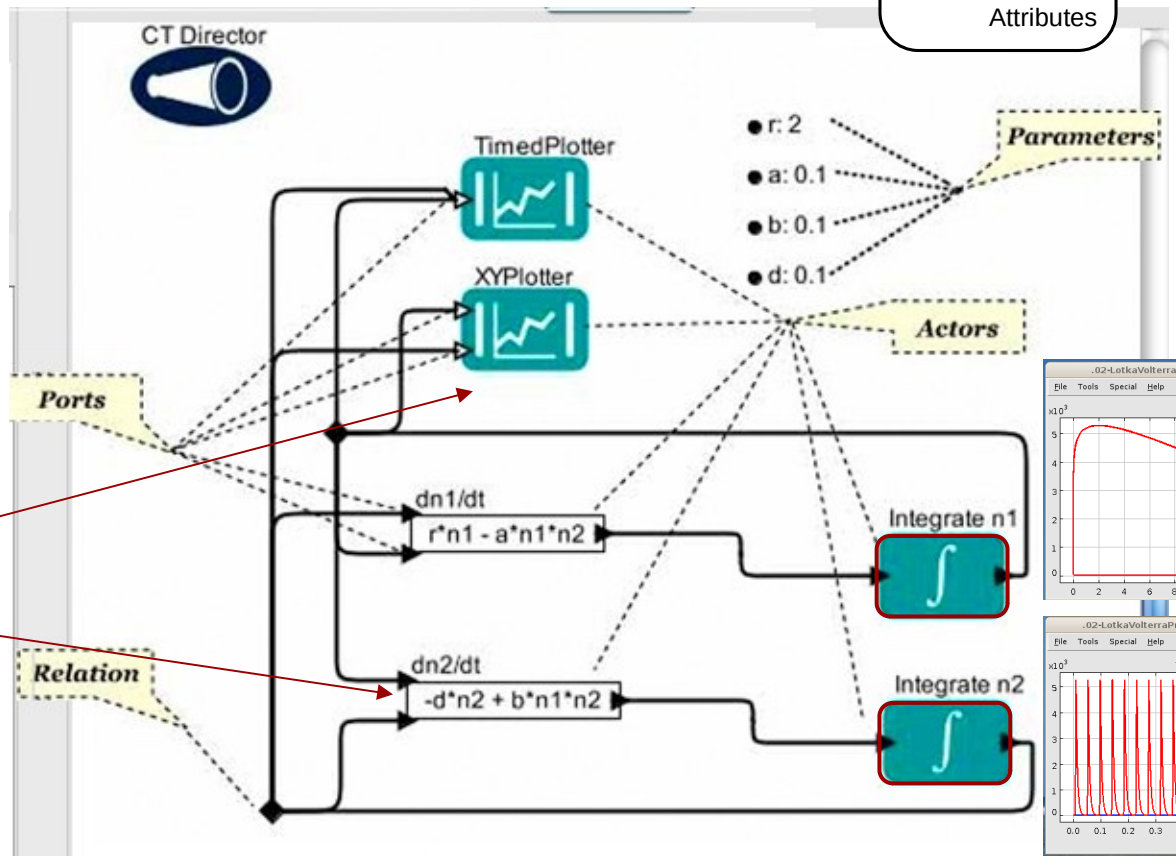
- Operate on data stored in a variety of formats, locally and over the internet;
- Integrating disparate software components, such as merging scripts with compiled code;
- Facilitating remote, distributed execution of models.
- Select and then connect relavent analytical components and data sources to create a "scientific workflow“ — an executable representation of the steps required for generating results;
- Share and reuse data, workflows, and components developed by the community to address common needs.

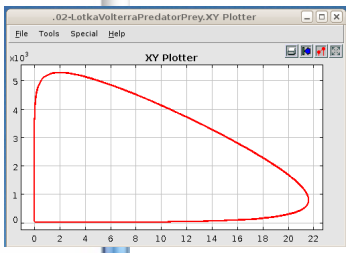
Kepler Workflow System

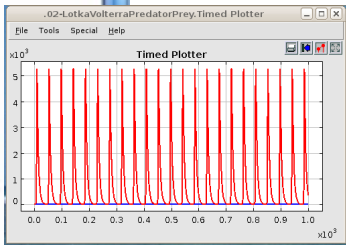
http://edutechwiki.unige.ch/en/Kepler_workflow_system



- Kepler Workflow Style
  - Directors (define computing and communication models)
  - Actors
    - Ports
    - Attributes
  - Relations
- Example: Integration of a template, a data set







Traditional Software Development

- Architecture Design
- Flowchart / Process

Software Architects
Software Engineers

- Coding / Debugging

Programmers

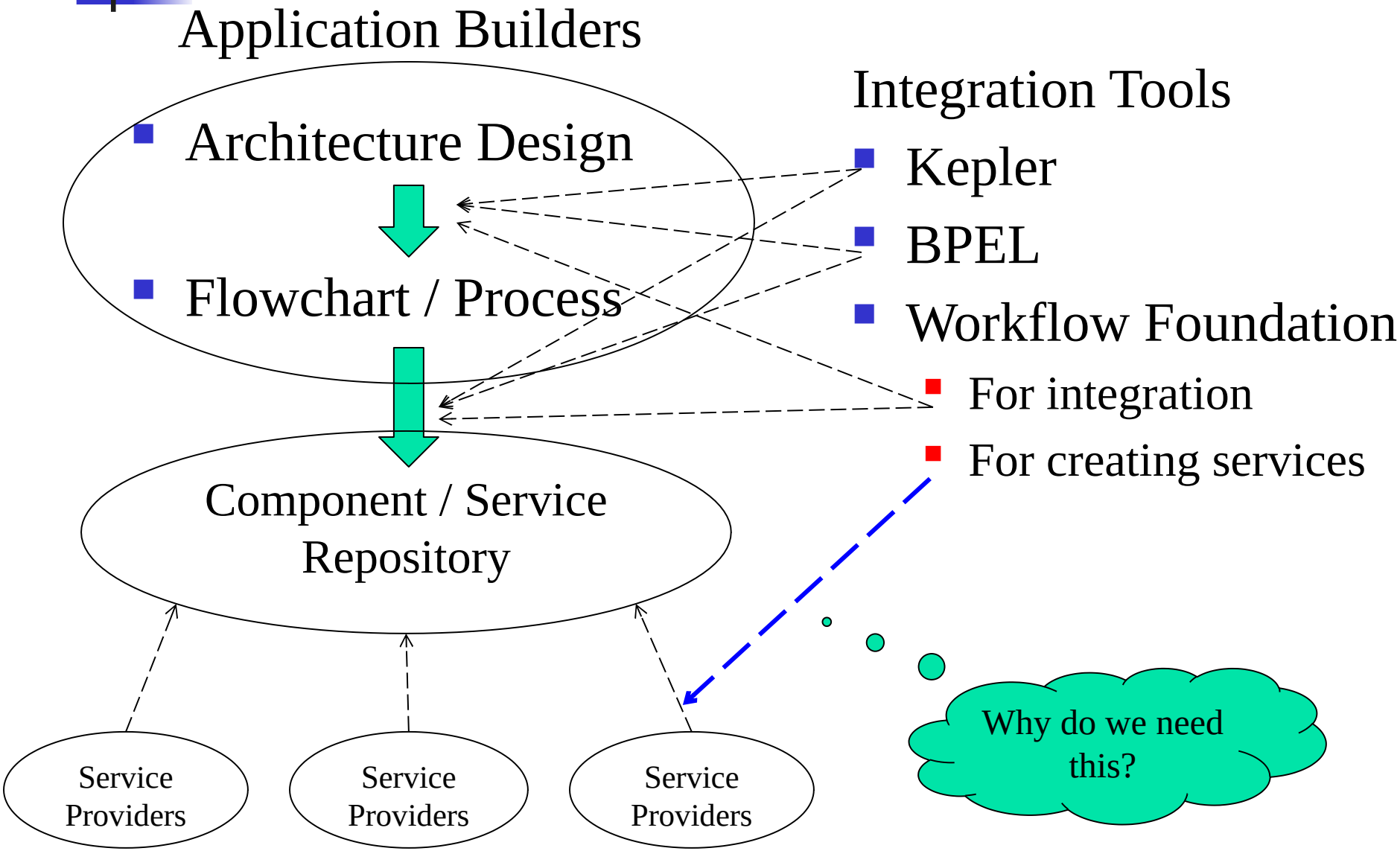
- Testing / Verification

Testers

- Deployment / Maintenance

After Sale Supporters

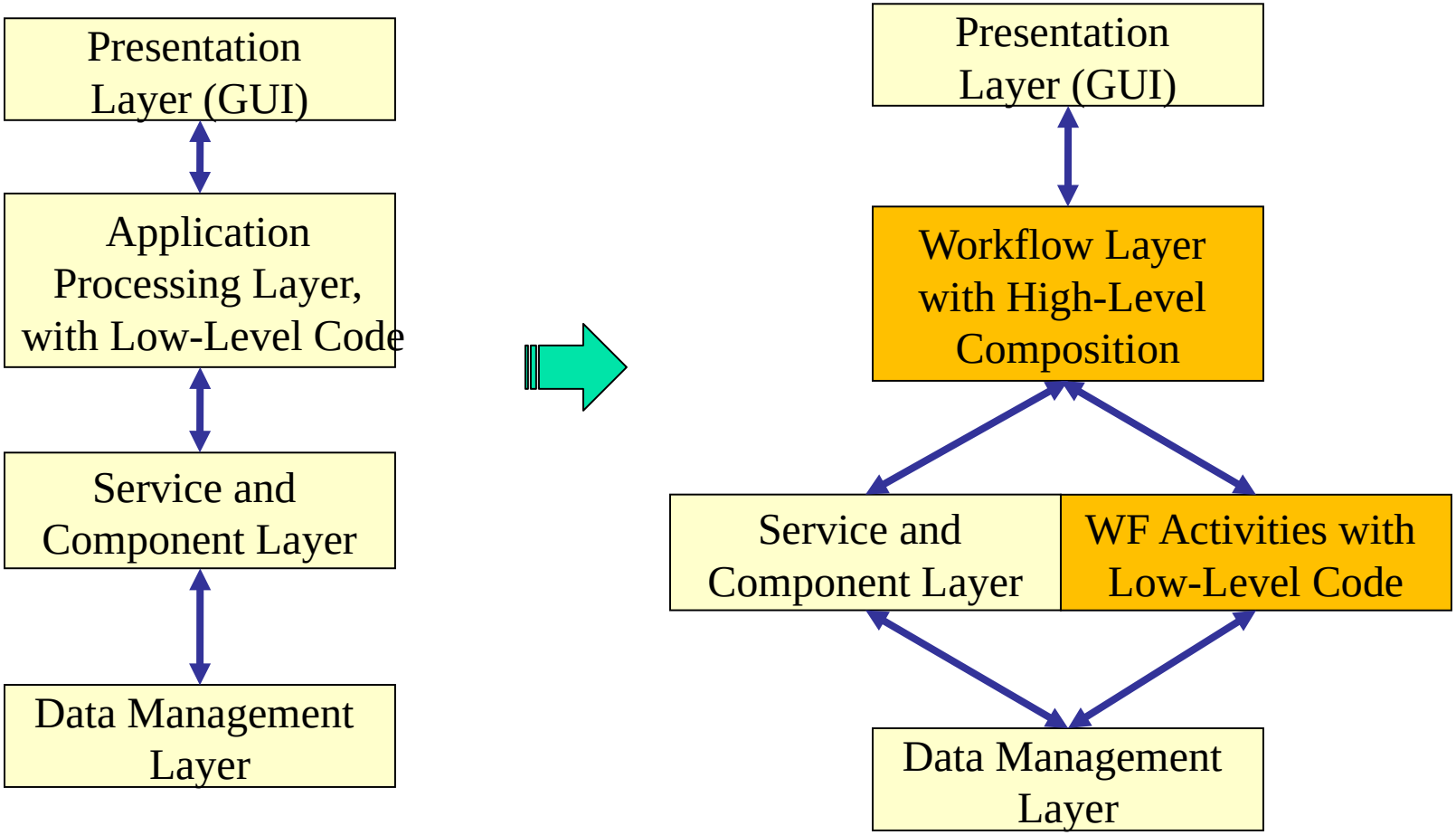
Architecture and Process Integration



Why Do We Need Workflow for Coding?

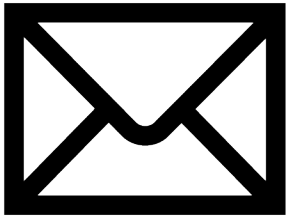
- Adding a layer of abstraction (graphical approach) to make application development faster and easier.
- It makes the application's main logic more visible.
- By providing a straightforward picture of what is going on, the architect can help developers more quickly to understand an application's structure.
- This can be especially helpful for the people who must maintain the deployed applications, since learning a new application well enough to change it can be a time-consuming process.
- Allows traditional programming style to seamlessly integrate with workflow

From Tiered Architecture Perspectives



Example: How Do I Write Code to Send a Mail/Message to a Client in My Web Application?

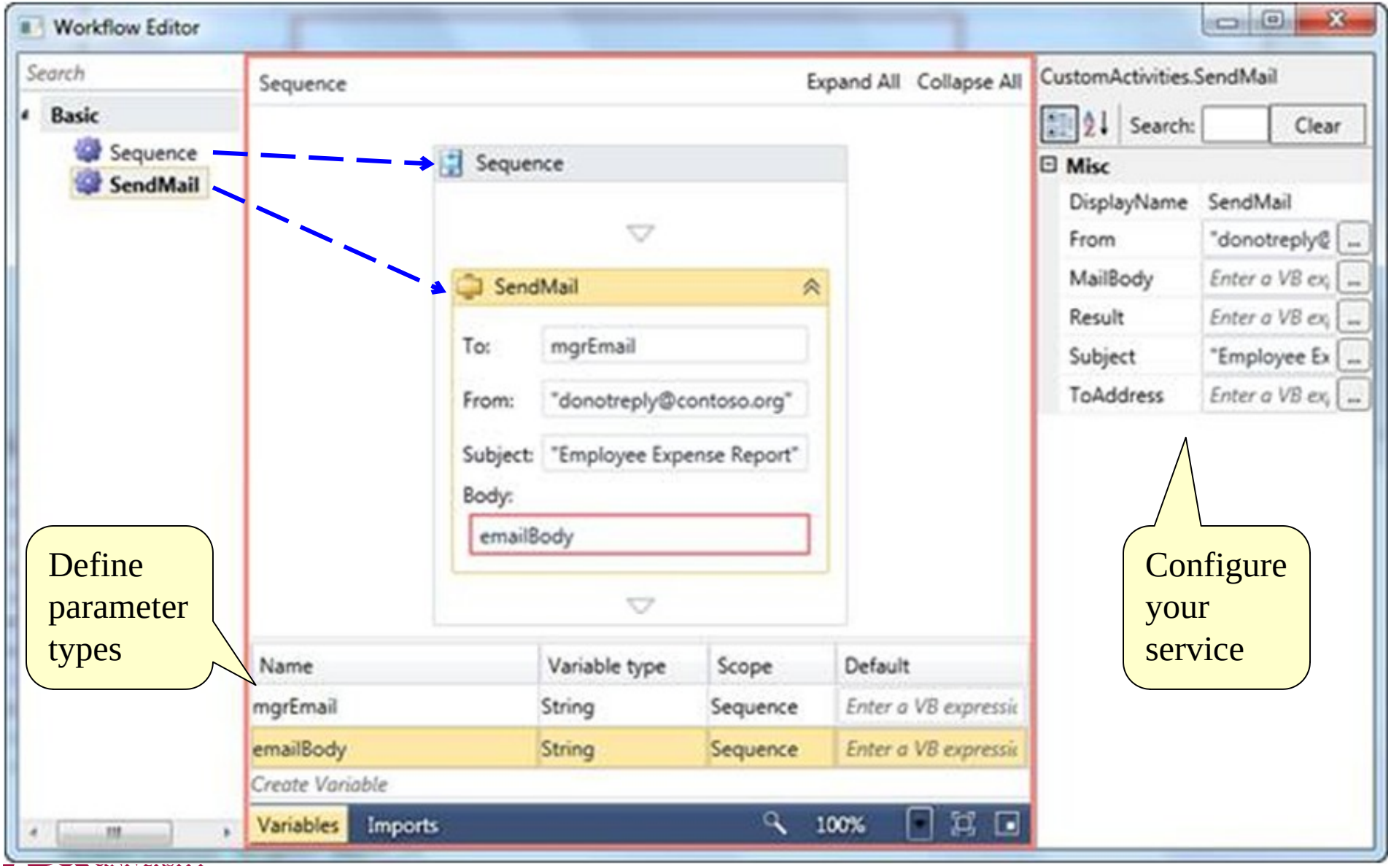
```
MService.ServiceClient TestSvcProxy = new MService.ServiceClient();
string EmailAddress = "account@gmail.com";
string Password = "password";
MService.CARRIER PROVIDER = MService.CARRIER.NAME;
string address, Message;
System.Console.Write("Please enter an address: ");
Address = System.Console.ReadLine();
System.Console.Write("Please enter the service provider: ");
System.Console.ReadLine();
System.Console.WriteLine("\nPlease enter a message: ");
Message = System.Console.ReadLine();
bool Result = TestSvcProxy.SMS(EmailAddress, Password, address,
PROVIDER, Message);
if (Result)
    System.Console.WriteLine("\n\nMessage has been sent.");
```



Mail Service

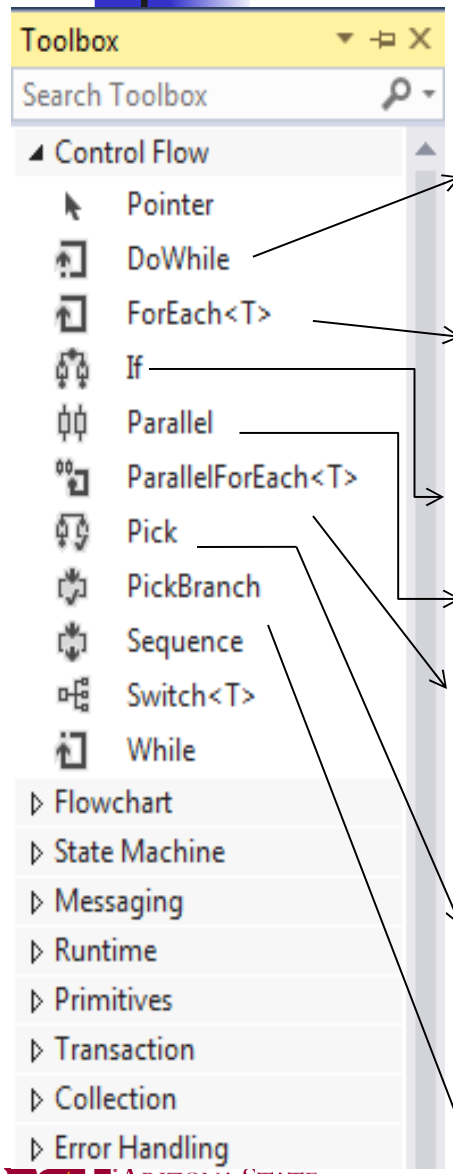
Example: Drag and Drop Designer in WF

http://msdn.microsoft.com/en-us/library/ee342461.aspx



WF Built-in Activities: Control Flow (1)

http://msdn.microsoft.com/en-us/library/dd647759.aspx



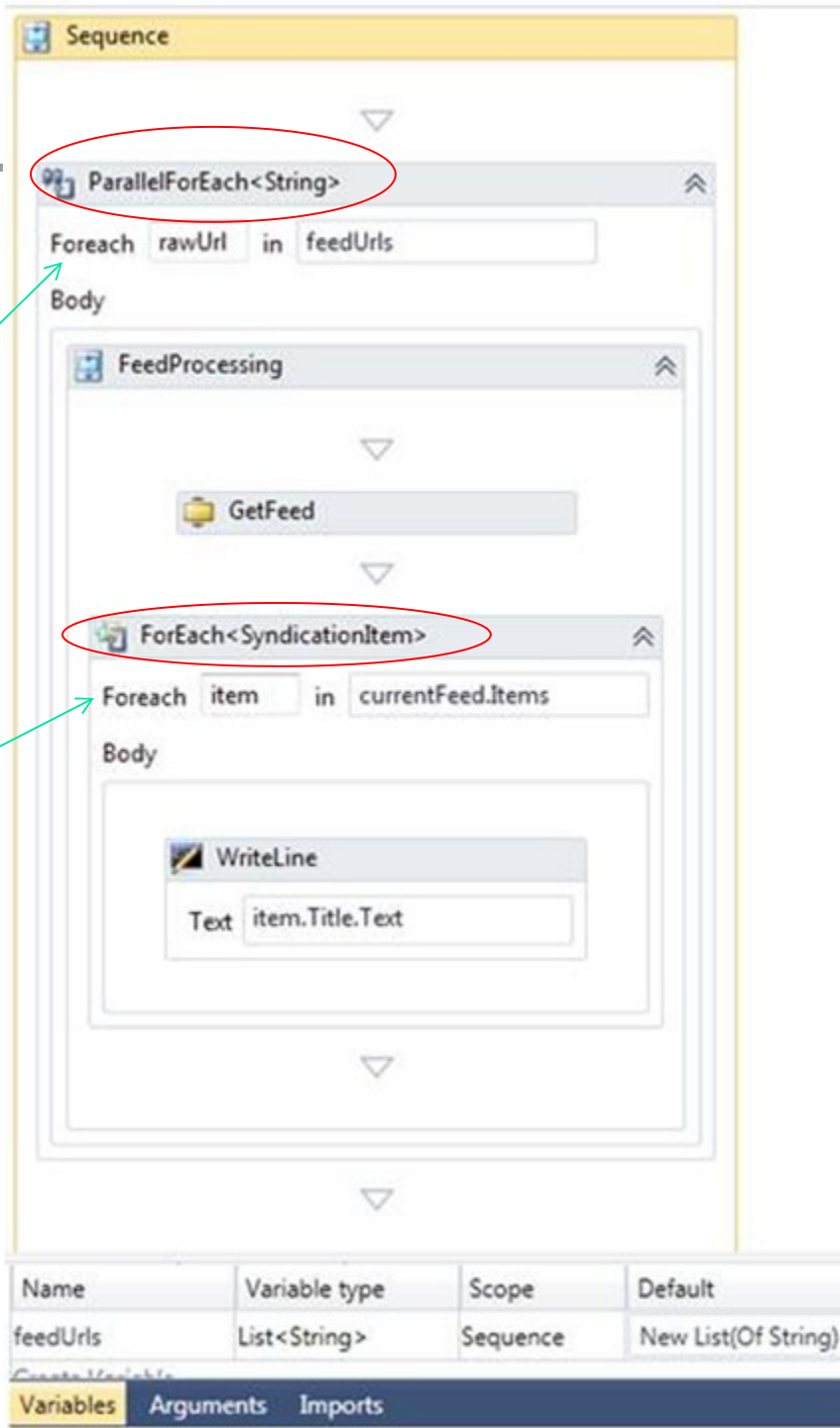
- **DoWhile**: executes an activity, then checks a condition, and repeats until the condition is false.
- **ForEach**: executes an activity for each object in a collection.
- **If**: creates a branch of execution.
- **Parallel**: forks multiple activities.
- **ParallelForEach**: Enumerates the elements of a collection and executes an embedded statement for each element of the collection in parallel.
- **Pick**: allows waiting for a set of PickBranch events, then executing only the activity associated with the first event to occur; event-driven programming!
- **PickBranch**: a member of Pick activity

ParallelForEach and ForEach

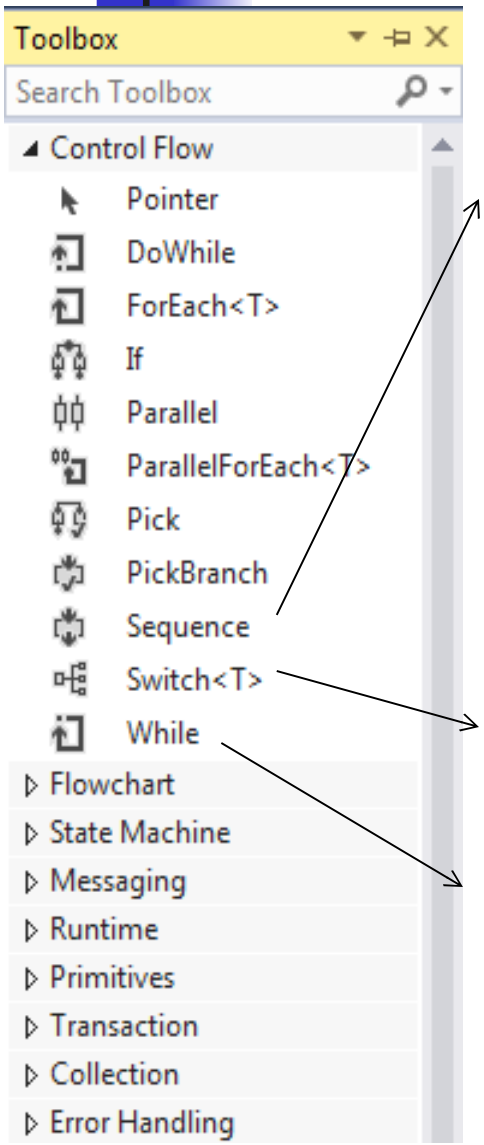
http://msdn.microsoft.com/en-us/library/ee342461.aspx

Example: Call many URLs in parallel:

- Take a list of URLs and asynchronously get all RSS feeds using Foreach <URL>.
- After the feed is returned, the Foreach<item> is used to iterate over the feed items and process them.

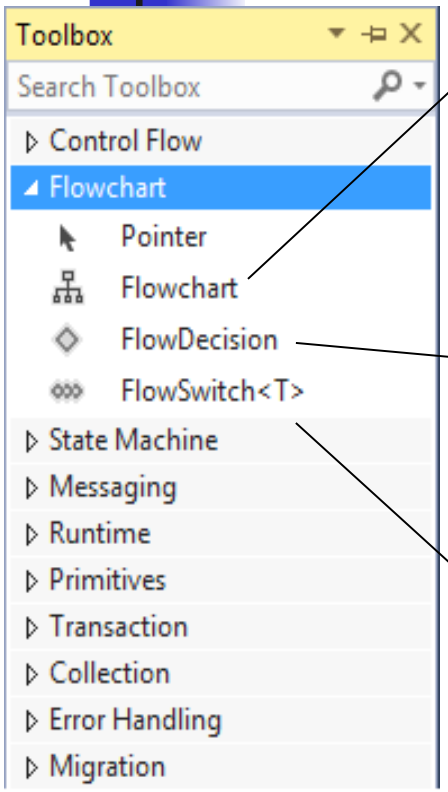


WF Built-in Activities: Control Flow (2)

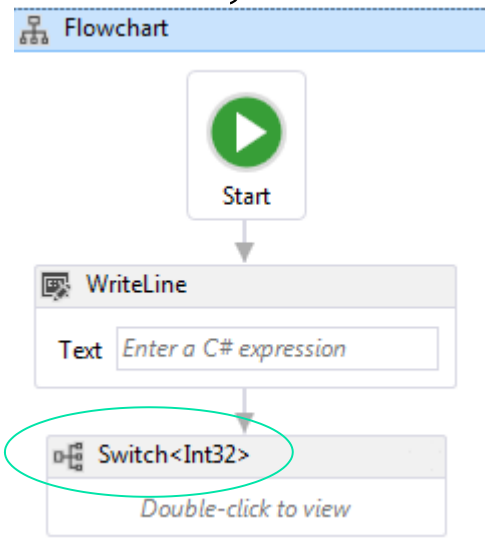


- **Sequence**: groups together a set of activities that are executed sequentially. Sequence is also useful inside workflows. For example, a While activity can contain only one other activity. If that activity is a Sequence, a developer can execute an arbitrary number of activities within the while loop.
- **Switch**: provides a multi-way branch of execution.
- **While**: executes a single activity as long as a condition as true.

WF Built-in Activities: Flowchart (3)

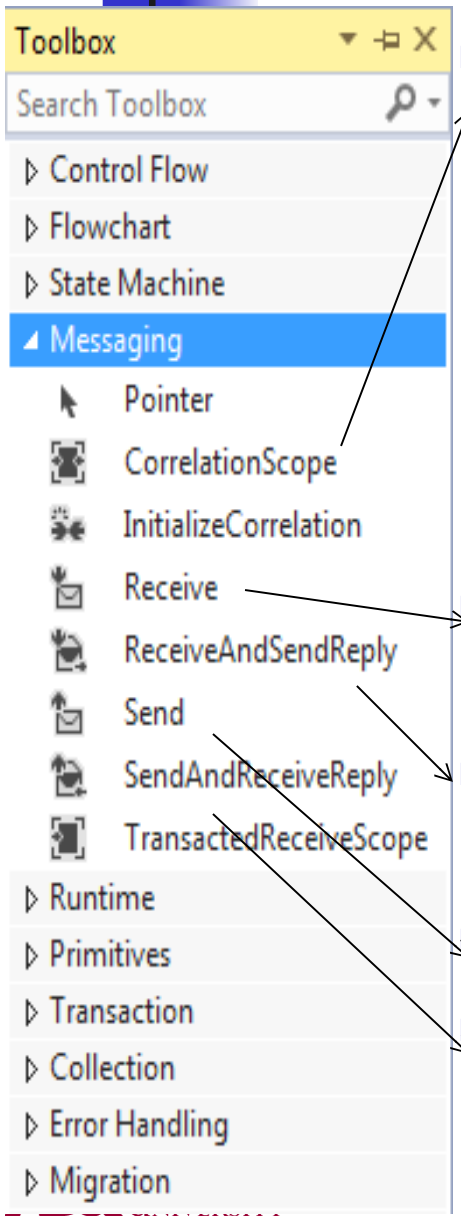


- **Flowchart**: groups together a set of activities that are executed sequentially, but also allows control to return to an earlier step. It creates a local component, similar to CodeActivity's role.
- **FlowDecision**: Branches execution based on a Boolean condition, similar to **If**, but applied at the flowchart level.
- **FlowSwitch**: Branches execution based on an exclusive switch, similar to **Switch** in C#, but applied at the flowchart level.



WF Built-in Activities: Messaging (4)

http://msdn.microsoft.com/en-us/library/dd487617.aspx

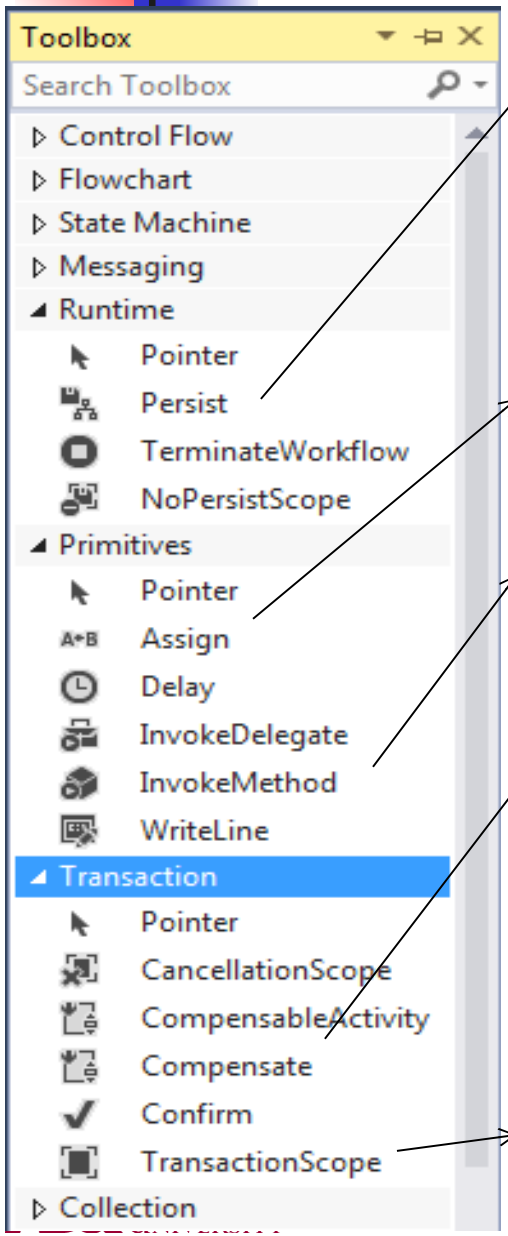


- **CorrelationScope**: Provides implicit CorrelationHandle for child messaging activities. The CorrelationHandle is only visible to child activities, which associates activities together in a correlation by representing a particular shared InstanceKey or transient context in the workflow. Used for asynchronous communication.
- **Receive**: An activity that receives a message via WCF service.
- **ReceiveAndSendReply**: Creates a sequence of two communications.
- **Send**: An activity that sends a message via WCF.
- **SendAndReceiveReply**: Creates an asynchronous communication sequence.

Creating Asynchronous Service in WF

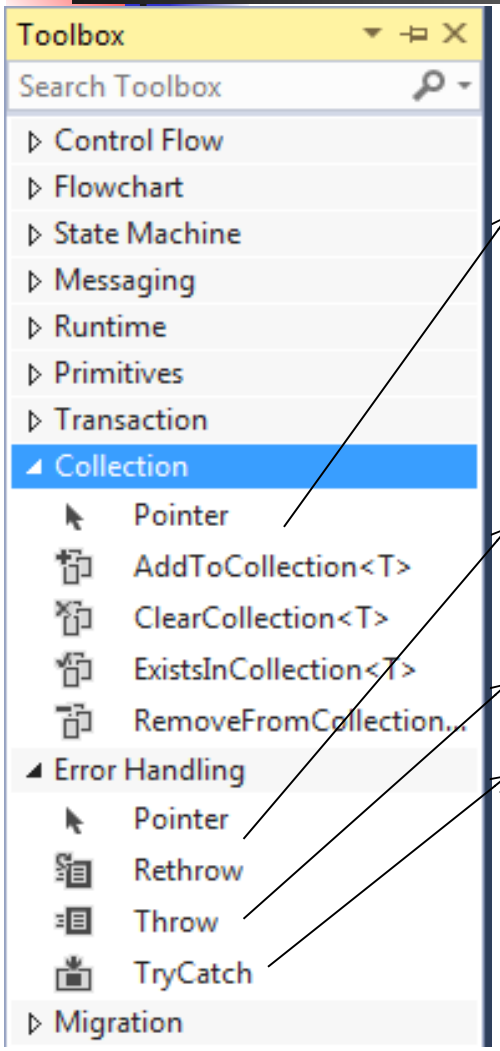
- **SendAndReceiveReply**: is a template used for creating a pair of pre-configured **Send** and **ReceiveReply** activities within a **Sequence** activity
- Send and Receive are correlated as part of an asynchronous request/response message exchange pattern on the client. **SendAndReceiveReply** also
  - Configures the OperationName, ServiceContractName properties of the Send activity.
  - Binds the Request property of the ReceiveReply activity to the Send activity.
  - Creates a CorrelationHandle as a variable in the parent activity.

WF Built-in Activities (5)



- **Persist**: explicitly requests the WF runtime to persist the workflow, to allow a workflow to be re-loaded later on the machine or even on another machine other, if necessary. Save all states.
- **Assign**: assigns a value to a variable in the workflow.
- **InvokeMethod**: Invokes a method on the object, synchronously or asynchronously.
- **Compensate**: An activity used to explicitly invoke the compensation handler of a CompensableActivity. It provides a way of handling a problem that occurs in a long-running transaction.
- **TransactionScope:** Makes a code block transactional. This class cannot be inherited.

WF Built-in Activities (6)



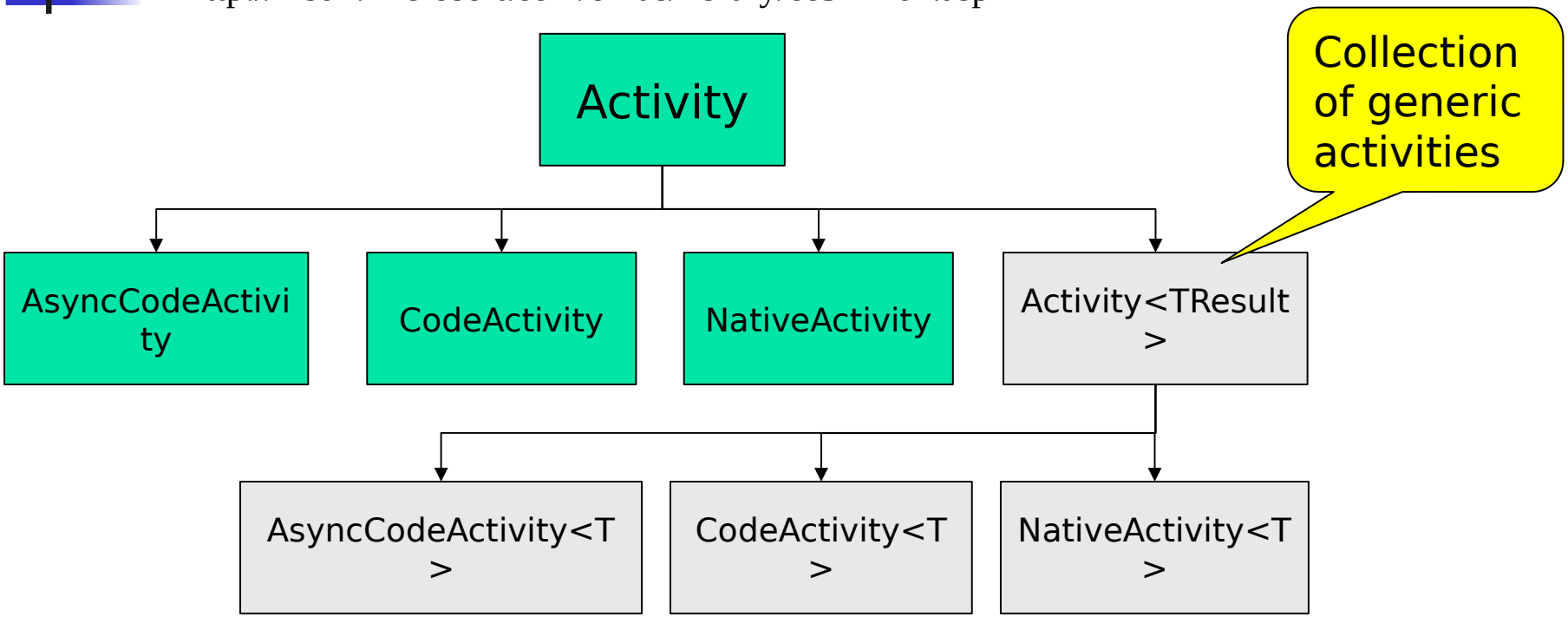
- **AddToCollection**, **ClearCollection**, **ExistInCollection**, **RemoveFromCollection**: allow to add, remove, clear, and check membership of a collection.
- **Rethrow**: throws a previously thrown exception from within a Catch activity.
- **Throw**: raises an exception.
- **TryCatch**: allows creating a try/catch block to handle exceptions. In the block, it contains workflow elements to be executed by the workflow runtime within an exception handling block.

WF Custom Activities

- The developer can wrap any block of C#, VB, and WF code into a custom activity (local component) to integrate into the workflow;
- Custom activities can be simple, performing just one task, or they can be composite activities containing arbitrarily complex logic;
- A business application created using WF might implement application-specific logic as an activity;
- A software vendor using WF might provide a set of custom activities to make its customers' lives easier.

Custom Activity Class Hierarchy

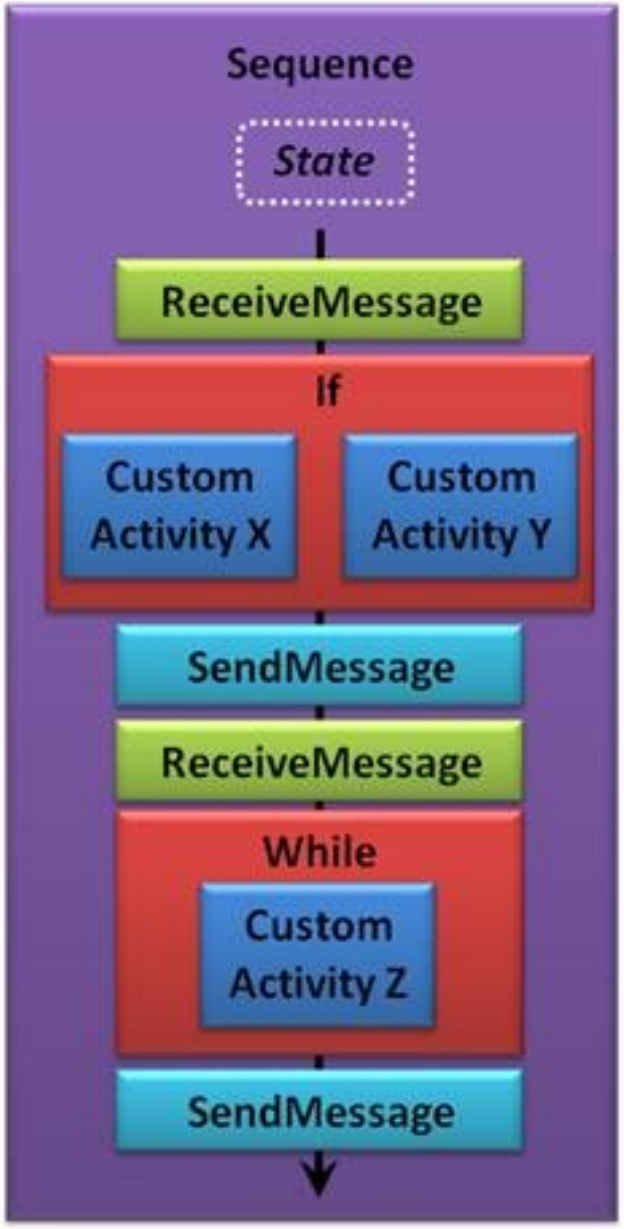
http://msdn.microsoft.com/en-us/library/ee342461.aspx



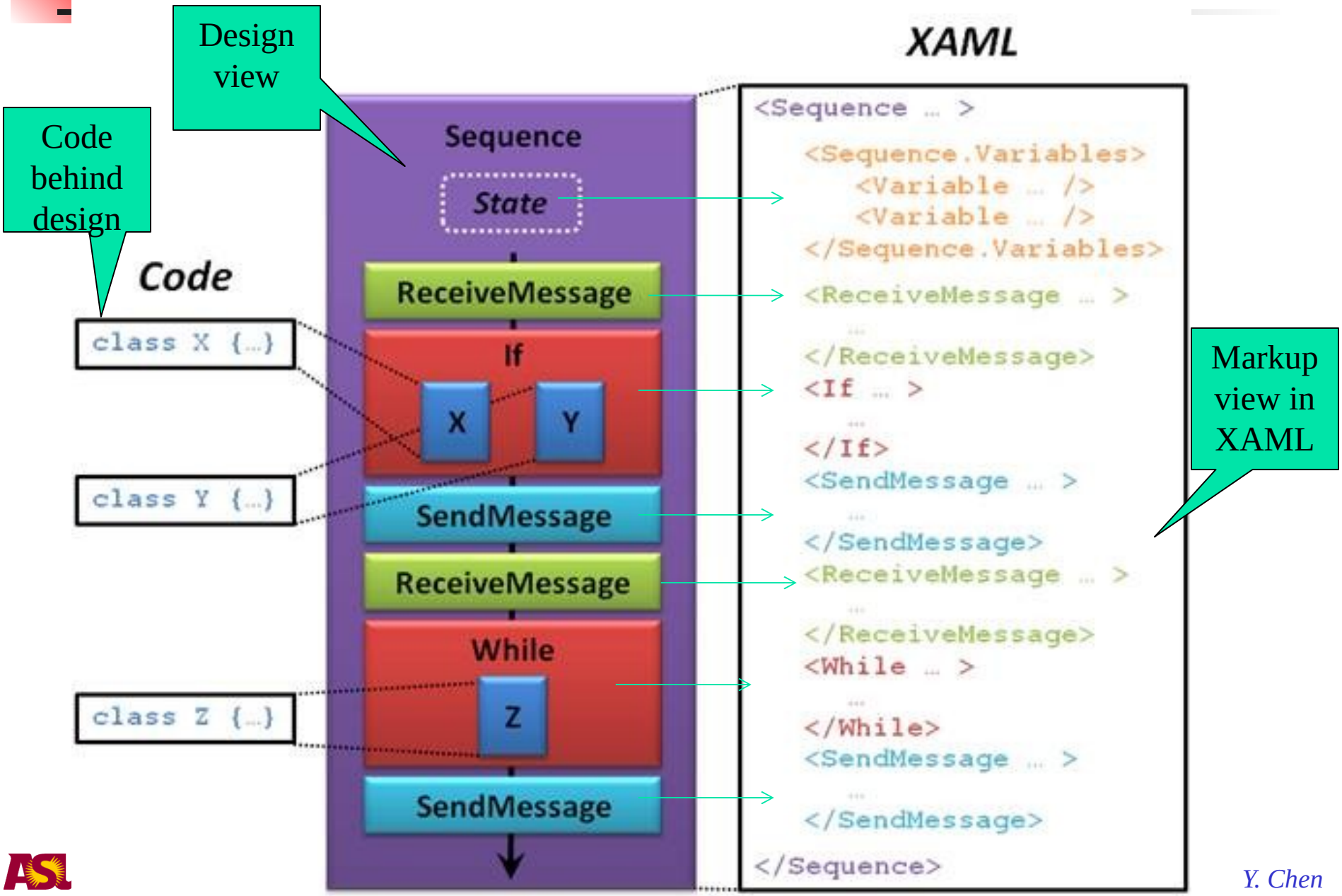
- **Activity** – can compose of other activities.
- **AsyncCodeActivity** – used when your activity perform work asynchronously.
- **CodeActivity** – Creating a local component when you need to write code to do the job.
- **NativeActivity** –when your activity needs access to the runtime internals, for example to schedule other activities or create bookmarks.

A Typical Workflow of Client Service

http://msdn.microsoft.com/en-us/library/dd851337.aspx



WF Designer, Markup and Code Behind



Creating a Workflow Service

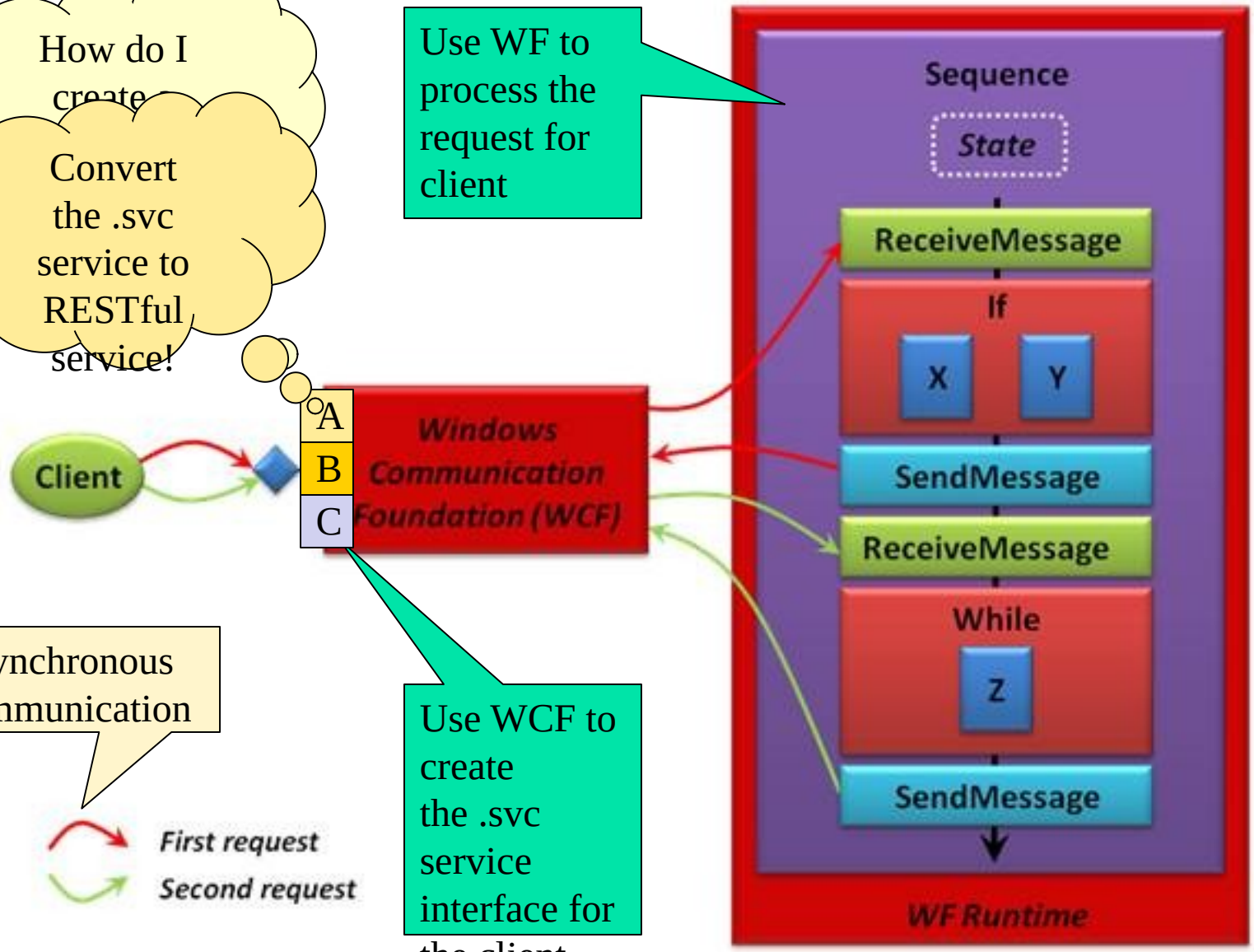
- Building business logic as a service often makes sense, for example, when the same set of operations must be accessed from a
  - Console application
  - Windows forms application;
  - Website application.
- Implementing this logic as a service consisting of a set of operations can make these applications to interoperate with the operations;
- WCF interface can be used for creating SOAP and REST interface for WF service.

WF Using WCF for Creating Services

How do I create a

Convert the .svc service to RESTful service!

Use WF to process the request for client



Asynchronous communication

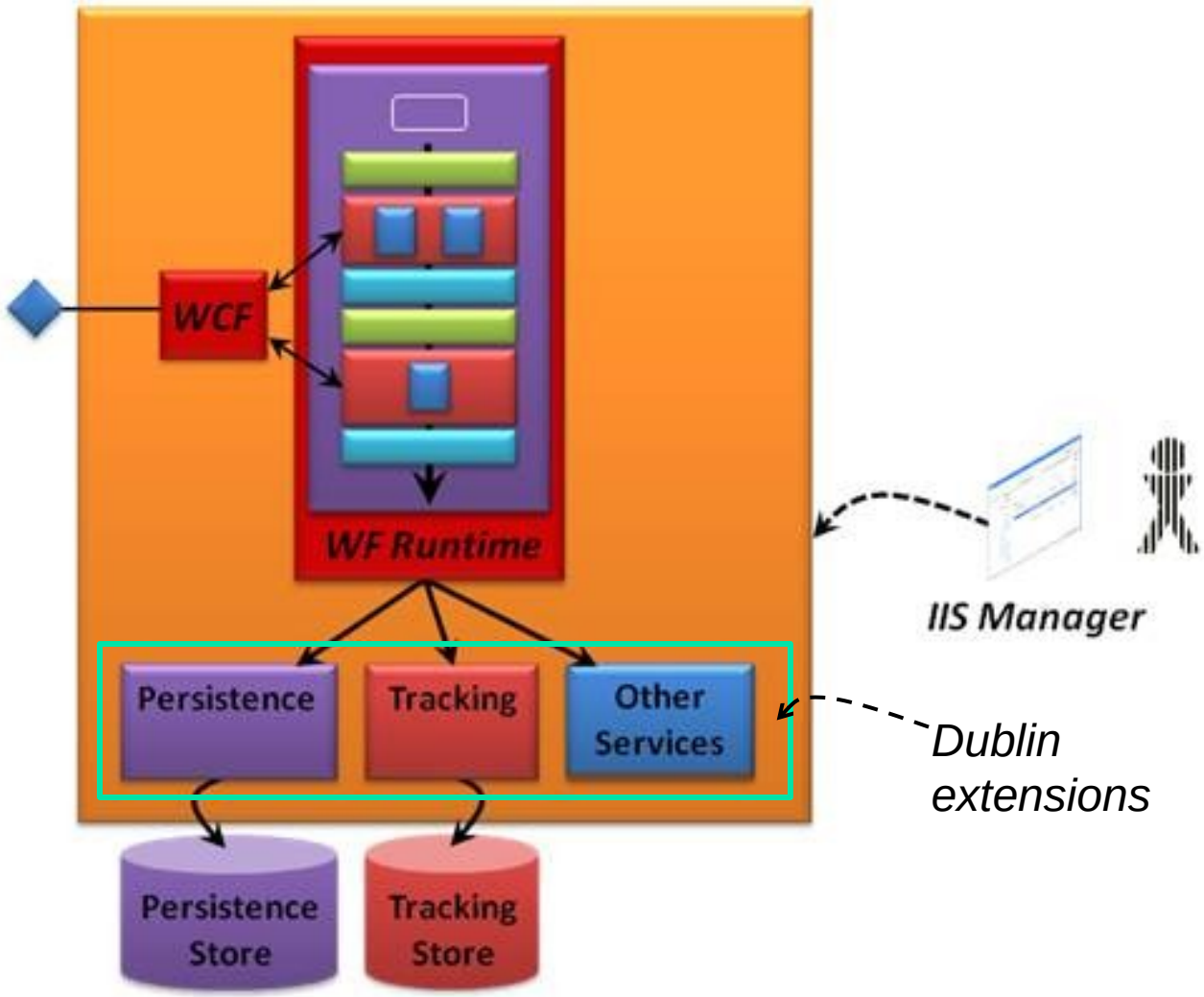
Use WCF to create the .svc service interface for the client.

Hosting WF Service

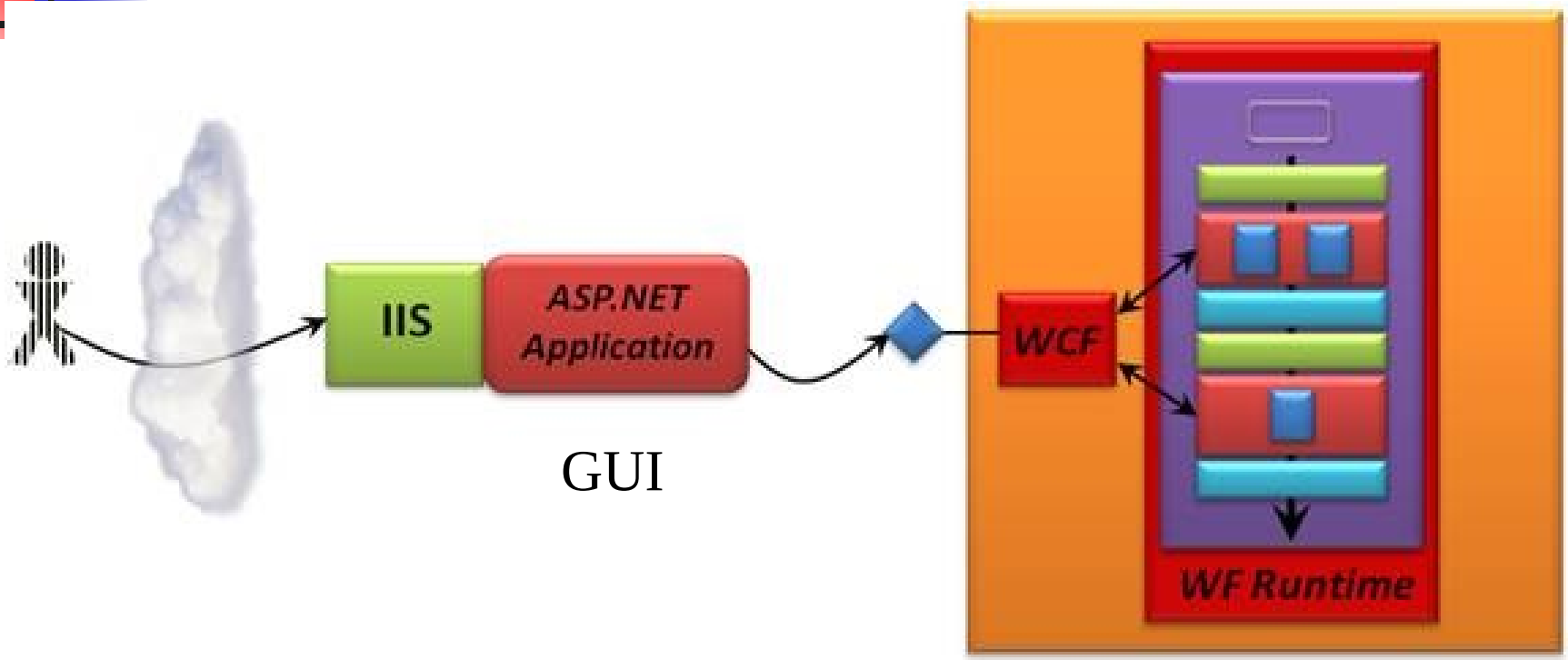
- Similar to WCF, WF services can be hosted in different ways:
  - Self-hosting
  - IIS hosting: running a thread of a worker process
  - "Dublin" hosting
- "Dublin" hosting is based on IIS with extensions for additional support:
  - Persistence management: Saving the process (code and states for later access) and waiting for the second part of service call persistently;
  - Tracking and monitoring
  - Other utility services

Hosting WF Service with Dublin

- IIS provide basic hosting service
- Dublin is the code name for extended IIS and WAS (Website Administration Services) for WF service hosting
- The primary goal of "Dublin" is to make IIS and WAS more attractive as a host for workflow services.
  - Persistence
  - Tracking



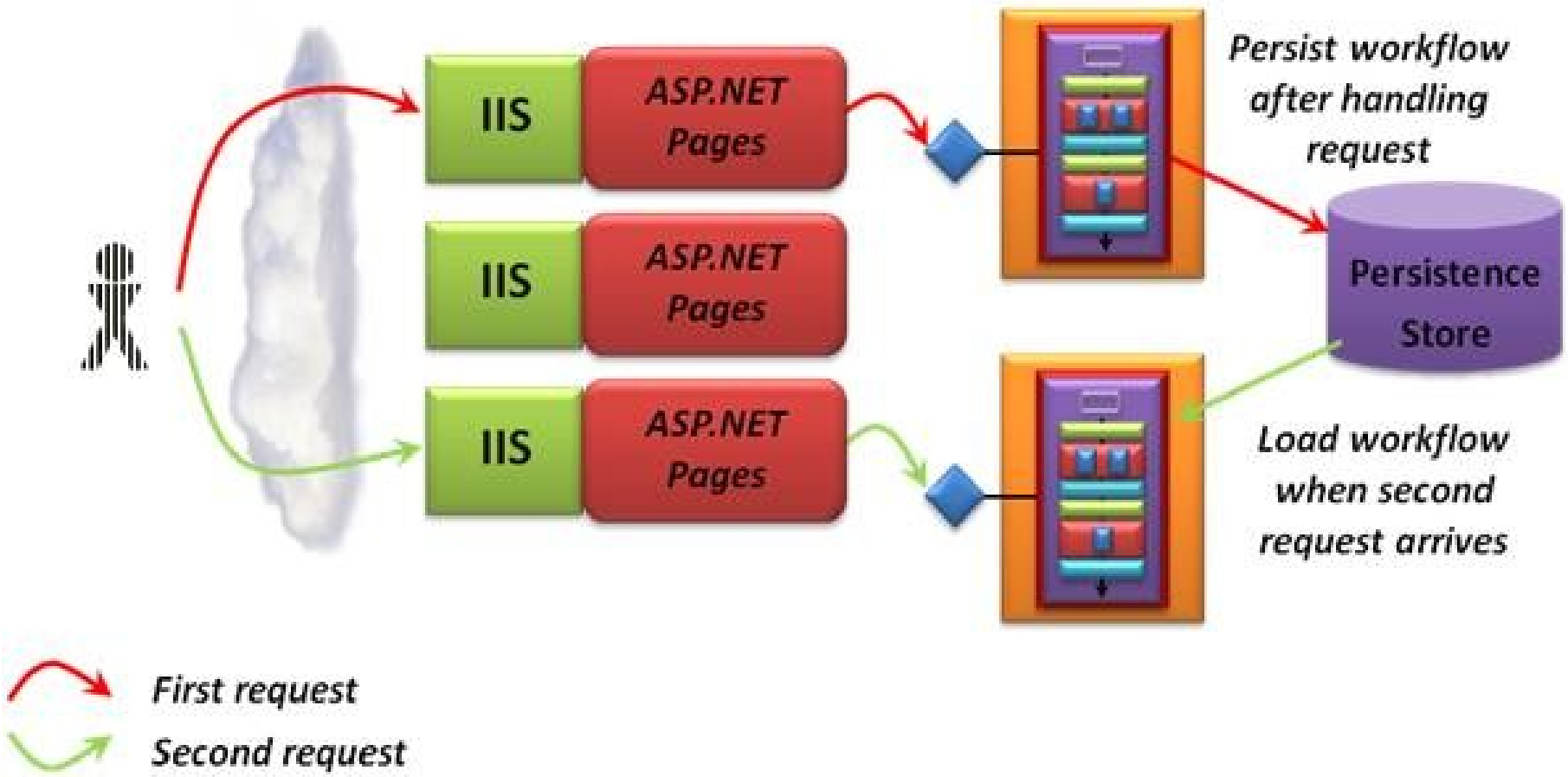
ASP .Net GUI for WF Services



Why do we use WF service behind ASP .Net? We could use code behind ASP .Net page to implement the functionality.

- Tier design: separating presentation layer from application logic layer;
- WF can manage states and asynchronous communication easier than ASP .Net
- With parallel construct and parallel threading, WF is easier for implementing parallel computing.

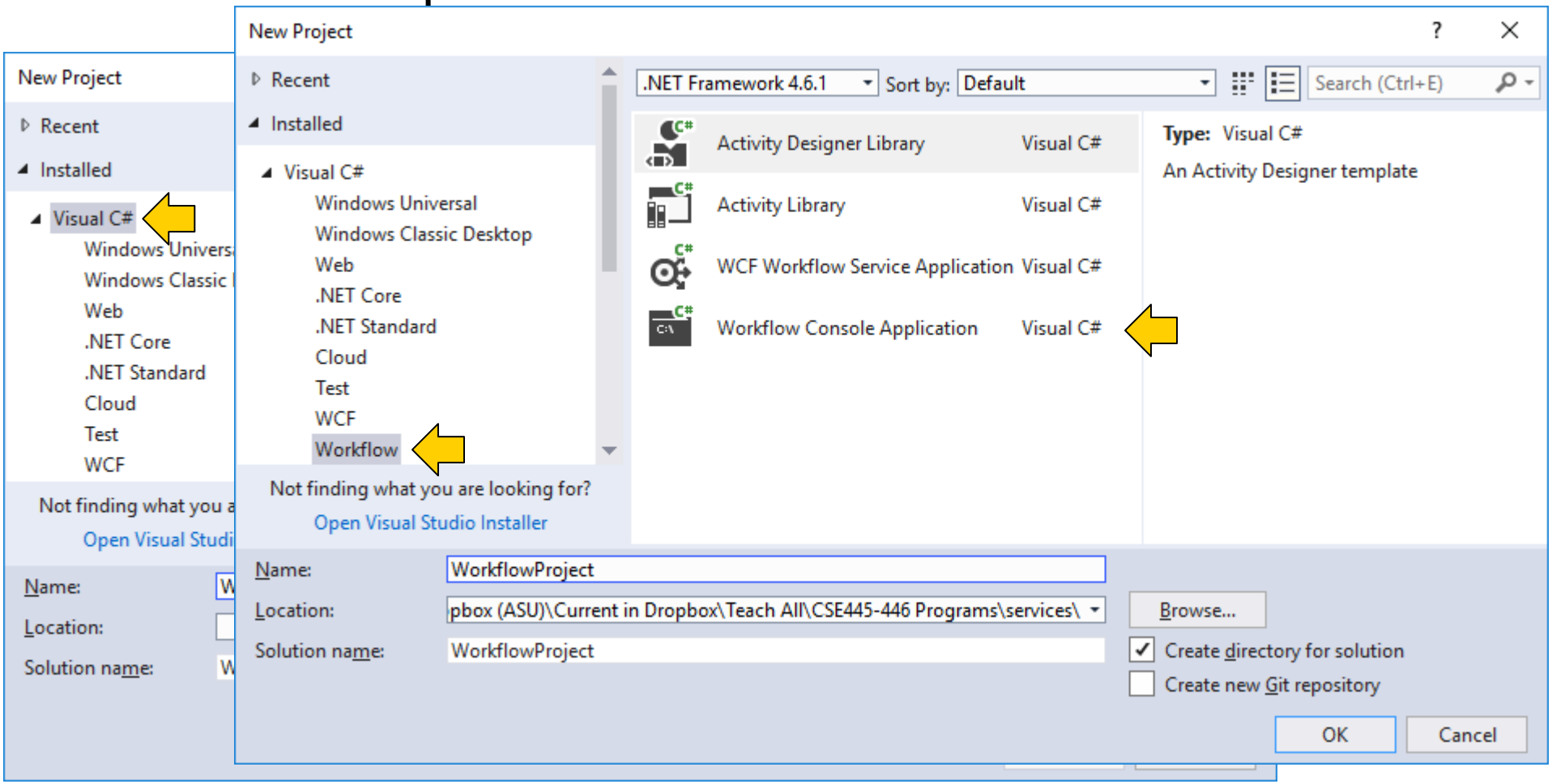
Concurrent Objects for WF Services



Multiple instances can be created for a single client's multiple requests

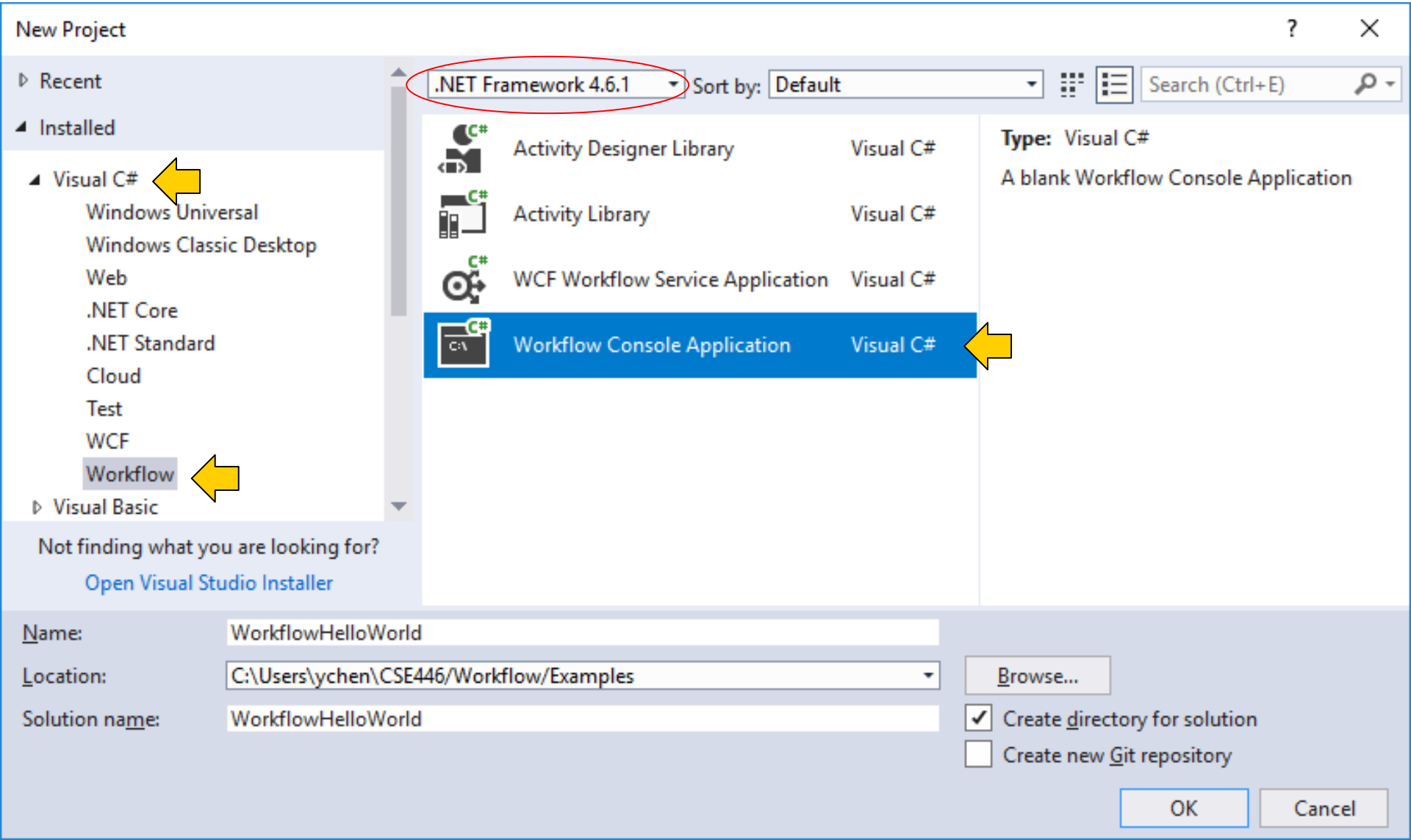
Getting Started With WF Development

- In VS 2017, if you do not see Workflow template, choose Open Project, and open an existing Workflow project.
- Workflow template will be added

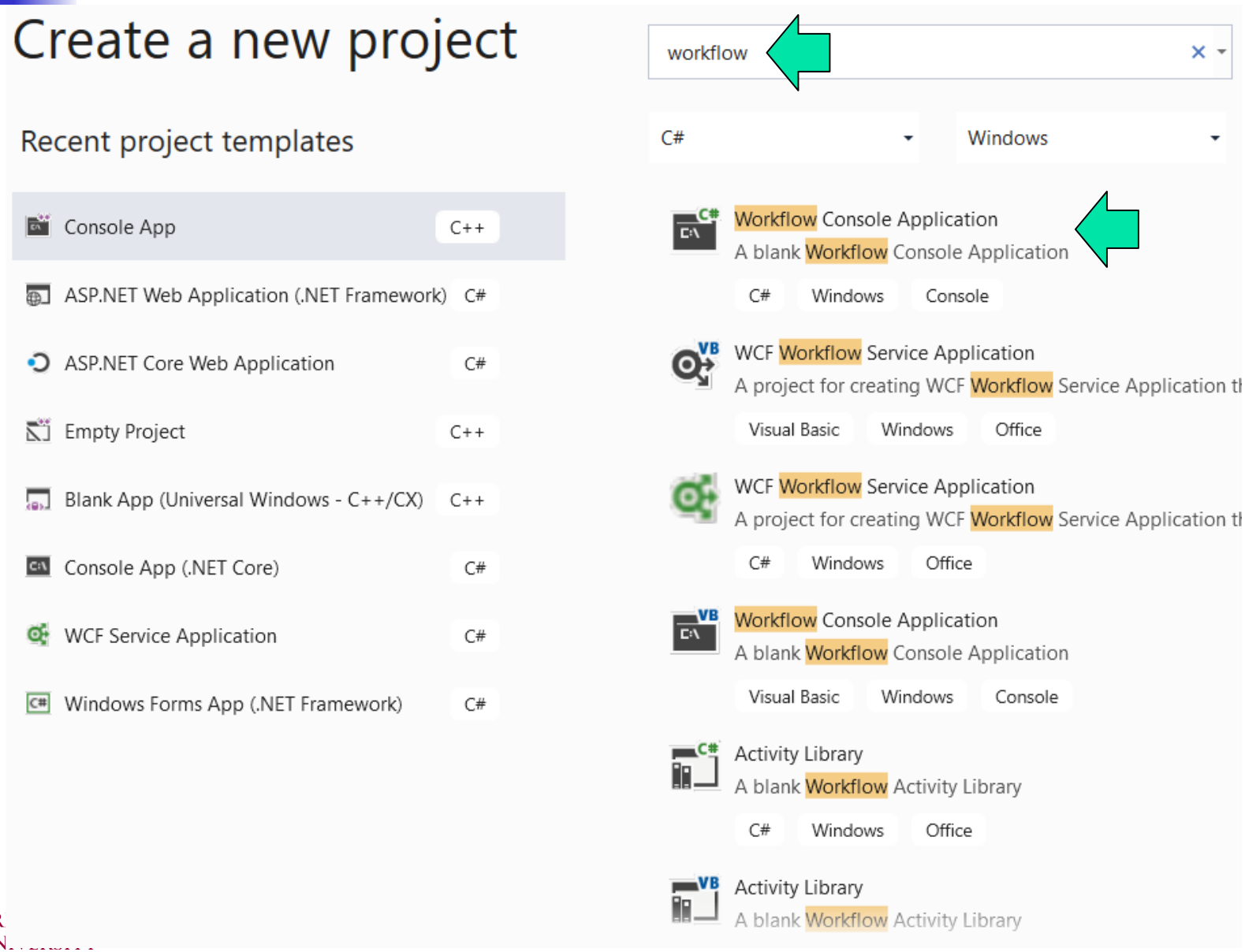


Getting Started With WF Development

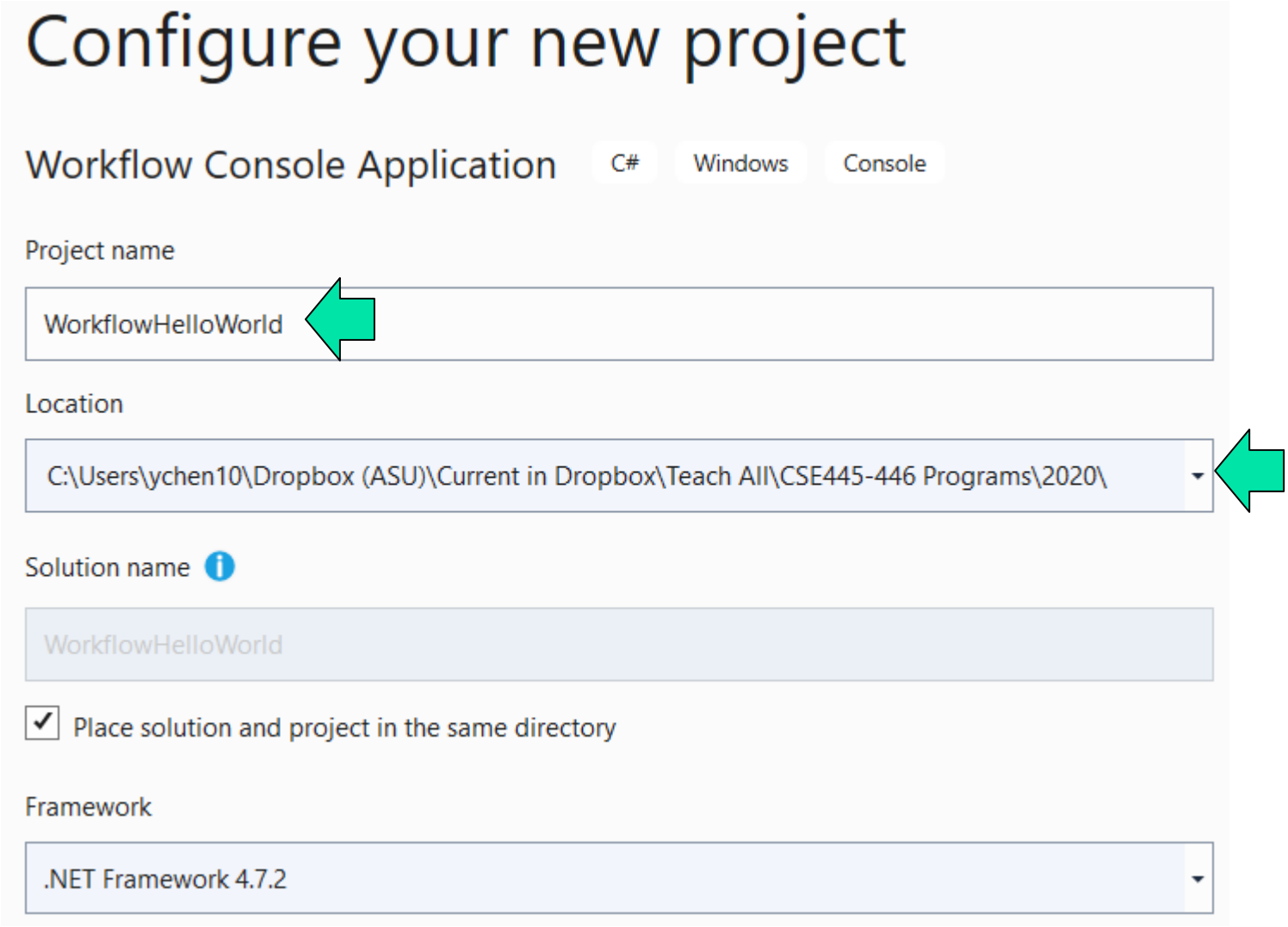
- In VS, File New Project …



Getting Started With WF Visual Studio 2019

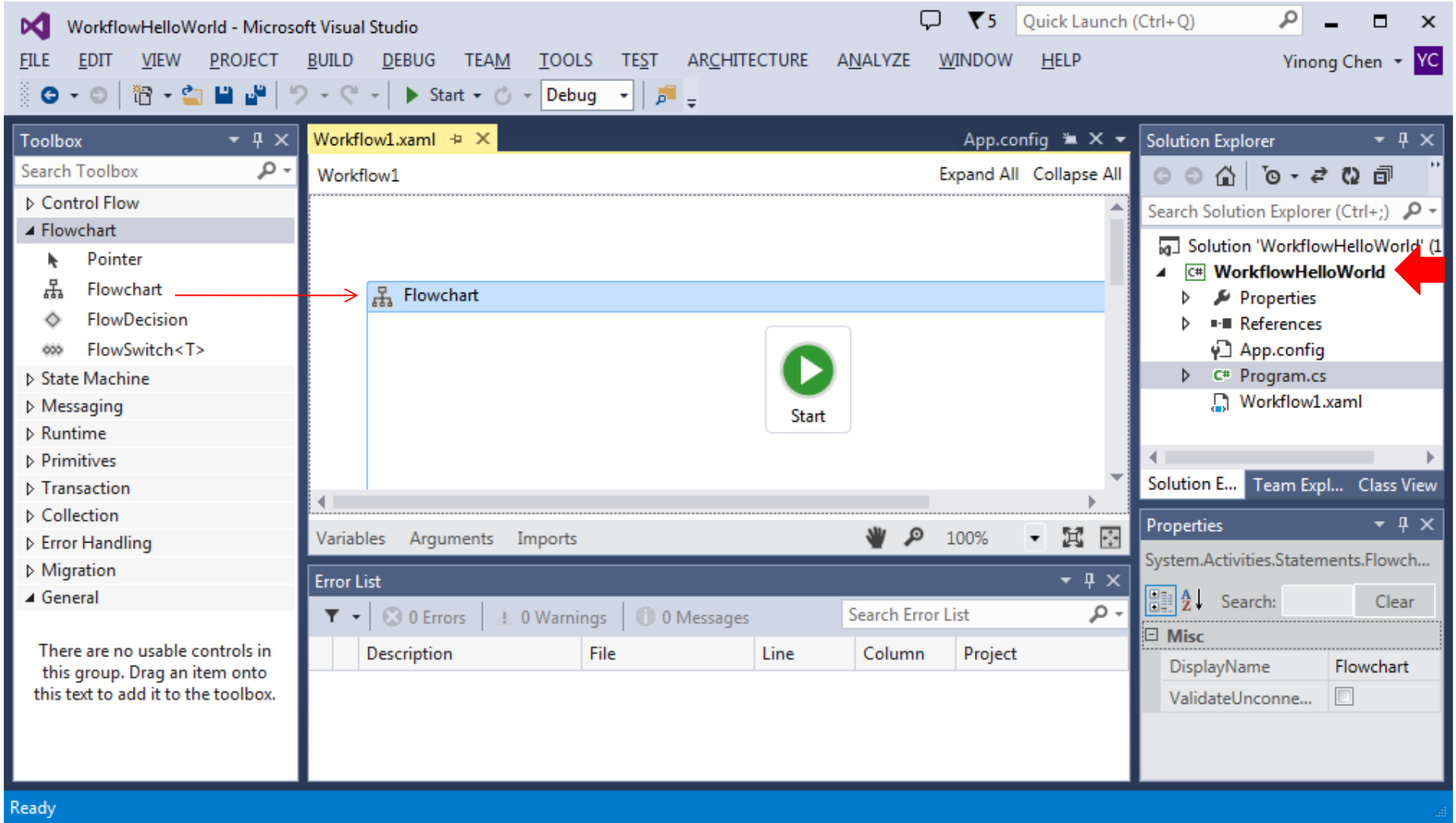


Choose Project Name and Location



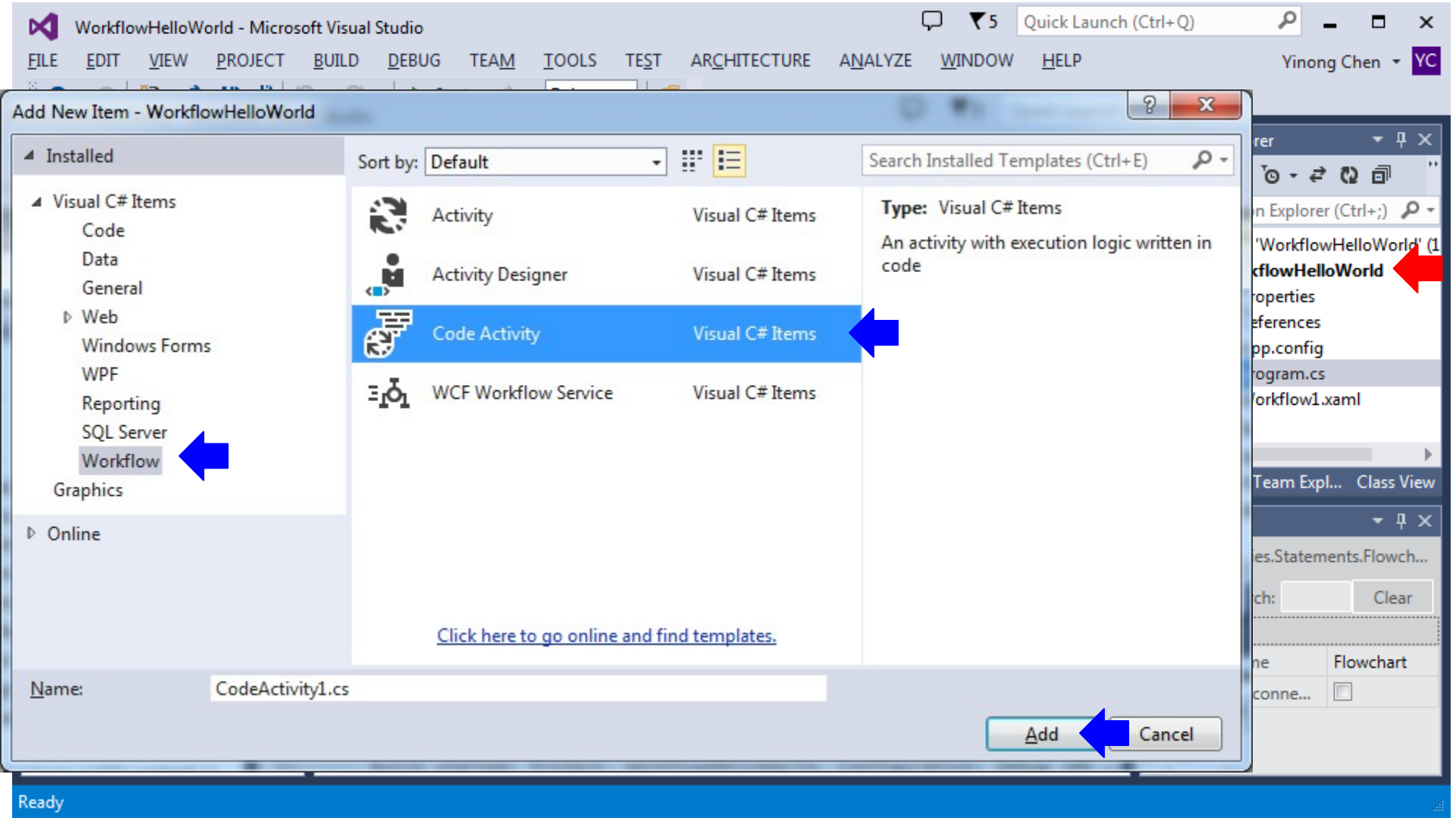
Adding a Flowchart Item into Workflow

1. Add Flowchart into the Workflow
2. Right click project name “WorkflowHelloWorld” and Choose “Add New Item”



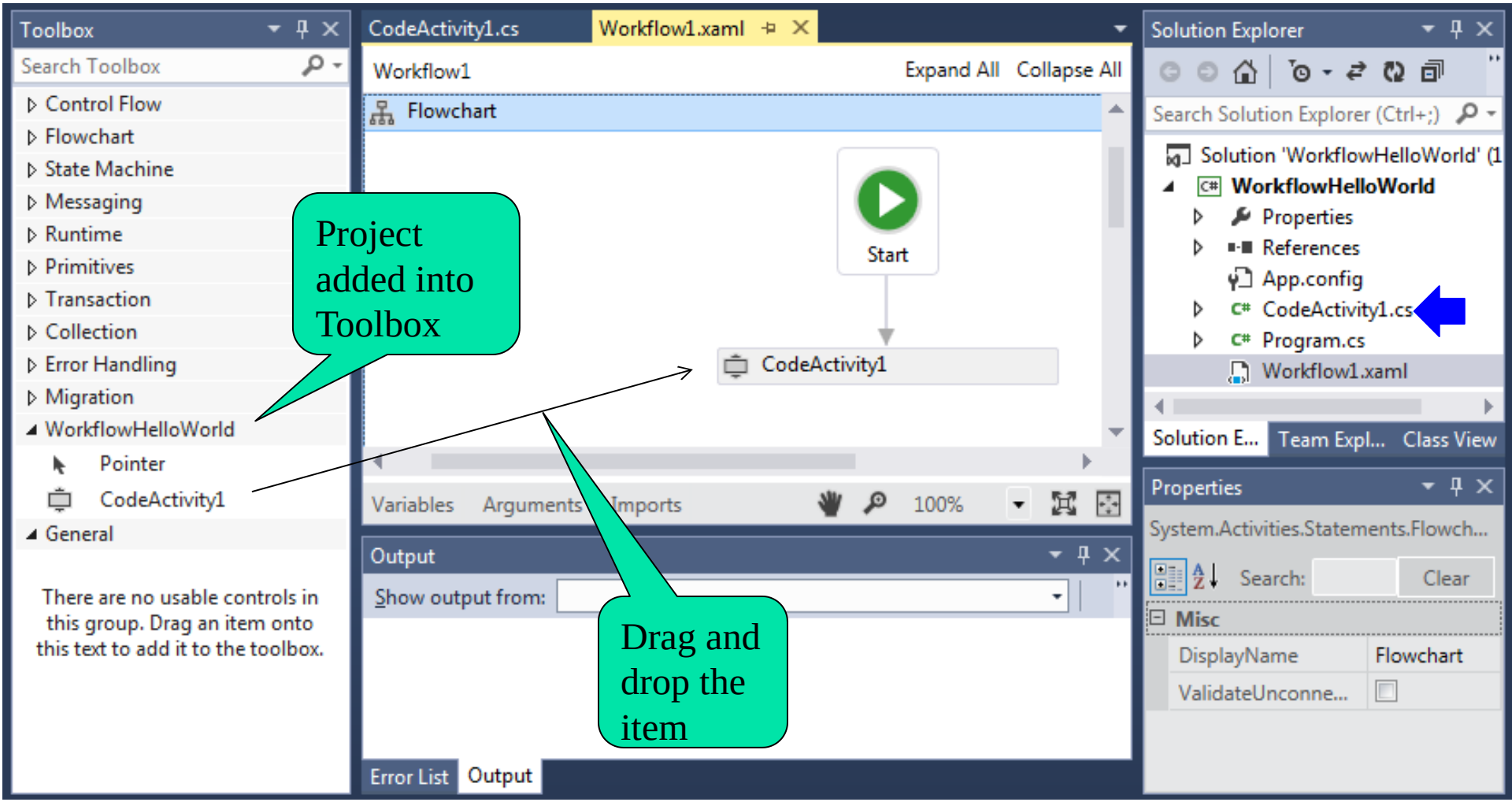
Adding a Flowchart Item into Workflow

1. Right click project name “WorkflowHelloWorld” and Choose “Add New Item”
2. Choose CodeActivity and Add



Creating a Workflow

1. CodeActivity1 is added into the project
2. Build the project, the WorkflowHelloWorld project is added into the Toolbox
3. Drag CodeActivity1 from Toolbox into the workflow



Code Behind the CodeActivity1

CodeActivity1.cs | Workflow1.xaml

WorkflowHelloWorld.CodeActivity1 | Execute(CodeActivityContext context)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;

namespace WorkflowHelloWorld
{

    0 references
    public sealed class CodeActivity1 : CodeActivity
    {
        // Define an activity input argument of type string
        1 reference
        public InArgument<string> Text { get; set; }

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

100 %

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'WorkflowHelloWorld' (1
- WorkflowHelloWorld
  - Properties
  - References
  - App.config
  - CodeActivity1.cs
  - Program.cs
  - Workflow1.xaml

Code Behind the Main Program

Program.cs | CodeActivity1.cs | Workflow1.xaml

WorkflowHelloWorld.Program | Main(string[] args)

```
using System;
using System.Linq;
using System.Activities;
using System.Activities.Statements;

namespace WorkflowHelloWorld
{

    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Activity workflow1 = new Workflow1();
            WorkflowInvoker.Invoke(workflow1);
        }
    }
}
```

Solution Explorer

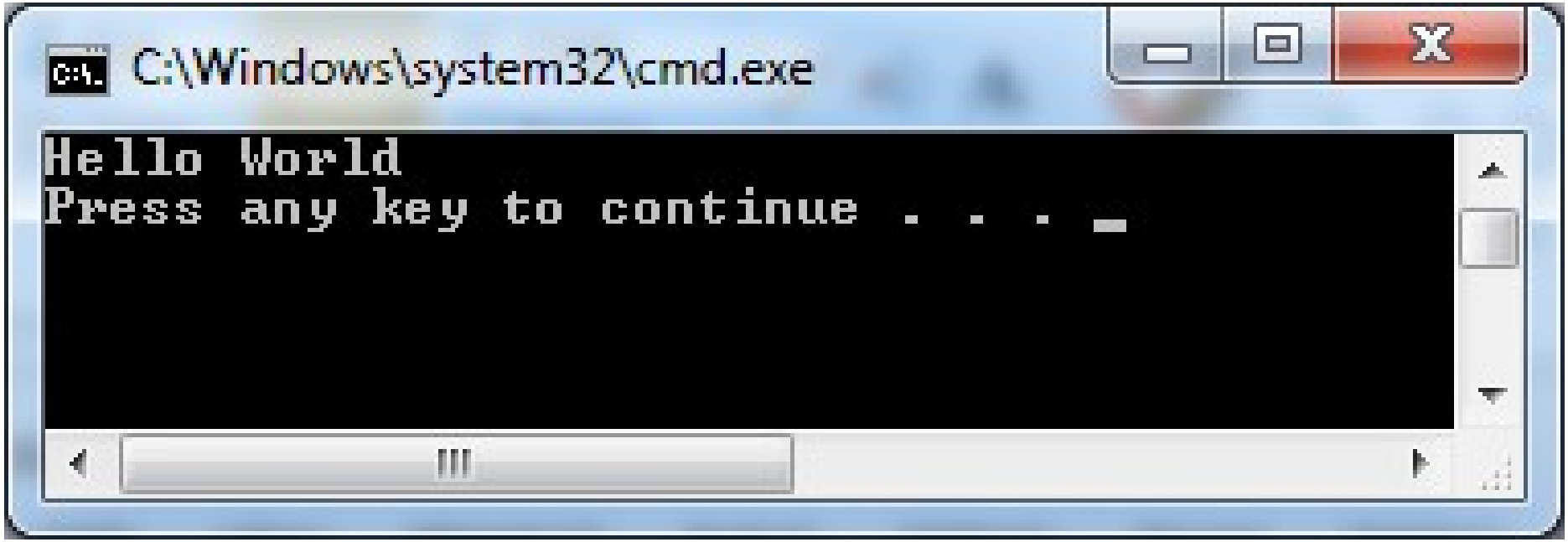
Search Solution Explorer (Ctrl+;)

- Solution 'WorkflowHelloWorld' (1
- WorkflowHelloWorld
  - Properties
  - References
  - App.config
  - CodeActivity1.cs
  - Program.cs
  - Workflow1.xaml

Solution E... | Team Expl... | Class View

Properties

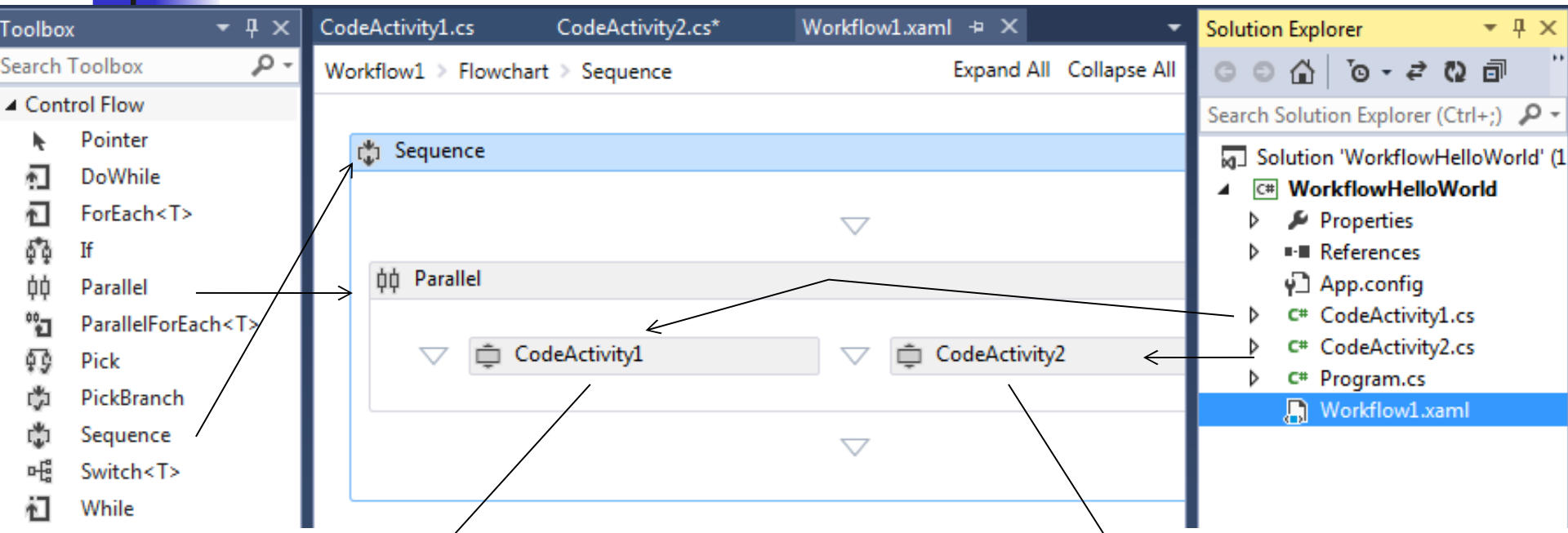
Executing the Code



Workflow: Support Control Flow and Data Flow

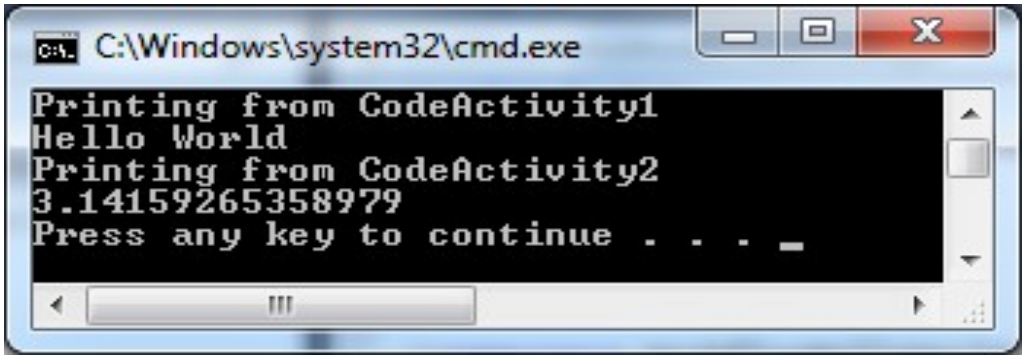
- Control flow model: Assuming one processor will process the entire program, and thus, the code are organized in a **sequence**;
  - It is possible to use multithreading programming to perform parallel computing, but not easy.
- Data flow model: Assuming multiple processors will process different part of the program. A piece of code will be executed when the input **data** is ready.
- Workflow supports multiple computing models
  - Sequence construct
  - Parallel construct
  - Finite State Machine Model

Adding Constructs for Parallel Computing



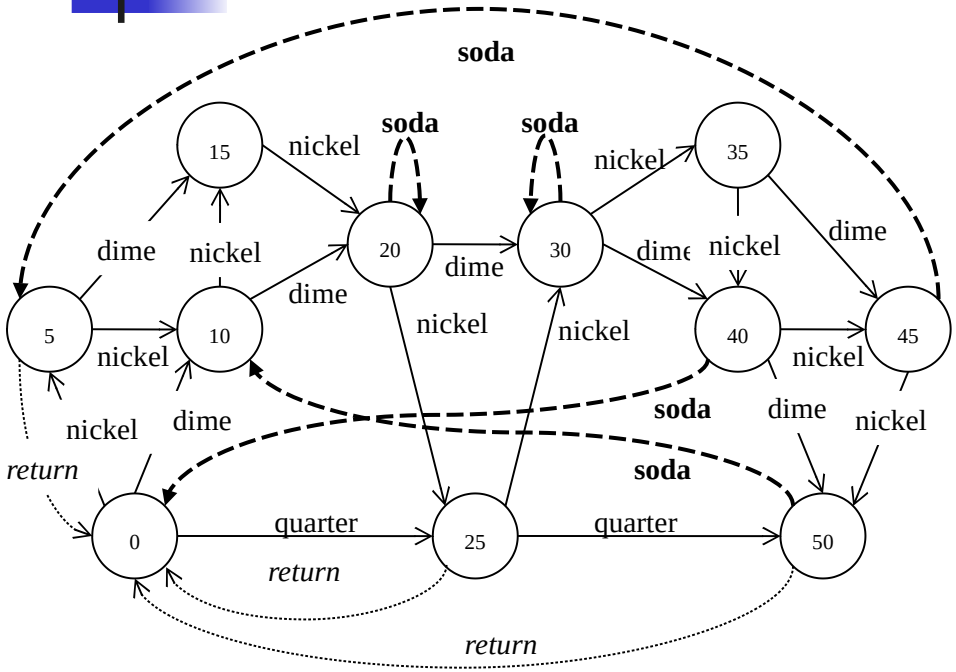

```
Console.WriteLine("Printing from CodeActivity1"); // Thread 1
Console.WriteLine("Hello World"); // Add this line of code
```

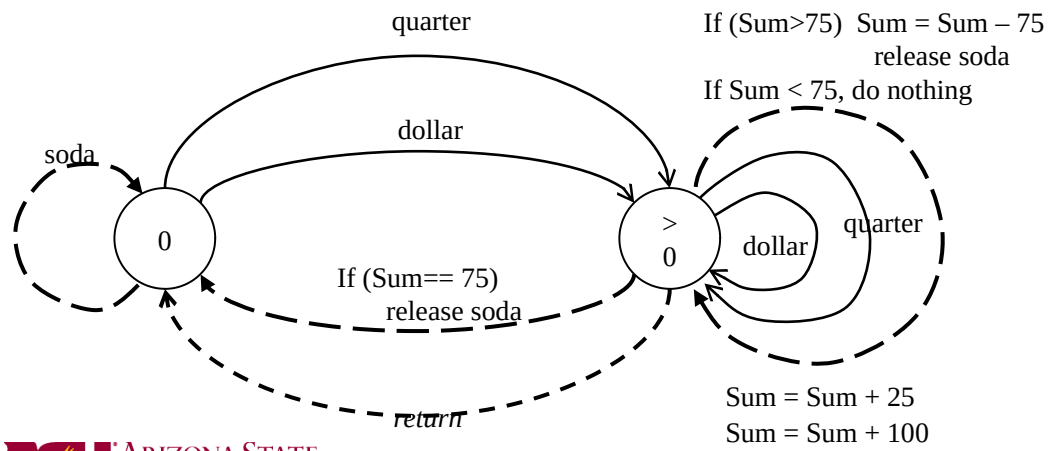
```
Console.WriteLine("Printing from CodeActivity2"); // Thread 2
Double piValue = System.Math.PI;
Console.WriteLine(piValue);
```



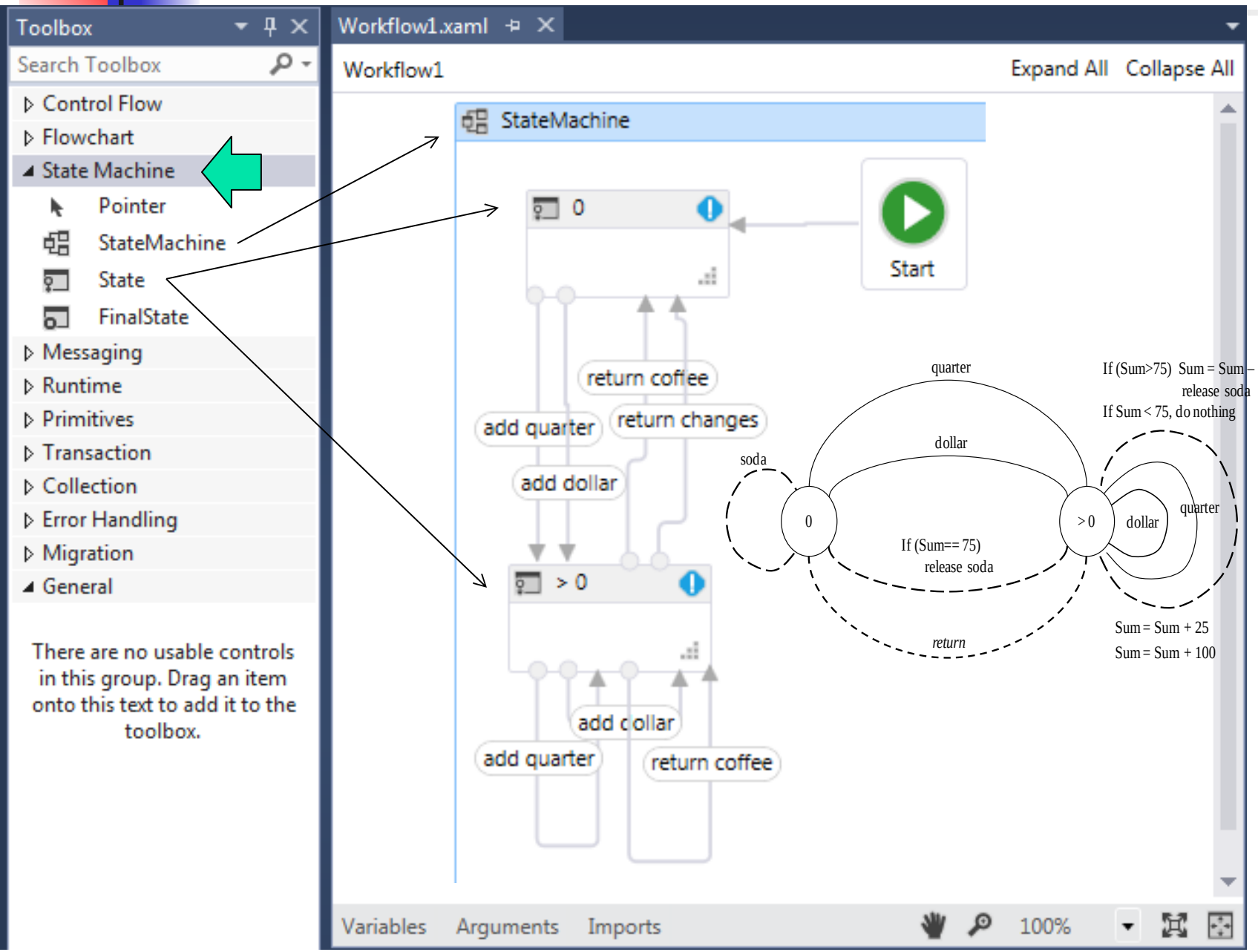
Finite State Machine Model

http://neptune.fulton.ad.asu.edu/WSRepository/CoffeeMachine/





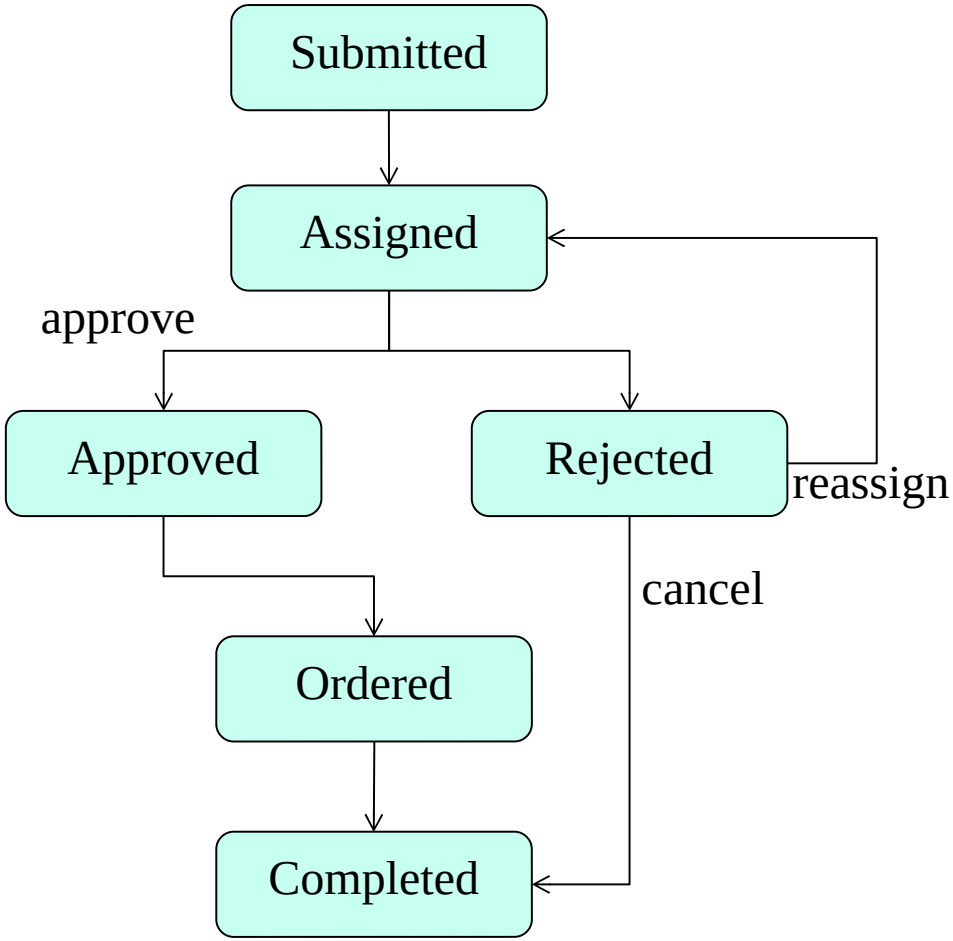
Implementing a State Machine



Event-Driven and State Machine for eBusiness Application

http://msdn.microsoft.com/en-us/magazine/cc163281.aspx

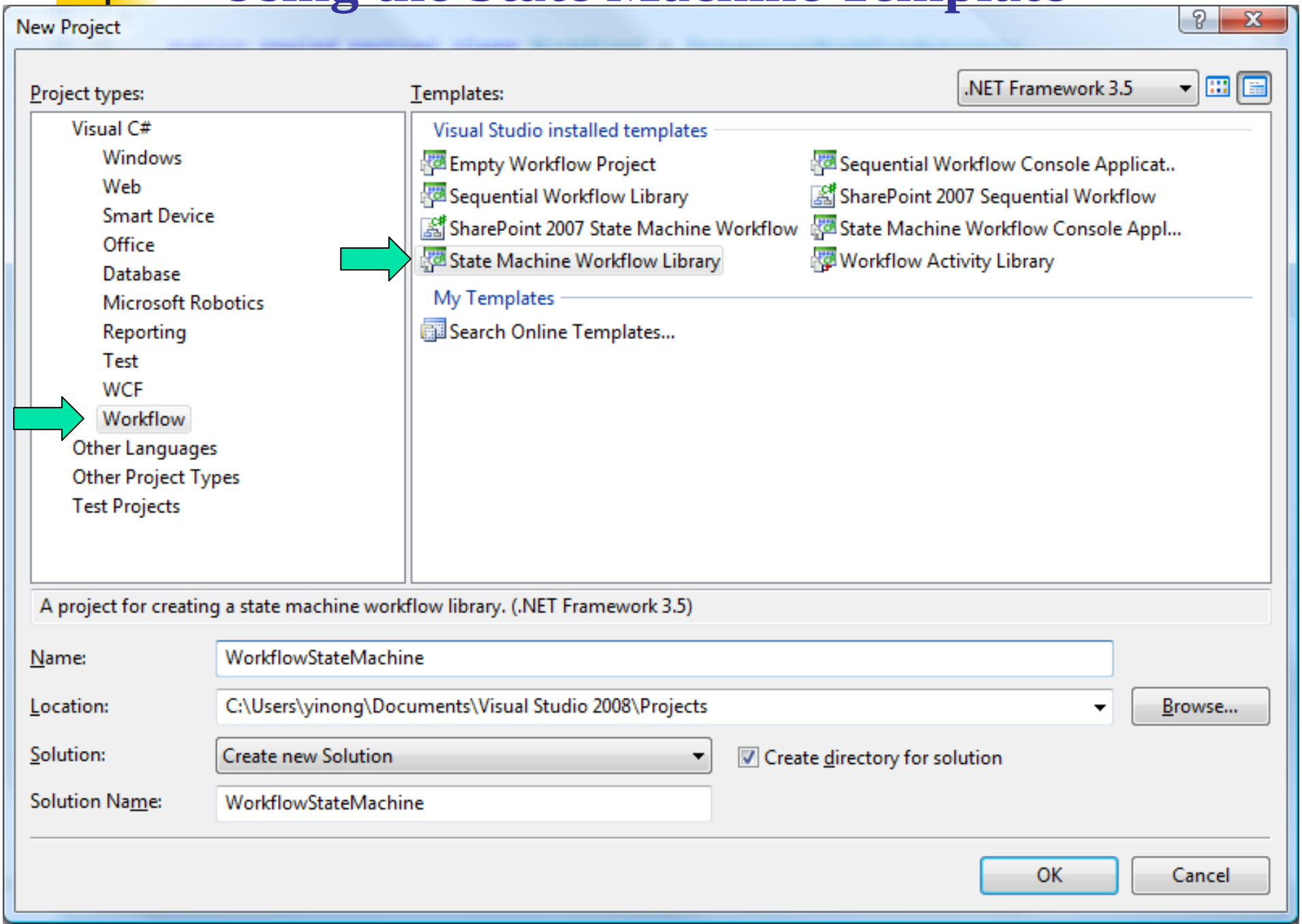
An Ordering Process



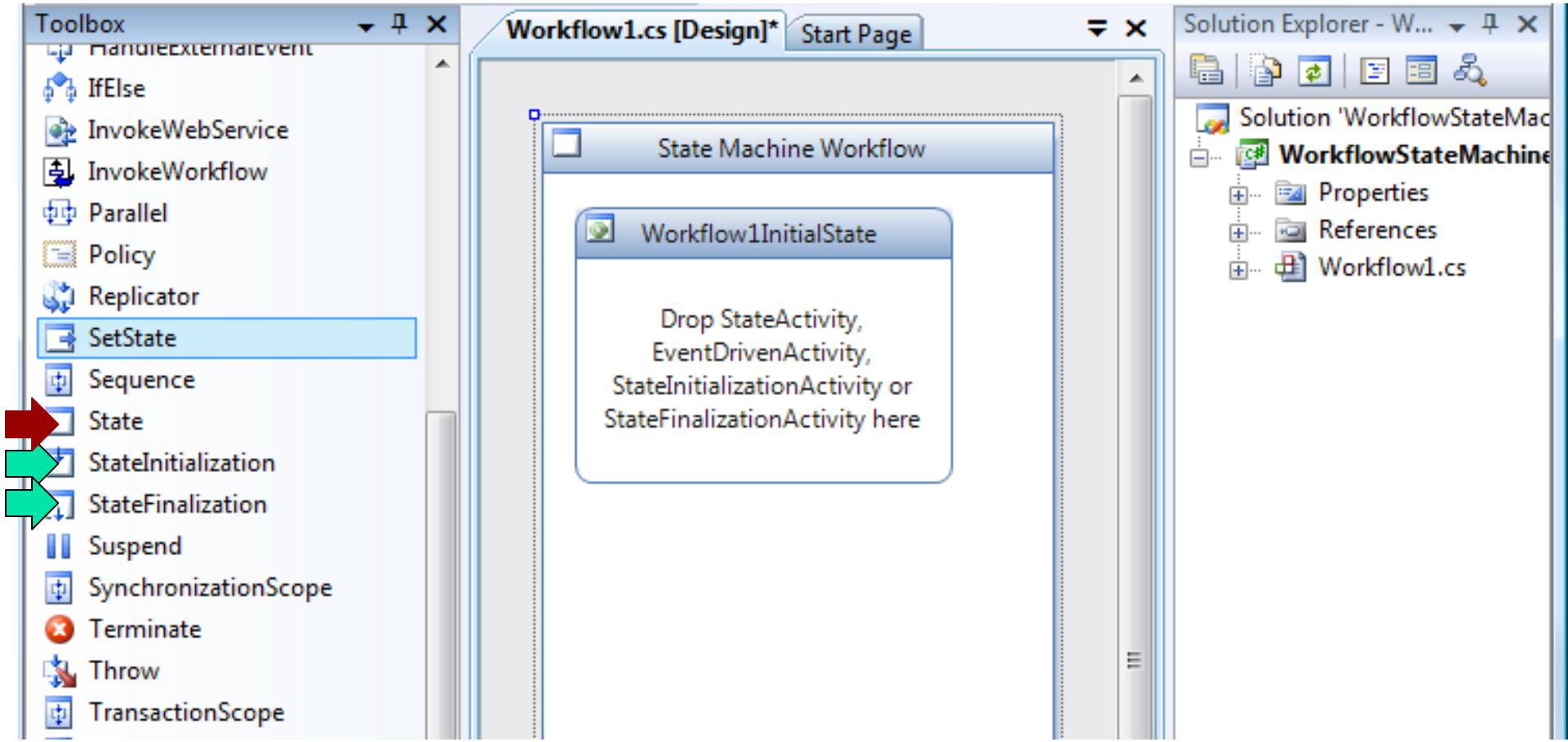
State Machine of Ordering Process in Table View

State	Allowed Transitions
Submitted	Assigned
Assigned	Approved or Rejected
Approved	Ordered
Rejected	Assigned or Completed
Ordered	Completed
Completed	(None)

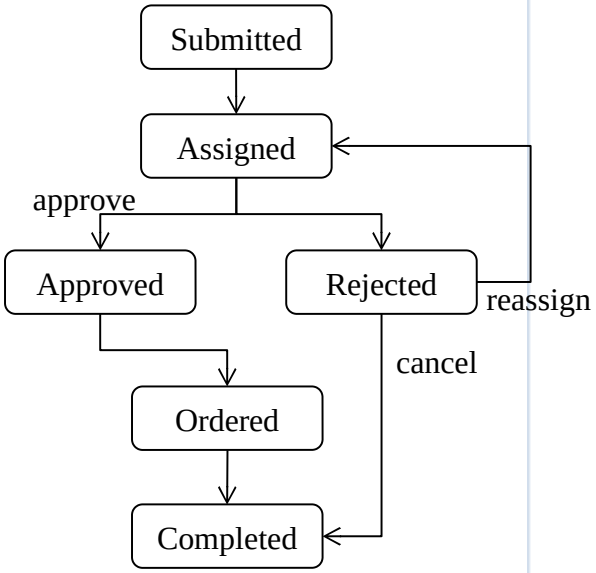
Using the State Machine Template

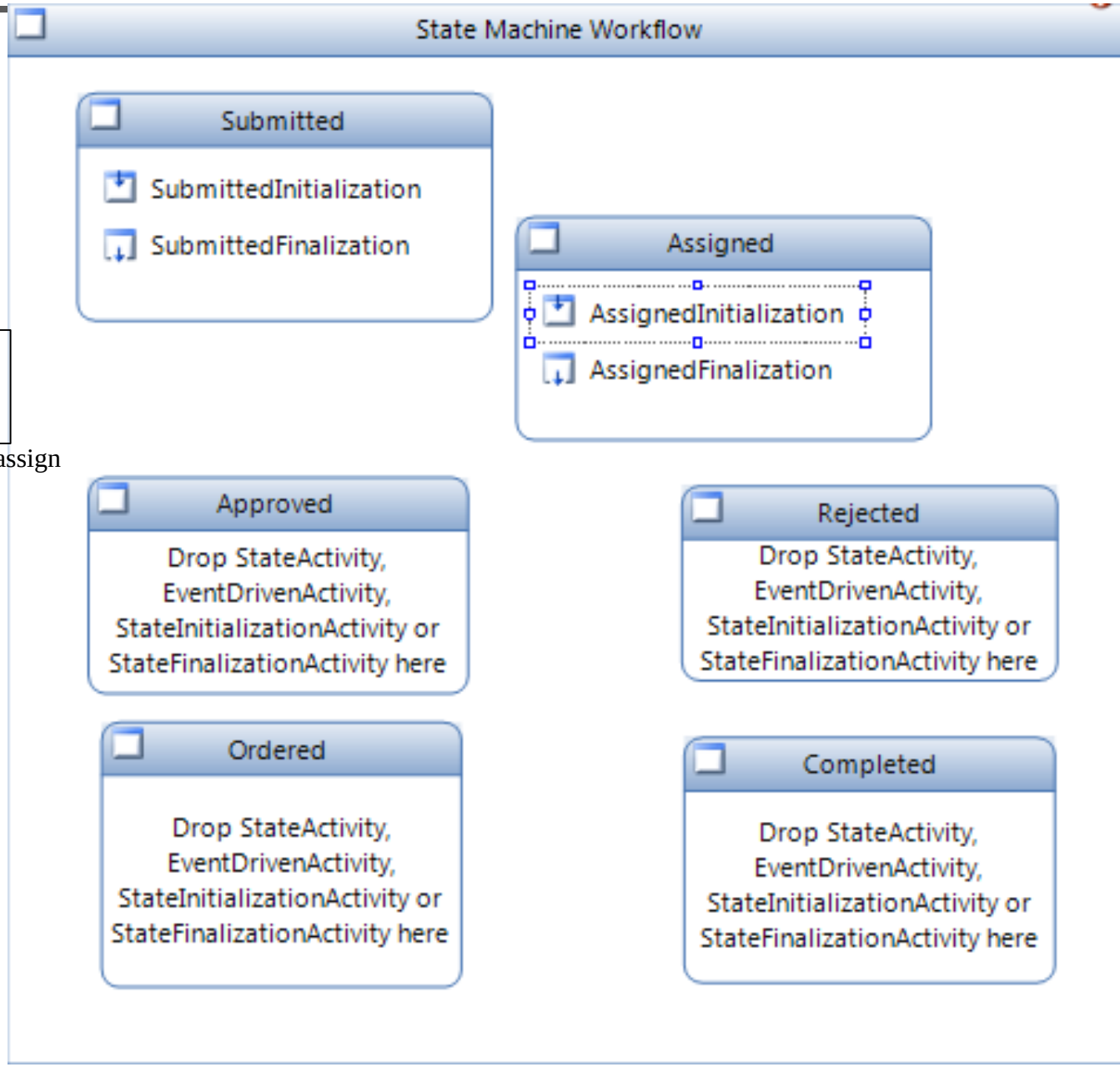


State Machine Template and Services

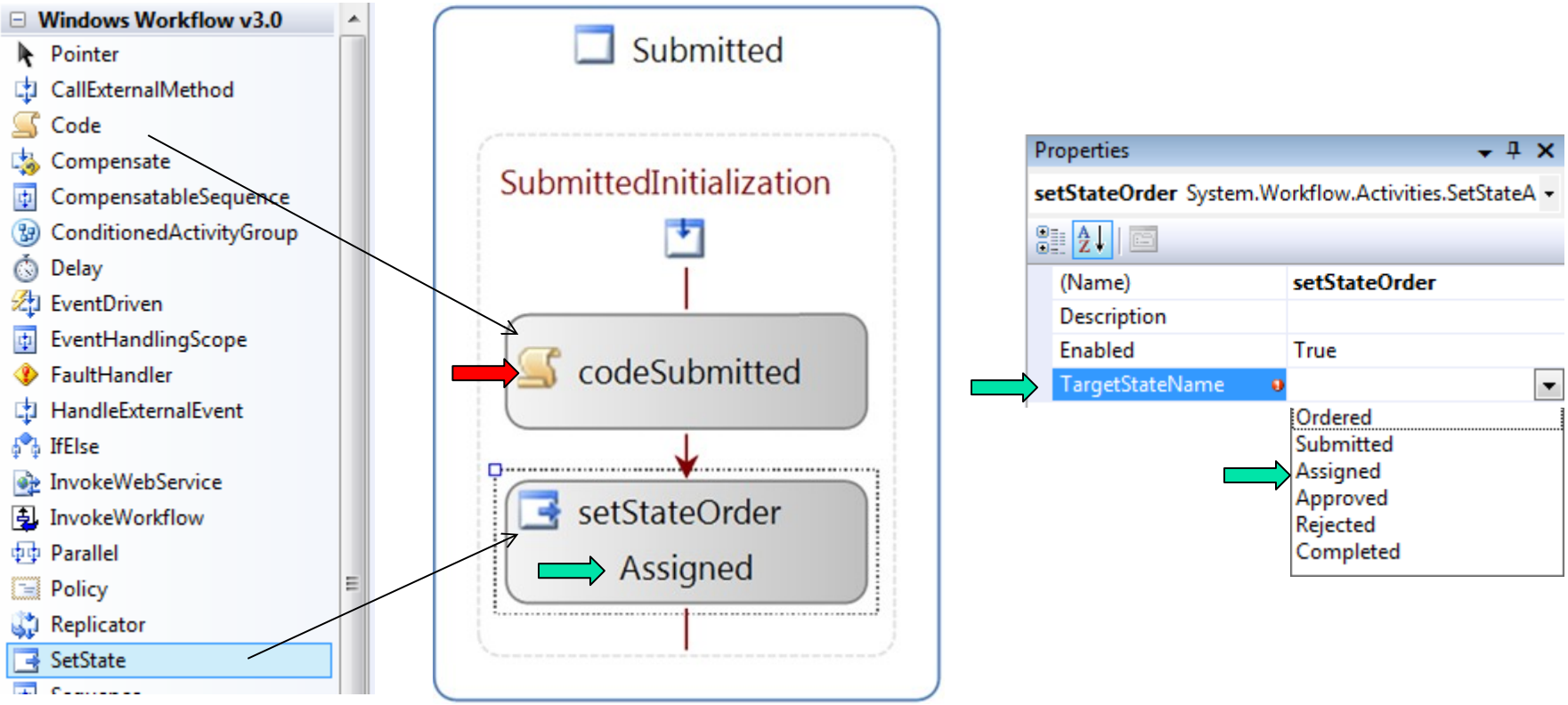


Define States

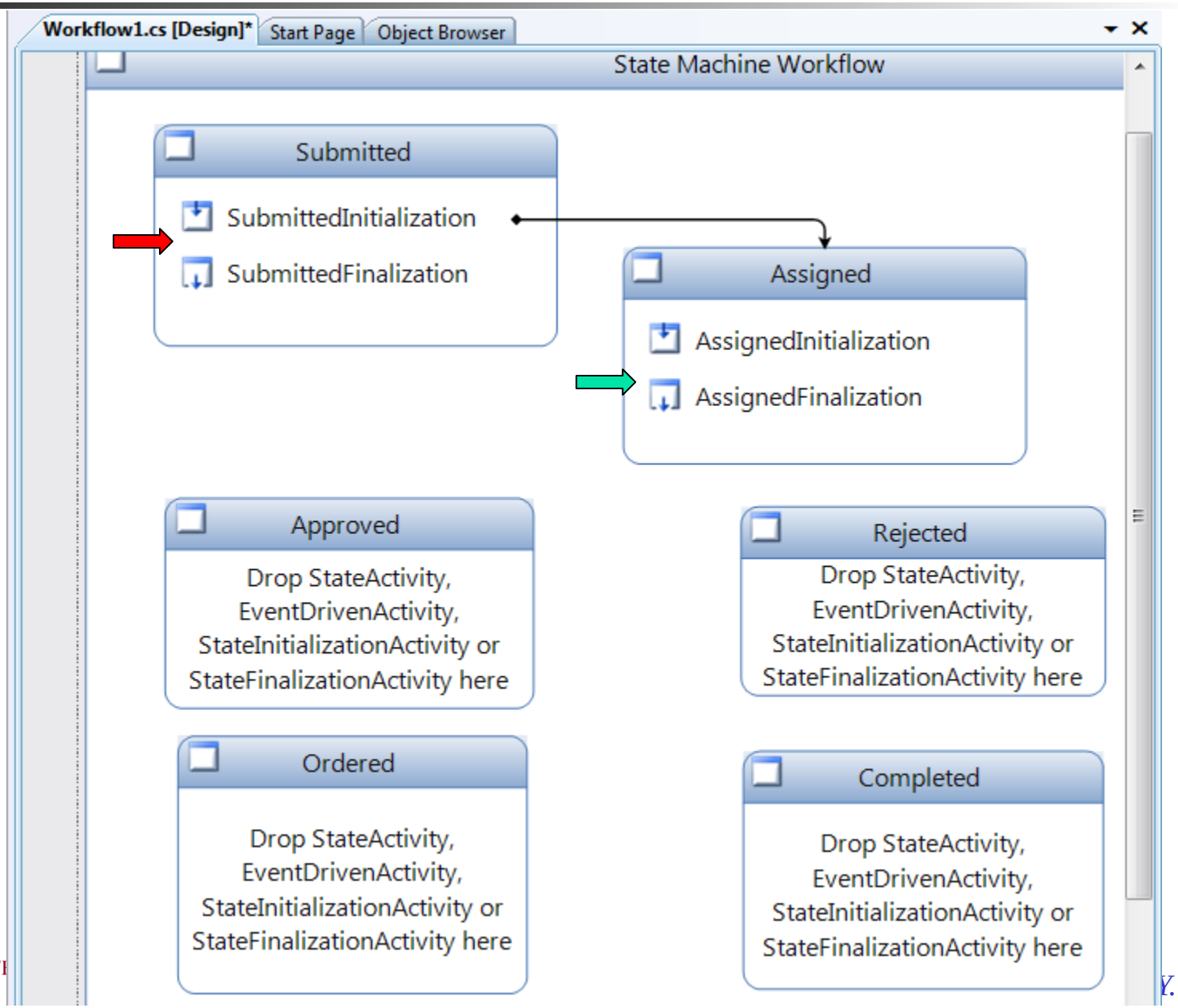




Connecting States Through Target State

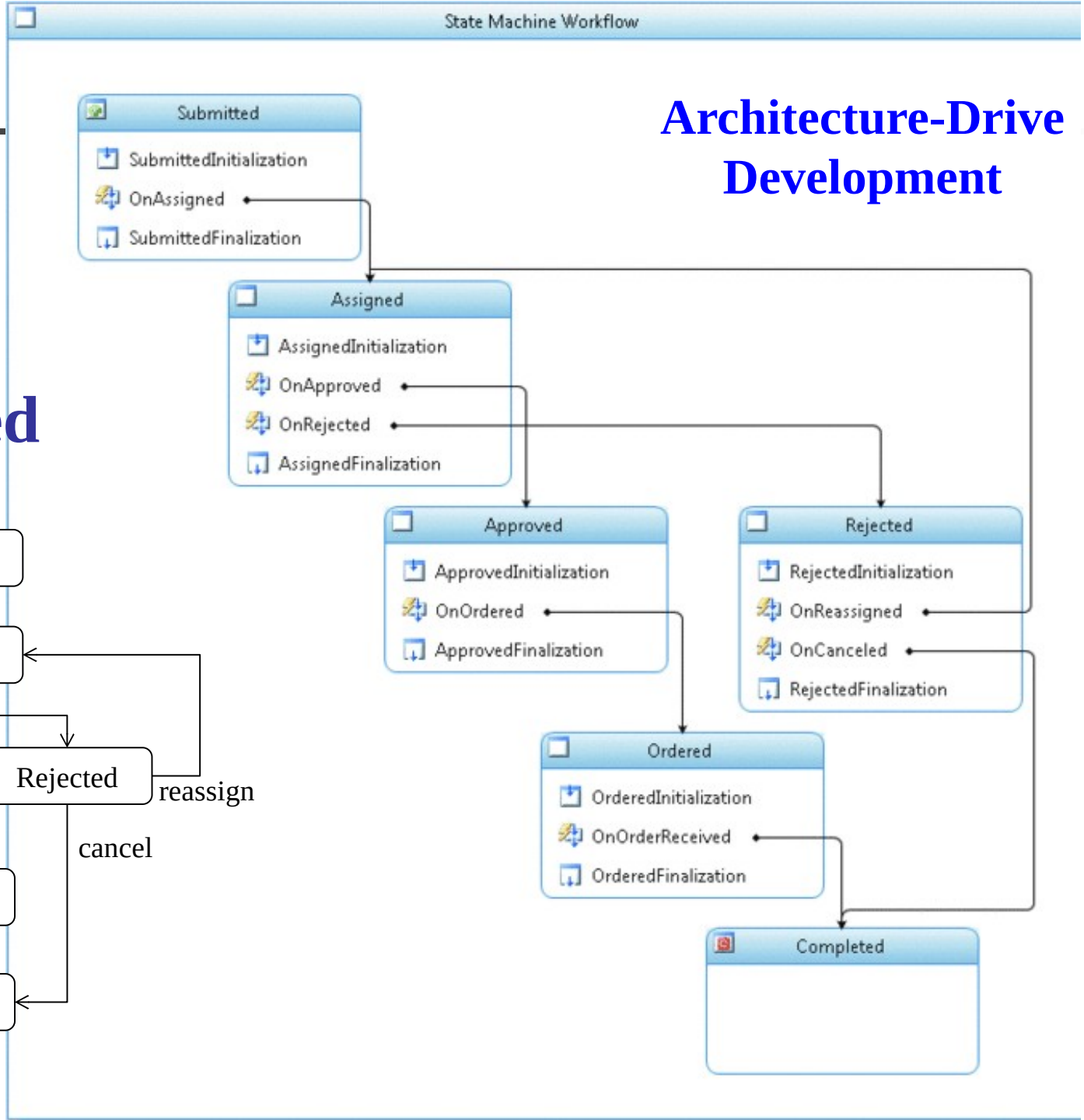


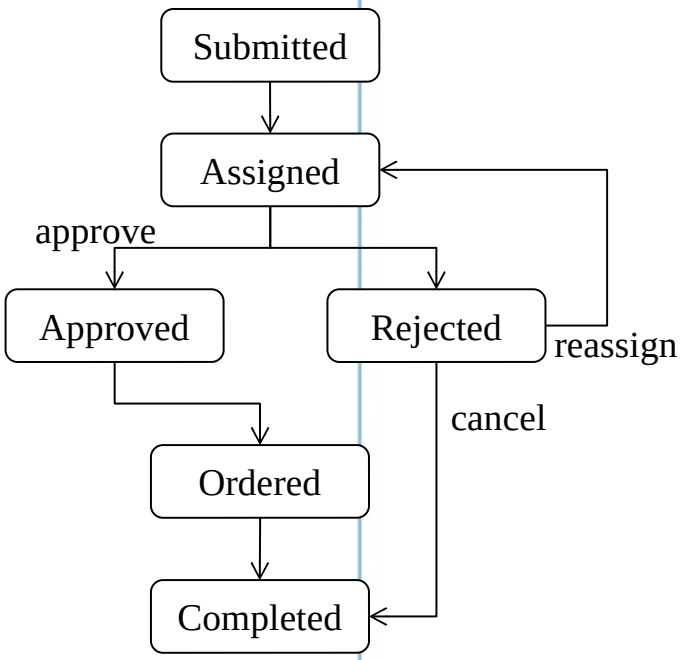
States are Connected



Architecture-Drive Development

All Connected





Steps Implementing the Application Logic

1. Determine the events required for data exchange and state transitions.
2. Define an External Data Exchange interface.
3. Add EventDriven activities to the state machine workflow.
4. Reference the external data exchange interface.
5. Implement the External Data Exchange interface.
6. Add the External Data Exchange to the workflow runtime.
7. Use the External Data Exchange to exchange data and transition states.

Code Behind: Event Interfaces

```
namespace ExternalDataExchange
{
    [ExternalDataExchange()] public interface IEventService
    {
        event EventHandler<SupplyFulfillmentArgs> Assigned;
        event EventHandler<SupplyFulfillmentArgs> Approved;
        event EventHandler<SupplyFulfillmentArgs> Rejected;
        event EventHandler<SupplyFulfillmentArgs> Reassigned;
        event EventHandler<SupplyFulfillmentArgs> Canceled;
        event EventHandler<SupplyFulfillmentArgs> Ordered;
        event EventHandler<SupplyFulfillmentArgs> OrderReceived;
    }
}
```

Implementing the Event Interface

```
public void RaiseAssignedEvent(System.Guid instanceId,
    string assignedTo)
{ // Check to see if event is defined
    if (this.Assigned != null)
    { // Create the EventArgs for this event
        LocalService.SupplyFulfillmentArgs args = new
            LocalService.SupplyFulfillmentArgs(instanceId);
        args.AssignedTo = assignedTo;
        // Raise the event
        this.Assigned(this, args);
    }
}
```

Summary the Lecture

- Key Ideas of Workflow-based Application Development
- WF Constructs and Activities
- Creating WF workflow Application
- Integrating WF and WCF
- Creating WF Services
- Event-Driven Approach and State Machine in WF
- Workflow Supports
  - Architecture-Drive Development
  - Control flow and Dataflow
  - Synchronous and Asynchronous communications