|  |
| --- |
| **E:\My Documents\Desktop\Logo_FPT_University_doc.jpgMinistry of education and training** |
| Software Project Management Plan |
| Toll Management |
|  |
| |  |  | | --- | --- | | **PSN Team** | | | **Group Members** | |  |  | | --- | --- | | Nguyễn Anh Nhật Linh | 00499 | | Nguyễn Duy Vũ | 00521 | | Trần Hải Đăng | 00357 | | Nguyễn Xuân Hiếu | 00380 | | Tô Văn Quân | 00425 | | | **Supervisor** | Teacher Nguyễn Văn Sang | | **Ext Supervisor** | Huỳnh Anh Dũng,Nguyễn Tất Trung | | **Capstone Project code** | TOM | |
| **Hanoi, 18th May, 2011** |

**Record of Changes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Change Item** | **Description** | **By** | **Version** |
| 18/5/2011 | All | Create the document | LinhNan | 0.1 |
|  |  |  |  |  |
|  |  |  |  |  |

# Table of Contents

[Table of Contents 2](#_Toc293760251)

[I. Problem Definition 4](#_Toc293760252)

[1. Name of this Capstone Project 4](#_Toc293760253)

[2. Problem Abstract 4](#_Toc293760254)

[3. Project Overview 4](#_Toc293760255)

[3.1 The Current System 4](#_Toc293760256)

[3.2 The Proposed System 4](#_Toc293760257)

[3.3 Boundaries of the System 5](#_Toc293760258)

[3.4 Development Environment 5](#_Toc293760259)

[II. Project Organization 6](#_Toc293760260)

[1. Software Process Model 6](#_Toc293760261)

[2. Roles and Responsibilities 7](#_Toc293760262)

[3. Tools and Techniques 7](#_Toc293760263)

[III. Project Management Plan 8](#_Toc293760264)

[1. Tasks 8](#_Toc293760265)

[1.1 Create Software Requirements Specification 8](#_Toc293760266)

[1.2 Design Database 8](#_Toc293760267)

[1.3 Create Software Design Description 9](#_Toc293760268)

[1.4 Layout Design 9](#_Toc293760269)

[1.5 Create Coding Framework 9](#_Toc293760270)

[1.6 Coding 10](#_Toc293760271)

[1.7 System Test 10](#_Toc293760272)

[1.8 Input Initial Data 11](#_Toc293760273)

[1.9 Deployment 11](#_Toc293760274)

[2. Task Sheet: Assignments and Timetable 12](#_Toc293760275)

[3. Meeting Minutes 12](#_Toc293760276)

[3.1 Meeting 6/5/2011 12](#_Toc293760277)

[3.2 Kickoff Meeting 6/5/2011 12](#_Toc293760278)

[3.3 Meeting 7/5/2011 13](#_Toc293760279)

[3.4 Meeting 13/5/2011 14](#_Toc293760280)

[3.5 Meeting 18/5/2011 14](#_Toc293760281)

[IV. Coding Conventions 16](#_Toc293760282)

[1. Naming Conventions 16](#_Toc293760283)

[2. Coding Styles 23](#_Toc293760284)

[3. Comments 28](#_Toc293760285)

[4. Coding Conventions Specific for this Project 28](#_Toc293760286)

[V. References 29](#_Toc293760287)

# Problem Definition

## Name of this Capstone Project

The Capstone project name is Toll Management. It’s a total solution for toll collection that we are ambitious for applying in Viet Nam.

## Problem Abstract

In the Toll management project, the group with five members target to develop a system for managing all steps of a toll collection activity. Almost toll collection stations in Viet Nam using traditional method for toll collection activity: using human’s activity like seeing, checking, handling…. Or some stations applied automatic system for recognizing number plate, but toll collection activity is still depended on human so much, synchronizing timely data about number plate info between local station and Vietnam Register System is impossible. A total solution integrating all activities that overcomes all those disadvantages and is possible to deploy for all toll collection stations in Viet Nam in near future is the propose of this project.

Once completed, the product will archive the following characteristics:

* Recognizing number place of transports exactly, quickly for thoroughly traffic.
* Having a good initial database for managing toll collection: monthly, daily.
* Having various reports for manager review activities of each station.
* Ability to synchronize timely data between local database and Vietnam Register System’s database is the difference.
* Contributing to Viet Nam’s traffic development in general and toll collection in particular.

## Project Overview

### The Current System

Currently, there are some running systems that similar to one of modules (that we researched in report one) that the group’s building. But they are almost independent systems, not one total solution that controls all steps of a toll collection activity. The product is building from a new idea that integrates all modules to a total unique system.

### The Proposed System

The core system will be an application that controls recognizing transports’ number plate. There will be a website that is hosted on a server computer. There will be application that crawl data from Viet Nam Register System.

* The core system will provide the following main features:
* Automatically taking picture of transport’s number plate from an installed camera when the transport moves forward a suitable position.
* Automatically recognizing the text of the number plate in the picture.
* Managing checking the number plate and printing daily ticket basing on database that is place on the server computer.
* The website will provide the following main features in front-end:
* Users responsible access the website and log in the website.
* Authenticated users manage storing toll collection data by the website such as adding, editing… ticket info of a number plate.
* Authenticated users export some reports such as finance, daily, weekly or monthly traffic flow….
* View all client stations info by the website.
* The crawled data application provides the following main features:
* Crawling data from Viet Nam register system using provided web services.
* Synchronizing data with local database.

All modules will use just one database that is placed on a server computer.

### Boundaries of the System

Currently, Viet Nam register system does not provide any web service to get data, so the group will simulate a database that is considered to have synchronized data. And because there are restrictions on capstone project completion time and resources involved, present our initial goals and also the boundaries of the system under development of this Capstone Project will include:

* The competed core application: number plate recognition.
* The completed website.
* All the process documents involved

### Development Environment

Below is the list of hardware and software requirements needed for development environment:

#### Hardware requirements:

* Clients
* Cpu : Intel(R) Pentium(R) 4 2.0 GHz or better supported
* Ram : 512MB RAM (1GB for Windows Vista/ Windows 7)
* Hard Disk: 50GB of free space
* Server :
* Cpu : Intel(R) Core 2 Duo(R) 2.4 GHz or better supported
* Ram : 4GB
* Hard Disk: 100GB of free space

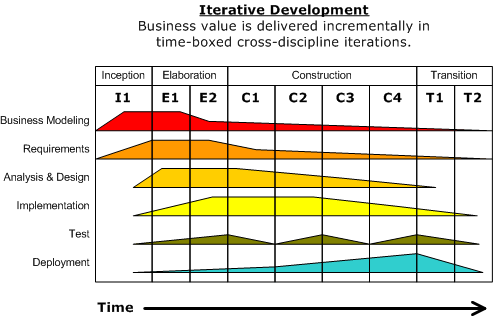
#### Software requirements:

* Operating system: Windows 7/Windows Server 2008
* Web Server: IIS 7
* Framework: .NET Framework 4.0
* IDE: Visual Studio 2010
* DBMS: SQL Server 2008 R2
* Source Control: Google.code

# Project Organization

## Software Process Model

The process model used for developing this project is Rational-Unified-Process model.

[](http://upload.wikimedia.org/wikipedia/commons/0/05/Development-iterative.gif)

Four project life cycle phases of this model:

* Inception:
* Stakeholder on scope definition and cost/schedule estimates that decides success of failure: resources, plan, and milestones…
* At the end of the phase: Establishing a baseline by which to compare actual expenditures versus planned expenditures
* Elaboration:
* Analyze businesses problem.
* A description of the software architecture in a software system development process.
* An executable architecture that realizes architecturally significant use cases.
* Risk list which are revised.
* A development plan for the overall project.
* All requirements definition of the project.
* Construction: The primary objective is to build the software system. In this phase, the main focus is on the development of components and other features of the system. This is the phase when the bulk of the coding takes place. This phase produces the first external release of the software
* Transition: The primary objective is to 'transit' the system from development into production, making it available to and understood by the end user. The activities of this phase include training the end users and maintainers and beta testing the system to validate it against the end users' expectations. The product is also checked against the quality level set in the Inception phase.

## Roles and Responsibilities

|  |  |  |  |
| --- | --- | --- | --- |
| No | **Full name** | **Role in Group** | **Responsibilities** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Tools and Techniques

* Technologies: .NET, MVC 3.0, JQuery.
* Design tool: Photoshop CS5
* UML tool: Enterprise Architect v.
* Development tool: Visual studio 2008.
* Architecture and design patterns: MVC.
* Tortoise SVN.
* Microsoft office 2007.
* Microsoft SQL Server 2008 x86.
* Skype.
* FxCop

# Project Management Plan

## Tasks

### Create Software Requirements Specification

#### Description

Create software requirements specification

#### Output

Software Requirement Specification (SRS) document

#### Deliverables

Deliver SRS document before 14/2/2011

#### Resources Needed

5 people for 1.5 weeks

#### Dependencies and Constraints

None

#### Risks

Because there is no actual user, and the requirements come from all the team members, conflicts may happen regularly.

### Design Database

#### Description

Create logical and physical database design

#### Output

Database Design and Database script

#### Deliverables

Deliver with Software Design Description before 28/2/2010

#### Resources Needed

2 people for 3 days

#### Dependencies and Constraints

Depends on the completion of SRS

#### Risks

SRS may not be detailed enough to capture the business rules, causing the database design to be inappropriate or will be changed much in the future.

### Create Software Design Description

#### Description

Design the system in an OOP manner.

#### Output

Architecture design, detailed design, diagrams and design specification

#### Deliverables

SDD before 28/2/2010

#### Resources Needed

5 people for 1.5 weeks

#### Dependencies and Constraints

Depends on the completion of SRS

#### Risks

Risks may include choosing inappropriate architecture and design patterns, causing the system hard to maintain, or causing high coding efforts

### Layout Design

#### Description

Create the main GUI for layout, and create global styles that will be applied to the GUI

#### Output

HTML layout and CSS files

#### Deliverables

None

#### Resources Needed

1 person for 1 week

#### Dependencies and Constraints

None

#### Risks

None

### Create Coding Framework

#### Description

Map the architecture design into source code, create the project solution files and common classes, and implement common functions.

#### Output

Visual Studio project and solution files containing coding framework

#### Deliverables

None

#### Resources Needed

1 person for 1 week

#### Dependencies and Constraints

Dependent on Architecture design and SRS

#### Risks

All the technologies used in this project are so new that the team leader may not master before creating coding framework. This leads to inappropriate implementation and hard to train other members.

### Coding

#### Description

Implement the system to reflect the requirements

#### Output

Source code of the project, unit test reports

#### Deliverables

Executable programs and source code before 20/4/2011

#### Resources Needed

5 people for 5 weeks

#### Dependencies and Constraints

Depends on the completion of SRS, SDD, Database design, Coding framework

#### Risks

* Unit test may not be performed thoroughly causing spending many efforts in system test phase.
* May lack of time to implement all the requirements
* The team don’t have a common place, so it is hard to perform collaboration and CM
* Team members may not easily get used to new technologies

### System Test

#### Description

Perform system test for the system

#### Output

System test report

#### Deliverables

Software Test Documentation before 20/4/2011

#### Resources Needed

5 people for 1.5 weeks

#### Dependencies and Constraints

Coding is finished

#### Risks

* Lack of professional testers in team
* Developers are also responsible for system testing, this may lead to compromise

### Input Initial Data

#### Description

Collect and input initial data for the system before going live

#### Output

Initial data

#### Deliverables

None

#### Resources Needed

5 people for 1 week

#### Dependencies and Constraints

Coding and system test are finished

#### Risks

None

### Deployment

#### Description

Deploy the system to the Internet

#### Output

Running website with domain and hosting

#### Deliverables

None

#### Resources Needed

2 people for 3 days

#### Dependencies and Constraints

Coding and system test are finished, initial data is inputted

#### Risks

None

## Task Sheet: Assignments and Timetable

Refer to the Master Plan document.

## Meeting Minutes

### Meeting 6/5/2011

**Meeting Minutes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | Requirement Brainstorming | **Date** | 13/05/2011 |
| **Facilitator** | Detech Building | **Time** | 15:00 – 16:00 |
| **Location** | 8 - Ton That Thuyet Street, Cau Giay District | **Scribe** | DangTH |
| **Attendees** | SangNV (Supervisor), ), LinhNAN (Team Lead), VuND, QuanTV, HieuNX, DangTH | | |
| **Absent** |  | | |

| **Key Points Discussed** | | |
| --- | --- | --- |
| No. | Topic | Highlights |
| 1 | Review idea | Discuss about what we have to do in this project |
| 2 | Process | Discuss about what process team must follow |

| **Action Plan** | | | |
| --- | --- | --- | --- |
| No. | Action Item(s) | Owner | Target Date |
| 1 | Research about Technical and Business | All members | 13/5 |

### Kickoff Meeting 6/5/2011

**Meeting Minutes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | Kickoff Meeting | **Date** | 06/05/2011 |
| **Facilitator** | Duck Grill Restaurant | **Time** | 18:00 – 21:00 |
| **Location** | Cau Giay Str. | **Scribe** | VuND |
| **Attendees** | DungHA (Supervisor), LinhNAN (Team Lead), VuND, QuanTV, HieuNX, DangTH | | |
| **Absent** |  | | |

| **Key Points Discussed** | | |
| --- | --- | --- |
| No. | Topic | Highlights |
| 1. | Team member introduction | Introduce all members of team to supervisor |
| 2. | Project introduction | Introduce briefly about the project |
| 3. | Kick-off Party | Make all team member closer |

| **Action Plan** | | | |
| --- | --- | --- | --- |
| No. | Action Item(s) | Owner | Target Date |
| 1. | Writing Report 1: Introduction | VuND + LinhNAN + QuanTV | 8/5 |

### Meeting 7/5/2011

**Meeting Minutes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | Requirement Brainstorming | **Date** | 07/05/2011 |
| **Facilitator** | Detech Building | **Time** | 9:00 – 11:00 |
| **Location** | 8 - Ton That Thuyet Street, Cau Giay District | **Scribe** | VuND |
| **Attendees** | TrungTT (Supervisor), DungHA(Supervisor), LinhNAN (Team Lead), VuND, QuanTV, HieuNX, DangTH | | |
| **Absent** |  | | |

| **Key Points Discussed** | | |
| --- | --- | --- |
| No. | Topic | Highlights |
| 1 | How to make a good thesis | Discuss about what thesis is and the different between thesis and report |
| 2 | Report 1 | Discuss about what report 1 must have |

| **Action Plan** | | | |
| --- | --- | --- | --- |
| No. | Action Item(s) | Owner | Target Date |
| 1 | Writing report 1 | VuND , LinhNAN, Quan TV | 8/5 |

### Meeting 13/5/2011

**Meeting Minutes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | Requirement Brainstorming | **Date** | 13/05/2011 |
| **Facilitator** | Detech Building | **Time** | 18:00 – 19:00 |
| **Location** | 8 - Ton That Thuyet Street, Cau Giay District | **Scribe** | DangTH |
| **Attendees** | SangNV (Supervisor), ), LinhNAN (Team Lead), VuND, QuanTV, HieuNX, DangTH | | |
| **Absent** |  | | |

| **Key Points Discussed** | | |
| --- | --- | --- |
| No. | Topic | Highlights |
| 1 | Toll Management information | Discuss about what kind of information a Toll Management should have |
| 2 | System Functions | Discuss about what function the system should have |
| 3 | Process | Unify the process team must follow |

| **Action Plan** | | | |
| --- | --- | --- | --- |
| No. | Action Item(s) | Owner | Target Date |
| 1 | Research about Technical and Business | All members | 20/5 |

### Meeting 18/5/2011

**Meeting Minutes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | Requirement Brainstorming | **Date** | 18/05/2011 |
| **Facilitator** | Detech Building | **Time** | 17:30 – 18:30 |
| **Location** | 8 - Ton That Thuyet Street, Cau Giay District | **Scribe** | VuND |
| **Attendees** | DungHA (Supervisor), TrungTT (Supervisor), LinhNAN (Team Lead), VuND, QuanTV, HieuNX, DangTH | | |
| **Absent** |  | | |

| **Key Points Discussed** | | |
| --- | --- | --- |
| No. | Topic | Highlights |
| 1 | Review and Comment Report 1 | Discuss about what report 1 lacks and comment for doing report 2 better |
| 2 | Report 2 Orientation | Discuss about what Report 2 should have |

| **Action Plan** | | | |
| --- | --- | --- | --- |
| No. | Action Item(s) | Owner | Target Date |
| 1 | Writing report 2 | VuND , LinhNAN, Quan TV | 21/5 |

# Coding Conventions

The following rules follow the standard rules for developing applications using .NET and C#.

## Naming Conventions

The following terms describe different ways to case identifiers.

#### Pascal Casing

The first letter in the identifier and the first letter of each subsequent concatenated word are capitalized. You can use Pascal case for identifiers of three or more characters. For example:

BackColor

#### Camel Casing

The first letter of an identifier is lowercase and the first letter of each subsequent concatenated word is capitalized. For example:

backColor

#### Uppercase

All letters in the identifier are capitalized. For example:

System.IO;

System.Web.UI;

You might also have to capitalize identifiers to maintain compatibility with existing, unmanaged symbol schemes, where all uppercase characters are often used for enumerations and constant values. In general, these symbols should not be visible outside of the assembly that uses them.

The following table summarizes the capitalization rules and provides examples for the different types of identifiers.

| Identifier | Case | Example |
| --- | --- | --- |
| Class | Pascal | AppDomain |
| Enum Type | Pascal | ErrorLevel |
| Enum Value | Pascal | FatalError |
| Event | Pascal | ValueChange |
| Exception class | Pascal | WebException  Note: Always ends with the suffix Exception |
| Read-only Static field | Pascal | RedValue |
| Interface | Pascal | IDisposable  Note: Always begins with the prefix I |
| Method | Pascal | ToString |
| Namespace | Pascal | System.Drawing |
| Parameter | Camel | typeName |
| Property | Pascal | BackColor |
| Protected instance field | Camel | redValue  Note: Rarely used. A property is preferable to using a protected instance field |
| Public instance field | Pascal | RedValue  Note: Rarely used. A property is preferable to using a public instance field |

#### Naming rules

* Use Pascal casing for Class names.

The following are examples of correctly named classes:

|  |
| --- |
| public class **FileStream**  public class **Button**  public class **String** |

* Use Pascal casing for Method names

void SayHello(string name)

{

...

}

* Use Camel casing for variables and method parameters

void SayHello(string name)

{

string fullMessage = "Hello " + name;

...

}

* All private member variables must be prefixed with underscore (\_) so that they can be identified from other local variables.

public class HelloWorld

{

private int \_count;

}

* Use Meaningful, descriptive words to name variables. Do not use abbreviations.

string address (not addr)

int salary (not sal)

* Do not use variable names that resemble keywords. The following code example illustrates correct naming standard for primitive types.

|  |  |  |  |
| --- | --- | --- | --- |
| C# Style name | Common use style name | Prefix | Example |
| sbyte | SByte | sbyt | sbytSalary |
| byte | Byte | byt | bytRasterData |
| short | Int16 | sht | shtAge |
| ushort | UInt16 | usht | ushtAccount |
| int | Int32 | int | intQuantity |
| uint | UInt32 | uint | uintEmployeeID |
| long | Int64 | lng | lngQuantity |
| ulong | UInt64 | ulng | ulngDistance |
| float | Single | flt | fltTotal |
| double | Double | dbl | dblTolerance |
| decimal | Decimal | dec | decAmountOfMoney |
| bool | Boolean | bln | blnFound |
| datetime | DateTime | dtm | dtmStart |
| char | Char | chr | chrFirstLetter |
| string | String | str | strFName |
| object | Object | obj | objCurrent |

* Prefix boolean variables, properties and methods with “is” or similar prefixes.

private bool \_isFinished;

* Namespace names should follow the standard pattern.

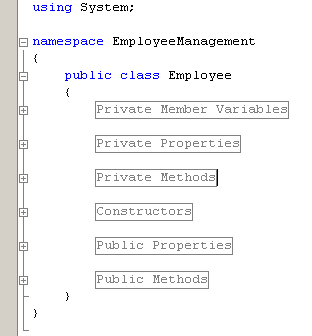
***<company name>.<product name>.<top level module>.<bottom level module>***

* Control Naming Standard

| Control type | Prefix | Example |
| --- | --- | --- |
| 3D Panel | pnl | pnlGroup |
| ADO Data | ado | adoBiblio |
| Animated button | ani | aniMailBox |
| Check box | chk | chkReadOnly |
| Combo box, drop-down list box | cbo | cboEnglish |
| Command button | btn | btnExit |
| Common dialog | dlg | dlgFileOpen |
| Communications | com | comFax |
| Control | ctr | ctrCurrent |
| Data | dat | datBiblio |
| Data-bound combo box | cbo | cboLanguage |
| Data-bound grid | grd | grdQueryResult |
| Data-bound list box | lst | lstJobType |
| Data repeater | drp | drpLocation |
| Date picker | dtp | dtpPublished |
| Directory list box | dir | dirSource |
| Drive list box | drv | drvTarget |
| File list box | fil | filSource |
| Flat scroll bar | fsb | fsbMove |
| Form | frm | frmEntry |
| Frame | fra | fraLanguage |
| Gauge | gau | gauStatus |
| Graph | gra | graRevenue |
| Grid | grd | grdPrices |
| Hierarchical flex grid | flex | flexOrders |
| Horizontal scroll bar | hsb | hsbVolume |
| Image | img | imgIcon |
| Image combo | Imgcbo | imgcboProduct |
| ImageList | ils | ilsAllIcons |
| Label | lbl | lblHelpMessage |
| Line | lin | linVertical |
| List box | lst | lstPolicyCodes |
| ListView | lvw | lvwHeadings |
| Menu | mnu | mnuFileOpen |
| Month view | mvw | mvwPeriod |
| MS Chart | ch | chSalesbyRegion |
| MS Flex grid | msg | msgClients |
| MS Tab | mst | mstFirst |
| OLE container | ole | oleWorksheet |
| Option button | opt | optGender |
| Picture box | pic | picVGA |
| Picture clip | clp | clpToolbar |
| ProgressBar | prg | prgLoadFile |
| Remote Data | rd | rdTitles |
| RichTextBox | rtf | rtfReport |
| Shape | shp | shpCircle |
| Slider | sld | sldScale |
| Spin | spn | spnPages |
| StatusBar | sta | staDateTime |
| SysInfo | sys | sysMonitor |
| TabStrip | tab | tabOptions |
| Text box | txt | txtLastName |
| Timer | tmr | tmrAlarm |
| Toolbar | tlb | tlbActions |
| TreeView | tre | treOrganization |
| UpDown | upd | updDirection |
| Vertical scroll bar | vsb | vsbRate |

* File name should match with class name.

***For the class*** HelloWorld***, the file name should be HelloWorld.cs***

* Use #region to group related pieces of code together. If you use proper grouping using #region, the page should like this when all definitions are collapsed.
* Keep private member variables, properties and methods in the top of the file and public members in the bottom.
* Compound words should be cased correctly (FxCop Microsoft.Naming CA1702)

Avoid creating compound words from terms which exist in the dictionary as discrete terms. Do not create a compound word such as ***'StopWatch'*** or ***'PopUp'***. These terms are recognized in the dictionary and should be cased as ***'Stopwatch'*** and ***'Popup'.***

* Identifiers should differ by more than case. (FxCop Microsoft.Naming CA1708)

Do not use names that require case sensitivity for uniqueness. Components must be fully usable from both case-sensitive and case-insensitive languages. Since case-insensitive languages cannot distinguish between two names within the same context that differ only by case, components must avoid this situation.

* Identifiers should have correct prefix (FxCop Microsoft.Naming CA1715)

Prefix interface names with the letter 'I' to indicate that the type is an interface, as in IFormattable. Prefix generic type parameter names with the letter 'T' and provide them with descriptive names, as in Dictionary<TKey, TValue>, unless a single 'T' is completely self-explanatory, as in Collection<T>. Use Pascal casing for both interface and type parameter names. Use abbreviations sparingly. Do not use the underscore character. Do use similar names when defining a class/interface pair where the class is a standard implementation of the interface. The names should differ only by the letter I prefixed on the interface name, as with Component and IComponent.

* Identifiers should have correct suffix (FxCop Microsoft.Naming CA1710)

Types that extend certain base types have specified name suffixes. Types that extend Attribute, for example, should be suffixed in 'Attribute', as in ObsoleteAttribute. This rules checks types that extend several base types, including Attribute, Exception, EventArgs, IMembershipPermission, Stream, and others.

* Identifiers should not match keywords (FxCop Microsoft.Naming CA1716)

Identifiers which conflict with reserved language keywords should be avoided. Using a reserved keyword as an identifier makes it harder for consumers in other languages to use your API.

* Parameter names should not match member names (FxCop Microsoft.Naming CA1719)

Parameter names should be distinct from member names.

public class HelloWorld

{

private int \_count;

public HelloWorld(int count)

{

\_count = count; //Should not change the parameter name to \_count

}

}

* Property names should not match get methods (FxCop Microsoft.Naming CA1721)

A Get method was found with the same name as a property. Get methods and properties should have names that clearly distinguish their function. See the design guidelines for information regarding choosing properties over methods.

* Type names should not match namespaces (FxCop Microsoft.Naming CA1724)

Identifiers which conflict in whole or in part with namespace names should be avoided. Names that describe the purpose or contents of a type are preferred.

## Coding Styles

* Avoid writing very long methods. A method should typically have 1~50 lines of code. If a method has more than 50 lines of code, you must consider re factoring into separate methods.
* Do not hardcode configuration data. Configuration data should be put into configuration file or database.
* Avoid using member variables. Declare local variables wherever necessary and pass it to other methods instead of sharing a member variable between methods. When sharing a member variable between methods, it will be difficult to track which method changed the value and when.
* Do not make the member variables public or protected. Keep them private and expose public/protected Properties.
* Do not have more than one class in a single file.
* Avoid public methods and properties, unless they really need to be accessed from outside the class. Use “internal” if they are accessed only within the same assembly.
* Declare variables as close as possible to where it is first used. Use one variable declaration per line.
* Use StringBuilder class instead of String when manipulating string objects in a loop. The String object works in weird way in .NET. Each time a string is appended, it is actually discarding the old string object and recreating a new object, which is a relatively expensive operations.
* Never do a 'catch exception and do nothing'. If hiding an exception, there is no way to notice the exception happened or not. Lot of developers uses this handy method to ignore non significant errors. Best practices are to avoid exceptions by checking all the error conditions programmatically. In any case, catching an exception and doing nothing is not allowed. In the worst case, exceptions should be logged for later investigation and system should not be halted.
* When you re throw an exception, use the throw statement without specifying the original exception. This way, the original call stack is preserved.
* Write your own custom exception classes if required in your application. Do not derive your custom exceptions from the base class SystemException. Instead, inherit from ApplicationException.
* A class should be declared as sealed if there is no further inheritance.
* Abstract types should not have constructors (FxCop Microsoft.Design CA1012). Public constructors for abstract types do not make sense because you cannot create instances of abstract types.
* Avoid empty interfaces (FxCop Microsoft.Design CA1040). Interfaces are meant to contain members that specify a set of behaviors. To mark or label a class, use an attribute instead of an empty interface.
* Avoid excessive parameters on generic types (FxCop Microsoft.Design CA1005). Avoid generic types with more than two type parameters as users have difficulties understanding what type parameters represent in types with long type parameter lists.
* Avoid out parameters (FxCop Microsoft.Design CA1021). Using out parameters might indicate a design flaw. Although there are legitimate times to use out parameters, their use frequently indicates a design that does not conform to the design guidelines for managed code.
* Declare types in namespaces (FxCop Microsoft.Design CA1050). A type should be defined inside a namespace to avoid duplication.
* Do not declare static members on generic types (FxCop Microsoft.Design CA1000). The syntax for calling static members on generic types is complex as the type parameter has to be specified for each call.
* Do not declare protected members in sealed types (FxCop Microsoft.Design CA1047). Sealed types cannot be extended, and protected members are only useful if you can extend the declaring type. Sealed types should not declare protected members.
* Do not declare virtual members in sealed types (FxCop Microsoft.Design CA1048). Sealed types cannot be extended, and virtual members are only useful if you can extend the declaring type.
* Do not expose generic lists (FxCop Microsoft.Design CA1002). Do not expose List<T> in object models. Use Collection<T>, ReadOnlyCollection<T> or KeyedCollection<K,V> instead. List<T> is meant to be used from implementation, not in object model API. List<T> is optimized for performance at the cost of long term versioning. For example, if you return List<T> to the client code, you will not ever be able to receive notifications when client code modifies the collection.
* Do not hide base class methods (FxCop Microsoft.Design CA1061). Defining a method in a derived class with the same name and parameters that are more weakly typed as one that is defined in the base class will obstruct access to the method defined in the base class.
* Do not overload operator equals on reference types (FxCop Microsoft.Design CA1046). Most reference types, including those that override System.Object.Equals, do not override the equality operator (==). Most languages provide a default implementation of this operator.
* Do not pass types by reference (FxCop Microsoft.Design CA1045). Although there are legitimate times to use reference parameters, such use frequently indicates a design that does not conform to the design guidelines for managed code.
* Enumerators should be strongly typed (FxCop Microsoft.Design CA1038). Types that implement IEnumerator should also provide a version of the Current property that returns a type other than Object. Implement the interface member explicitly and make the strongly typed version public.
* Exceptions should be public(FxCop Microsoft.Design CA1064). Exception classes should either be public, or have a non-generic public ancestor.
* Implement standard exception constructors (FxCop Microsoft.Design CA1032). Multiple constructors are required to correctly implement a custom exception. Missing constructors can make your exception unusable in certain scenarios. For example, the serialization constructor is required for handling exceptions in XML Web services.
* Lists are strongly typed (FxCop Microsoft.Design CA1039). IList implementations should also provide versions of the IList members that are strongly typed, namely they should specify types other than Object for method and property parameter and return types. Implement the interface members explicitly and make the strongly typed versions public. It is safe to ignore violations of this rule when you are implementing a new object-based collection, such as a linked list, where types based on your collection will determine what the strong type is. These types should expose strongly typed members.
* Override methods on comparable types (FxCop Microsoft.Design CA1036). Types that implement IComparable should redefine Equals and comparison operators to keep the meanings of less than, greater than, and equals consistent throughout the type.
* Replace repetitive arguments with params array (FxCop Microsoft.Design CA1025). Several instances of same-type arguments can be better implemented as a parameter array argument. Generally, if a member declares three or more arguments of the same type, consider using a parameter array.
* Static holder types should be sealed (FxCop Microsoft.Design CA1052). Static holder types do not provide functionality that derived instances can extend. Inheriting from such a type indicates a flawed design.
* Use events where appropriate (FxCop Microsoft.Design CA1030). A method name suggestive of event functionality was encountered.
* Use generic event handler instances (FxCop Microsoft.Design CA1007). Do not declare new delegates to be used as event handlers when targeting a version of the .NET Framework that supports generics. Use an instance EventHandler<T> instead.
* Use properties where appropriate (FxCop Microsoft.Design CA1024). Properties should be used instead of Get/Set methods in most situations. Methods are preferable to properties in the following situations: the operation is a conversion, is expensive or has an observable side-effect; the order of execution is important; calling the member twice in succession creates different results; a member is static but returns a mutable value; or the member returns an array.
* Do not hardcode locale specific strings (FxCop Microsoft.Globalization CA1302). Do not use string literals for system items that have locale-specific designations. Special system locations should be retrieved using provided API such as GetFolderPath. See the System.Environment.SpecialFolder enumeration for more information.
* Set locale for data types (FxCop Microsoft.Globalization CA1306). In most cases, Locale should be explicitly set to CultureInfo.InvariantCulture on DataSet and DataTable instances. Upon creation of a DataSet or DataTable instance, the Locale is set to the current culture. In most cases, the Locale should be set to CultureInfo.InvariantCulture to guarantee proper sorting behavior in all cultures.
* Specify CultureInfo (FxCop Microsoft.Globalization CA1304). If an overload exists that takes a CultureInfo argument, it should always be called in favor of an overload that does not. The CultureInfo type contains culture-specific information required for performing numeric and string operations, such as casing, formatting, and string comparisons. In scenarios where conversion and parsing behavior should never change between cultures, specify CultureInfo.InvariantCulture, otherwise, specify CultureInfo.CurrentCulture.
* Specify IFormatProvider (FxCop Microsoft.Globalization CA1305). If an overload exists that takes an IFormatProvider argument, it should always be called in favour of an overload that does not. Some methods in the runtime convert a value to or from a string representation and take a string parameter that contains one or more characters, called format specifies, which indicate how the value is to be converted. If the meaning of the format specified varies by culture, a formatting object supplies the actual characters used in the string representation. In scenarios where sorting and comparison behavior should never change between cultures, specify CultureInfo.InvariantCulture, otherwise, specify CultureInfo.CurrentCulture.
* Specify StringComparison (FxCop Microsoft.Globalization CA1307). If an overload exists that takes a StringComparison argument, it should always be called in favour of an overload that does not.
* Use ordinal StringComparison (FxCop Microsoft.Globalization CA1309). For non-linguistic comparisons, StringComparison.Ordinal or StringComparison. OrdinalIgnoreCase should be used instead of the linguistically-sensitive StringComparison.InvariantCulture.

## Comments

* Comments should be written in order to make code understandable and maintainable.
* Only write comments when necessary, do not write comments for every line of code and every variable declared.
* Use **//** or **///** for comments. Avoid using **/\* … \*/**
* Do not write comments if the code is easily understandable without comment. The drawback of having lot of comments is that if you change the code and forget to change the comment, it will lead to more confusion.
* Fewer lines of comments will make the code more elegant. But if the code is not clean/readable and there are less comments, that is worse.
* Writes clean, readable code such a way that it doesn't need any comments or very little to understand.
* Perform spelling check on comments and also make sure proper grammar and punctuation is used.

## Coding Conventions Specific for this Project

* All strings must be trimmed before inserting to the database. Empty strings (after trimmed) are inserted as null. (Framework handles this already, just for some special cases)
* All repository methods must return Feedback (or derived) type.
* All repository methods must handle every exception occurring in the methods. All exceptions must be logged as Error.
* All repository methods should be unit tested
* All repository’s insert methods should create a new object based on the parameter object, and then insert the new object to the data context, instead of inserting directly the parameter object.
* Never initiate a repository class by new operator. Use Dependency Injection instead. Except for unit test cases.
* All messages must be stored in Psn.Resources.Messages resource file.
* If a repository method returns a specific message for that method, the message name in Resource file should be: MethodShortName\_MessageName. I.e InsertCategory\_DuplicateName
* All validation messages must be stored in Psn.Resources.ValidationMessages resource file
* All caching items must have cache time configured in the config file AppSettings.config
* All constants must be stored in Psn.Common.Constants class
* All configurable constants must be stored in config file AppSettings.config (i.e. CommentsPerPage)
* All open source libraries must be installed via Nuget (if exist there).
* Never use inline style and javascript. Use CSS and jQuery instead.
* Never use fixed link for internal links. Use routes instead. For example: must not use <a href=”www.place.vn”>..., use Html.ActionLink(“Index”, “Home”) instead.
* Validation attributes must use localization. Use attributes in Psn.Common.DataAnnotations (i.e. RequiredExtended, StringLengthExtended) to use default settings
* Use POST and GET methods appropriately. Use POST for actions that may update data (i.e. update/delete). Use GET for actions to retrieve data only (i.e. get details/list).
* Do not hard delete a record in database if the record has IsDeleted field. Update IsDeleted=1 instead.

# References

[1] Rational\_Unified\_Process model:

<http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process>

[2] .NET Naming Guidelines:

<http://msdn.microsoft.com/en-us/library/ms229002.aspx>

[3] FxCop rules and Code Analysis for Managed Code Warnings

<http://msdn.microsoft.com/en-us/library/ee1hzekz.aspx>