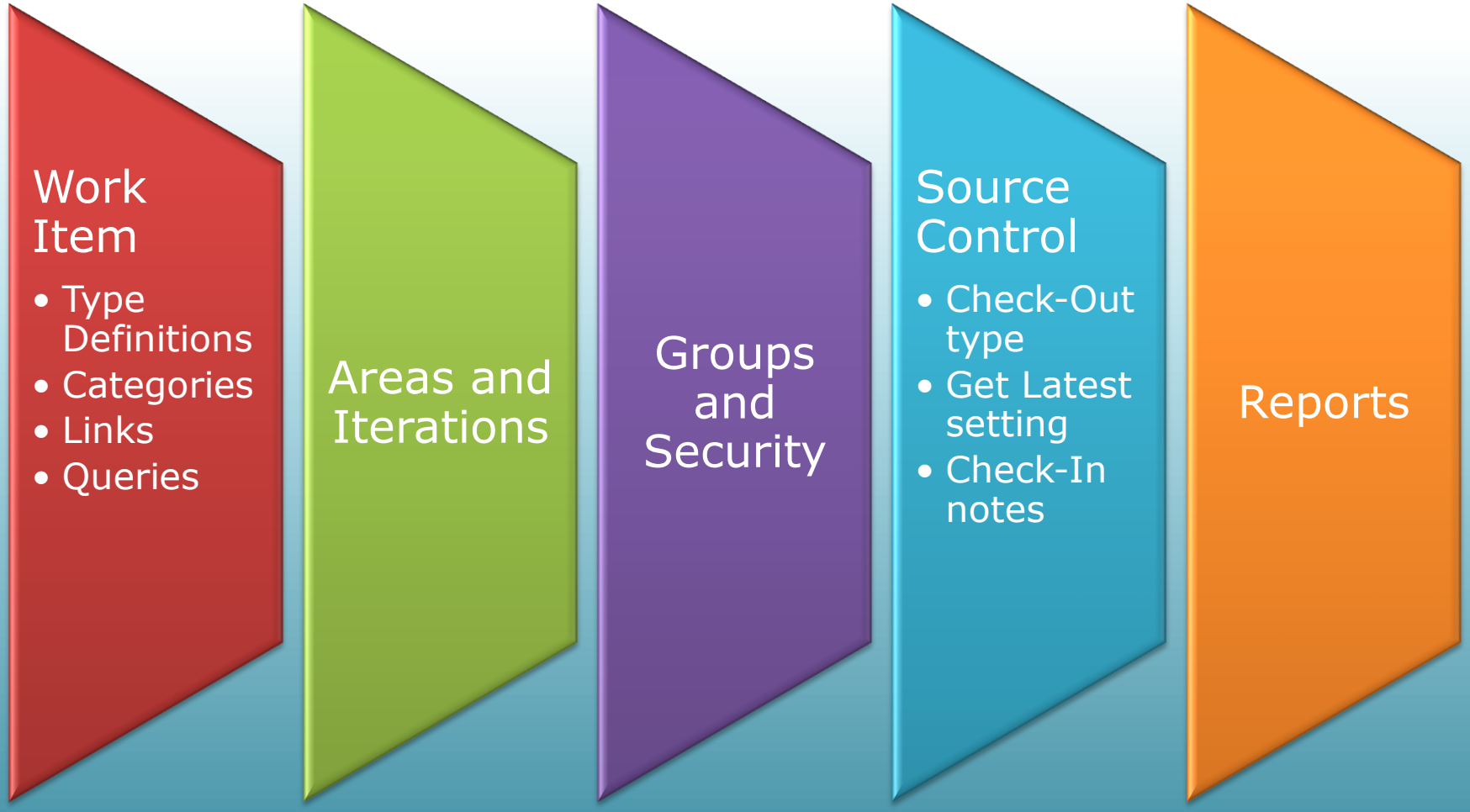
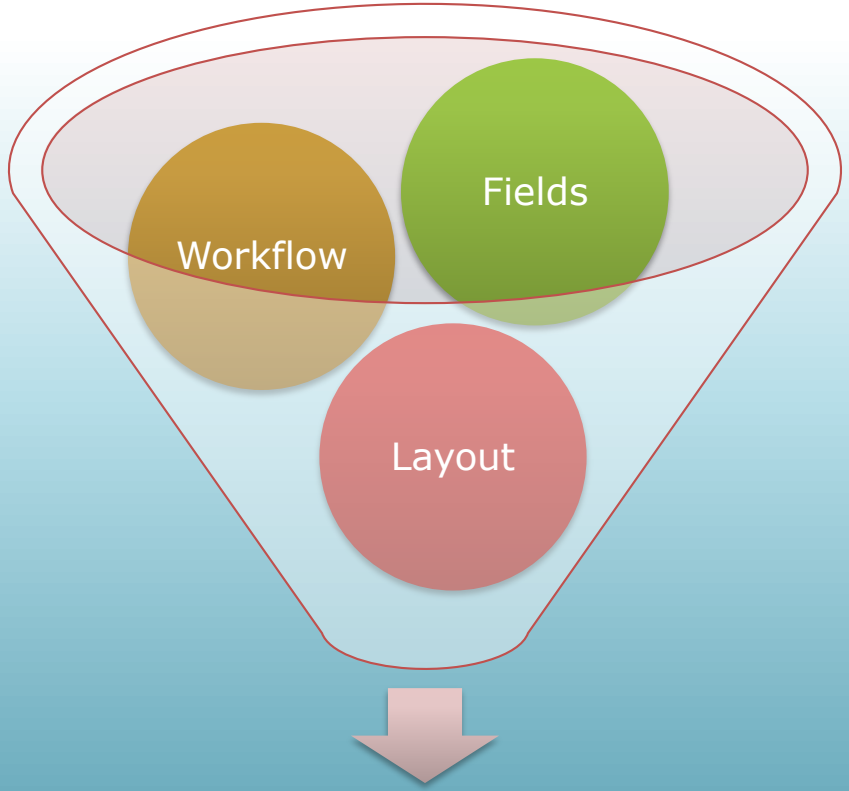
SSGS
IT EDUCON Services Pvt.Ltd.

# Customize Process

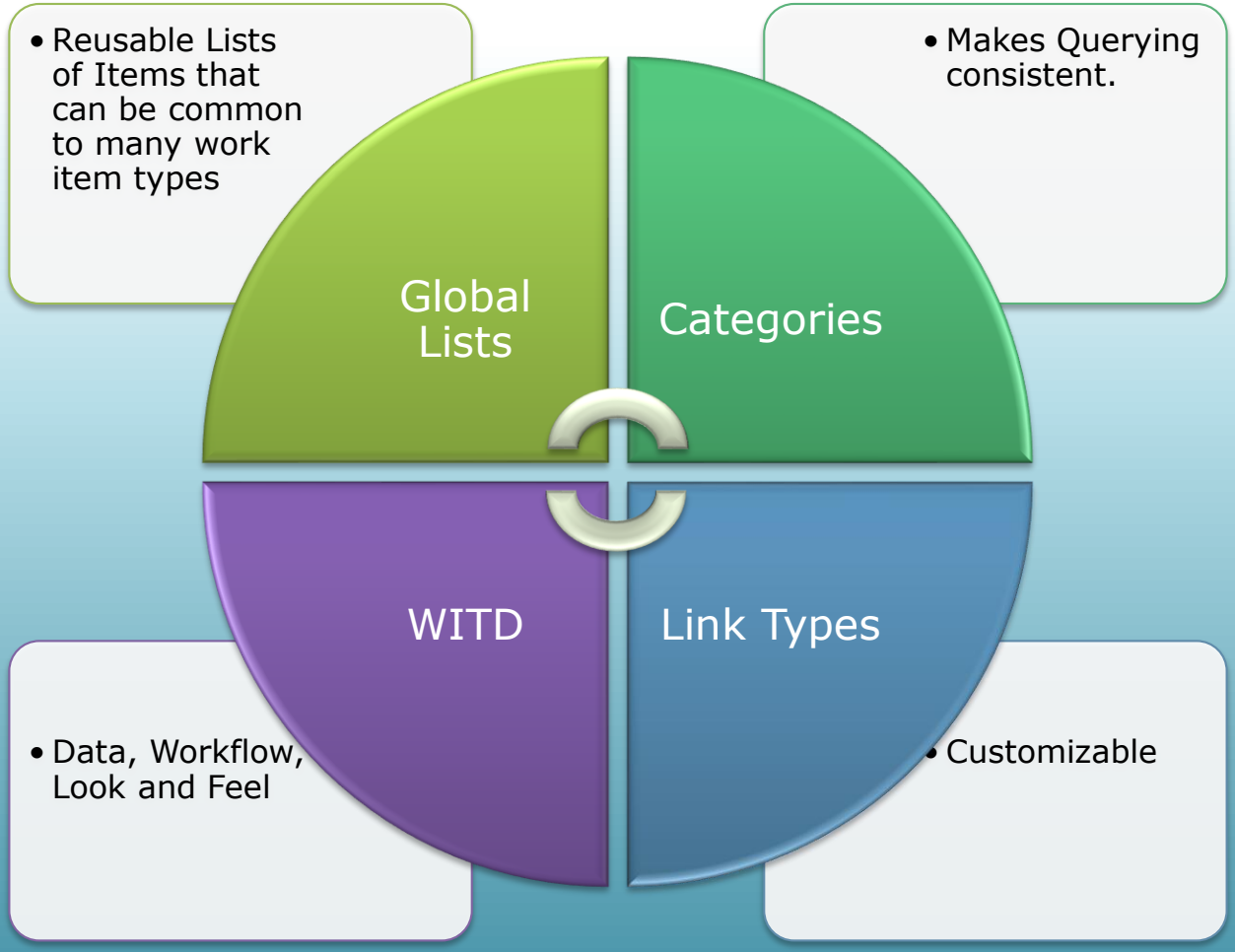
# Process Template



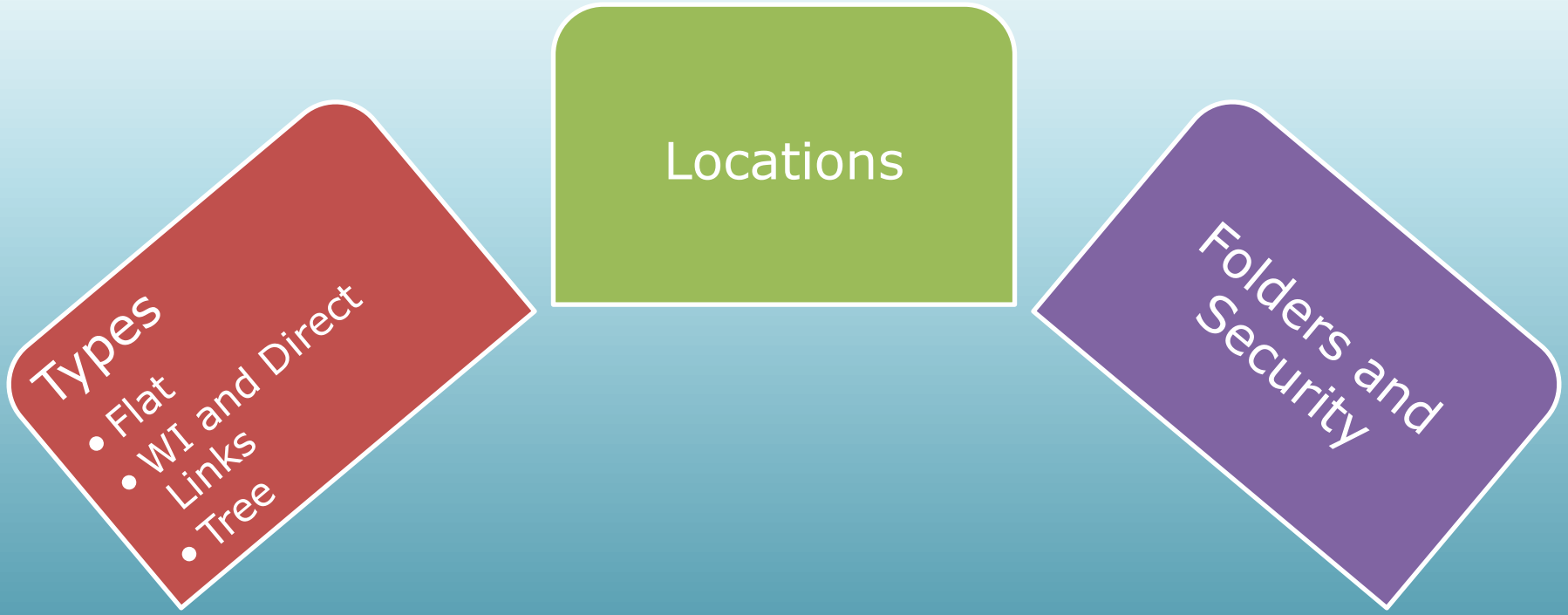
# Work Item Type



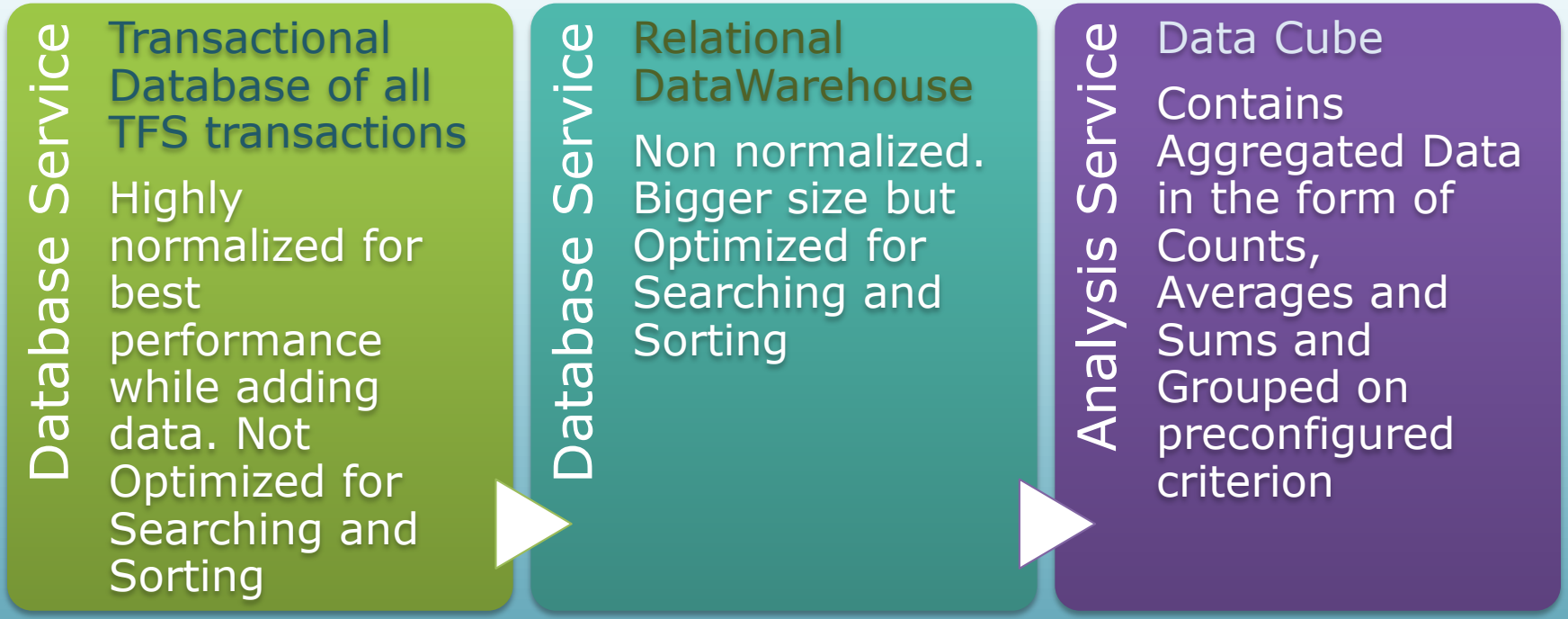
# Work Item Type



# Work Item Queries



# TFS Reporting



# TFS Reporting Security

Default security is sufficient for viewing reports

For user who needs to create report:

- Add user login to read database of the team project collection and TFS DataWarehouse
- Add user to TfsWarehouseReaders role in the analysis service
- Make user as the Content Manager in the Reporting Service

# Custom Reporting

- Create custom report in MS Excel
- Create custom report using Report Builder

# Work Item Template

- Work Item data is stored to be used to fill up frequently used values of fields.
- Reduces the time it takes to create work items

# Lab 9: Customize Process Template

- Install TFS Process Editor add-in to Visual Studio
- Download Agile process template
- In the TFS Process Editor edit a work item type definition
- Save and Upload Agile process template with a different name

# Team Foundation Server REST API

- Representational State Transfer (REST) APIs are service endpoints that support sets of HTTP operations (methods), which provide create, retrieve, update, or delete access to the service's resources.
- Most REST APIs are accessible through our client libraries, which can be used to greatly simplify your client code.

# TFS Authentication in code

- Use PAT (Personal Access Token)
- How to create PAT?

# TFS REST API Request

A REST API request/response pair can be separated into five components:

The request URI, in the following form:

*VERB https://{instance}[/{team-project}]/_apis[/{area}]/{resource}?api-version={version}*

instance: The Azure DevOps Services organization or TFS server you're sending the request to. They are structured as follows:

Azure DevOps Services: dev.azure.com/{organization}

TFS: {server:port}/tfs/{collection} (the default port is 8080, and the value for collection should be DefaultCollection but can be any collection)

resource path: The resource path is as follows: _apis/{area}/{resource}. For example _apis/wit/workitems.

api-version: Every API request should include an api-version to avoid having your app or service break as APIs evolve. api-versions are in the following format: {major}.{minor}[-{stage}[.{resource-version}]], for example:

api-version=1.0

api-version=1.2-preview

api-version=2.0-preview.1

# HTTP request message header fields:

- A required HTTP method (also known as an operation or verb), which tells the service what type of operation you are requesting. Azure REST APIs support GET, HEAD, PUT, POST, and PATCH methods.
- Optional additional header fields, as required by the specified URI and HTTP method. For example, an Authorization header that provides a bearer token containing client authorization information for the request.

# Optional HTTP request message body fields

- Support the URI and HTTP operation. For example, POST operations contain MIME-encoded objects that are passed as complex parameters.
- For POST or PUT operations, the MIME-encoding type for the body should be specified in the Content-type request header as well. Some services require you to use a specific MIME type, such as application/json.

# HTTP response message header fields

- An HTTP status code, ranging from 2xx success codes to 4xx or 5xx error codes. Alternatively, a service-defined status code may be returned, as indicated in the API documentation.
- Optional additional header fields, as required to support the request's response, such as a Content-type response header.

# Optional HTTP response message body fields

- MIME-encoded response objects may be returned in the HTTP response body, such as a response from a GET method that is returning data. Typically, these objects are returned in a structured format such as JSON or XML, as indicated by the Content-type response header.

# Coding using Client Libraries

- For .NET developers building Windows apps and services that integrate with Azure DevOps Services, client libraries are available for integrating with work item tracking, version control, build, and other services are now available.
- **Features**
  - Downloadable from nuget.org and easily importable into your Visual Studio projects
  - Libraries are licensed for redistribution in your apps and services
  - Access both traditional client object model APIs and REST APIs

# Packages and Their Use

| Package Name | Used For |
| --- | --- |
| Microsoft.TeamFoundationServer.ExtendedClient | Existing Windows apps leveraging an older version of the TFS Client OM. |
| Microsoft.TeamFoundationServer.Client | Provides access to version control, work item tracking, build, and more via public REST APIs. |
| Microsoft.VisualStudio.Services.Client | Provides access to shared platform services such as organization, profile, identity, security, and more via public REST APIs. |
| Microsoft.VisualStudio.Services.InteractiveClient | Provides support for interactive sign-in by a user. |
| Microsoft.VisualStudio.Services.DistributedTask.Client | Provides access to the Distributed Task Service via public REST APIs |
| Microsoft.VisualStudio.Services.Release.Client | Provides access to the Release Service via public REST APIs |

# Using TFS REST APIs

- Create a new project in Visual Studio
- Add NuGet package Microsoft.TeamFoundaitonServer.Client and all its dependencies
- Add NuGet package Microsoft.VisualStudio.Services.Interactive Client and its dependencies
- Create code as in next slide to get work item by specifying Id

# Authenticate using REST APIs

- NTLM (Windows Integrated)
- *var visualStudioServicesConnection = new VssConnection(new Uri(baseUri), new VssCredentials());*
- Basic
- var visualStudioServicesConnection = new VssConnection(new Uri(baseUri), *new VssBasicCredential(username, password)*);
- *VssConnection connection = new VssConnection(new Uri(collectionUri), new VssCredentials(new WindowsCredential(new NetworkCredential(username, password))));*
- Using PAT
- var visualStudioServicesConnection = new VssConnection(new Uri(baseUri), *new VssBasicCredential(string.Empty, pat)*);

# Authenticate using REST APIs

- Visual Studio Sign-in prompt
- *var visualStudioServicesConnection = new VssConnection(new Uri(baseUri), new VssClientCredentials());*
- Oauth Token
- *var visualStudioServicesConnection = new VssConnection(new Uri(baseUri), new VssOAuthCredential(accessToken));*
- Azure Active Directory
- var visualStudioServicesConnection = new VssConnection(new Uri(baseUri), *new VssAadCredential(username, password)*);

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.TeamFoundation.WorkItemTracking.WebApi;
using Microsoft.TeamFoundation.WorkItemTracking.WebApi.Models;
using Microsoft.VisualStudio.Services.Client;
using Microsoft.VisualStudio.Services.Common;
using Microsoft.VisualStudio.Services.WebApi;

namespace TFS_REST_API_Client
{
   class Program
   {
      const String c_collectionUri = "https://dev.azure.com/subodhsohoni";
      const String c_projectName = "SLB Migration";
      const String c_repoName = "SLB Migration";
      static void Main(string[] args)
      {
         VssCredentials creds = new VssClientCredentials();
         creds.Storage = new VssClientCredentialStorage();

         // Connect to Azure DevOps Services / TFS
         VssConnection connection = new VssConnection(new Uri(c_collectionUri), creds);
         WorkItemTrackingHttpClient witClient = connection.GetClient<WorkItemTrackingHttpClient>();
         WorkItem wi = witClient.GetWorkItemAsync(int.Parse(Console.ReadLine())).Result;
         Console.WriteLine(wi.Fields["System.Title"]);
      }
   }
}
```

# Other Classes in Client Lib

- GraphHttpClient, GraphGroup, GroupMembership – Users management
- GitHttpClient – To access git repository
- TfvcHttpClient – To access TFVC repository
- More Samples at

https://github.com/Microsoft/vsts-dotnet-samples/tree/master/ClientLibrary/Snippets/Microsoft.TeamServices.Samples.Client

# Lab 9A: Write TFS API code

- Create a console app project
- Add NuGet packages:
  - Microsoft.TeamFoundationServer.Client
  - Microsoft.VisualStudio.Services.InteractiveClient
- Create code as written in earlier slide

# Create Event Handler Using APIs

- Add reference to Microsoft.TeamFoundation.Framework.Server.dll from C:\Program Files\Microsoft Team Foundation Server 14.0\Application Tier\Web Services\bin
- -- for ISubscriber

- aAdd ref to Microsoft.TeamFoundation.WorkItemTracking.Server.dll from same folder -- for WorkItemChangedEvent

- Microsoft.TeamFoundation.WorkItemTracking.Client.dll -- for getting current work item

- Microsoft.TeamFoundation.Client.dll -- for TfsTeamProjectCollection