|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | **Logo_FPT_University_doc** | **MINISTRY OF EDUCATION AND TRAINING** |   **FPT UNIVERSITY**   |  | | --- | | Capstone Project | |
|  |
| Cooking recipes sharing website  Report 2: Software Project Management Plan |
| |  |  | | --- | --- | | **Master Chefs team** | | | **Group Members** | Trần Trung Kiên - 01017  Đặng Xuân Toàn - 01143  Trần Sỹ Tùng - 01302  Bùi Trung Kiên - 01359  Ngô Thị Việt Anh - 01286 | | **Supervisor** | Huỳnh Anh Dũng | | **Ext Supervisor** | Nguyễn Tất Trung | | **Capstone Project code** | CTA\_v1.0 | |
|  |

- Hanoi, 5/2012 -

**Record of Changes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Change Item** | **Description** | **By** | **Version** |
| 10/05/2012 | All | Create the document | ToanDX | 0.1 |
| 16/05/2012 | Coding conventions | Update coding conventions | ToanDX | 0.2 |
| 20/05/2012 | All | Add content, edit format document | ToanDX | 1.0 |

Contents

[I. Problem Definition 3](#_Toc333044134)

[1. Name of this Capstone Project 3](#_Toc333044135)

[2. Problem Abstract 3](#_Toc333044136)

[3. Project Overview 3](#_Toc333044137)

[3.1 The Current System 3](#_Toc333044138)

[3.2 The Proposed System 3](#_Toc333044139)

[3.3 Boundaries of the System 4](#_Toc333044140)

[3.4 Development Environment 4](#_Toc333044141)

[II. Project Organization 6](#_Toc333044142)

[1. Software Project Model 6](#_Toc333044143)

[2. Roles and Responsibilities 7](#_Toc333044144)

[III. Project Management Plan 8](#_Toc333044145)

[1. Tasks 8](#_Toc333044146)

[2. Task Sheet: Assignments and Timetable 9](#_Toc333044147)

[3. Meeting Minutes 9](#_Toc333044148)

[IV. Coding Conventions 12](#_Toc333044149)

[1. Naming Conventions 12](#_Toc333044150)

[2. Coding Styles 16](#_Toc333044151)

[3. Comments 21](#_Toc333044152)

[V. Risk management plan 22](#_Toc333044153)

[VI. Project success criteria 23](#_Toc333044154)

[VII. Communication and Reporting 23](#_Toc333044155)

[VIII. References 24](#_Toc333044156)

# Problem Definition

## Name of this Capstone Project

The official and formal project name is **Cook them all**. The product name is **recipes.vn** but it might be different when the project is completed and comes to deployment.

## Problem Abstract

As mentioned in Introduction document, we decide to build a cooking recipe system as a website where people can join and share their recipes, cooking knowledge, tips and experience for other people.

The main problem is how to make this website become a professional guide. The way how users share their recipes is also one of the most important functions. Users need this system to help them find the easy-to-follow recipes and our responsibility is to satisfy these requirements.

## Project Overview

### The Current System

The content of the current websites are taken up mainly by professional chefs or collectors. Interaction among users is not high. Furthermore, the content is often not intuitive, vivid and easy to understand for novices to cooking. The recipes are usually expressed by text makes readers difficult to follow. Some websites have pictures but do not specific for sharing cooking experience.

### The Proposed System

The website will contain 2 main parts: front-end and back-end.

**Front-end:**

|  |  |  |
| --- | --- | --- |
| No | User | Permission |
| 1 | Everyone | * Visit the site and see the cooking recipes and cooking tips, most rated recipes, latest recipes and most shared recipes. * Search for recipes. |
| 2 | Authenticated users | * Post new recipes. * Post comments, rate. * Can report bad contents. * Have a profile page which displays, their personal information, their recipes or recipes of people they followed. * Integrate with some other popular networks (Facebook, Google plus). |

**Back-end:**

|  |  |  |
| --- | --- | --- |
| No | User | Permission |
| 1 | Administrators | * View, search, view details, edit and approve contents. * View, search, add or edit categories. * View, search, ban or delete users. * Process reported bad contents, users. |
| 2 | Power users | * Edit the information of recipes to make it more interesting, clearer. * Delete recipes, comments. |

### Boundaries of the System

**Scope:**

* Support main functions of a normal website such as: customize personal information, view, search…
* Finish special functions of a cooking guide website such as post recipes, questions and answers, comment, personal profile…
* Research and apply algorithms for searching, security technique.

**Future:**

* Resolve all problems that we cannot finish on this project, such as: allow a lot of users connect at the same time; user can interact smoothly with others.
* Develop interface on other devices such as: mobile phone, tablet…

### Development Environment

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

#### 3.4.1 Hardware requirements

* Computer that has a 1.6 GHz or faster processor.
* 1GB (32 Bit) or 2GB (64 Bit) RAM (Add 512 MB if running in a virtual machine).
* 3GB of available hard disk space.
* 5400 RPM hard disk drive.
* DirectX 9 capable video card running at 1024 x 768 or higher-resolution display
* DVD-ROM Drive

#### 3.4.2 Software requirements

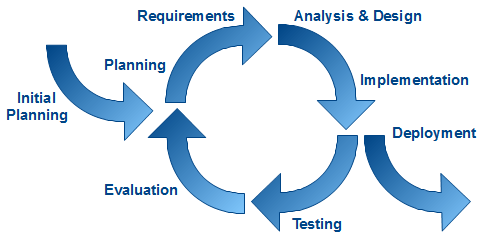
* Front-end technologies: ASP.NET MVC 3.0, jQuery, AJAX, HTML, CSS.
* ORM: Entity Framework.
* Architecture and design patterns: MVC.
* Programming Language: C#.
* Database: SQL Server 2008 R2
* Development tools: Visual Studio 2010, SQL Server 2008, Notepad++,
* Communication tools: Yahoo Messenger, Skype, Google Mail.
* Other tools: Microsoft Office 2010, Microsoft Project 2010, Microsoft Visio 2010.

# Project Organization

## Software Project Model

We choose iterative model to implement because of following reasons:

* This model is suitable for small and medium projects like our project.
* It is difficult to complete this project (especially requirement specification and implementation phases) for a short time.
* The iterative lifecycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. Our project doesn’t have clear requirements at the beginning. Therefore we choose this process model, in order to help project team run this project easier and get the best quality.

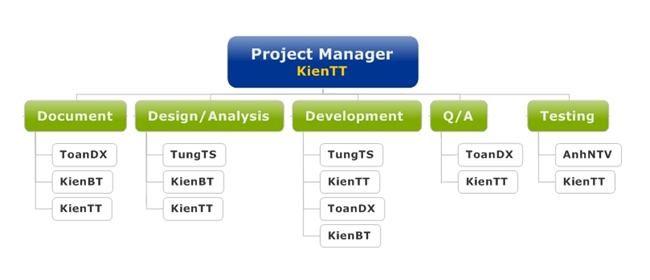


**Iterative Model**

An iterative lifecycle model consists of repeating the following six phases in sequence:

* **Planning phase**, in which the plan for this project is created.
* **Requirements phase**, in which the requirements for the software are gathered and analyzed.
* **Design phase**, in which a software solution to meet the requirements is designed. This may be a new design, or an extension of an earlier design.
* **Implementation phase**, when the software is coded and integrated.
* **Test phase**, when the software is tested.
* **Evaluation phase**, in which the software is evaluated, the current requirements are reviewed; changes and additions to requirements are also proposed.
* **Deployment phase**, in which the system is deployed.

## **Roles and Responsibilities**



|  |  |  |
| --- | --- | --- |
| No | Role in Group | Responsibilities |
| 1 | KienTT  Project Manager | * Manage the project. * Plan, control and monitor processes, and also manage the assigned project resources to meet project objectives. * Identify, monitor, and respond to risk. * Identify needed resource and assign individual responsibilities. |
| 2 | TungTS  Technical Leader | * Design Database. * Design high-level architecture and develop framework. * Training technique for team members. * Resolve technique problems. |
| 3 | KienBT  Business Analysis | * Understand clearly and deeply the project requirements and write SRS. * Assist other team members when they have unclear comprehension of business and project requirements * Report to manager about requirements and related problems. |
| 4 | ToanDX  Quality Assurance | * Control quality and managerial processes, control quality of reports after reviewed by PM. * Review SRS * Send daily emails for team members about current status of processes. If a process isn’t finished on time, QA must raise this issue in the next team meeting. |
| 6 | KienTT, TungTS, ToanDX, KienBT  Developer | * Code and execute unit tests. * Provide coding assistance for other members. |
| 7 | AnhNTV  Tester | * Create Test Plan and Test Scenarios * Write test reports. * Take part in all testing phases. * Take part in test team’s meetings. * Test the system, create testing reports. |

# Project Management Plan

## Tasks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Milestones** | **Description** | **Output** | **Deliverables** | **Resources Needed** |
| Initial idea | Start discussing about topic of capstone project | Topic of capstone project | Deliver initiator idea before 07/05/2012 | 5 people for 2 weeks |
| Create Project plan | Create project plan | Project plan | Deliver project plan before 22/05/2012 | 2 people for 1 weeks |
| Create Software Requirements Specification | Create Software Requirements Specification | Software Requirements Specification (SRS) document | Deliver SRS document before 26/06/2012 | 4 people for 4 weeks |
| Design Database | Create logical and physical database design | Database design and database script | Deliver database design and database script before 05/06/2012 | 2 people for 2 days |
| Create Soft Design Description  (SDD) | Design the system in OOP manner | Architecture design, detailed design, diagrams and design specification | Deliver SDD before 10/07/2012 | 3 people for 2 weeks |
| Layout Design | Create the main GUI for layout, and create global styles that will be applied to the GUI | HTML layout and CSS files | None | 1 person for 4 days |
| Create Coding Framework | Map the architecture design into source code, create the project solution files and common classes, and implement common functions | Visual Studio project and solution files containing coding framework | Deliver coding framework before 11/06/2012 | 1 person for 4 days |
| Coding | Implement the system to reflect the requirements | Source code of the project, unit test reports | Executable programs and source code before 15/08/2012 | 4 people for 4 weeks |
| System Test | Perform system test for the system | System test report | Deliver Software Test Documentation before 17/08/2012 | 5 people for 1.5 weeks |
| Input initial data | Input Initial data | Initial data | Deliver before 17/08/2012 | 5 people for 1 weeks |
| Deployment | Deploy the system to the internet | Running website with domain and hosting | None | 2 people for 3 days |

## Task Sheet: Assignments and Timetable

Refer to Master\_Plan in attached file.

## Meeting Minutes

At the end of each meeting, scribe collect all information write a meeting minute document.

**Weekly meeting minute 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | Meeting minute | **Date** | 07/05/2012 |
| **Facilitator** | Detech Building | **Time** | 9:00 – 10:00 |
| **Location** | FPT University – 8th Ton That Thuyet Str – Tu Liem – Ha Noi | **Scribe** | KienBT |
| **Attendees** | DungHA (Supervisor),TrungNT(Supervisor), KienTT (Team Lead), KienBT, ToanDX, TungTS, AnhNTV | | |
| **Absent** |  | | |

| **Key Points Discussed** | | |
| --- | --- | --- |
| No. | Topic | Description |
| 1. | Set up team, project | Naming team, project |
| 2. | Make schedule for meeting | Time: Everyday Monday: 8h to 11h  Place: 2nd floor, Detech building |
| 3. | Define project objectives |  |
| 4. | Discuss report 1 |  |

| **Action Plan** | | | |
| --- | --- | --- | --- |
| No. | Action Item(s) | People | Deadline |
| 1. | Writing Report 1: Introduction | KienBT, ToanDX | 15/05/2012 |
| 2. | Research existing system and their limitation | KienBT, ToanDX | 15/05/2012 |
| 3. | Draw briefly proposed system | KienTT, TungTS | 15/05/2012 |
| 4. | Start learning tutorial .NET MVC 3, HTML, Javascript, AJAX | KienTT, TungTS |  |
| 5. | Start learning Web design | KienTT |  |

**Kick-off meeting minute**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | Kick-off Meeting | **Date** | 10/05/2012 |
| **Facilitator** | Detech Building | **Time** | 9:00 – 10:00 |
| **Location** | FPT University – 8th Ton That Thuyet Str – Tu Liem – Ha Noi | **Scribe** | KienBT |
| **Attendees** | DungHA (Supervisor),TrungNT(Supervisor), KienTT (Team Lead), KienBT, ToanDX, TungTS, AnhNTV | | |
| **Absent** |  | | |

| **Key Points Discussed** | | |
| --- | --- | --- |
| No. | Topic | Description |
| 1. | Team member introduction | Introduce all members of team to supervisor |
| 2. | Project introduction | Introduce briefly about the project |
| 3. | Talk about project | SMARTER structure, slides, report 1… |

| **Action Plan** | | | |
| --- | --- | --- | --- |
| No. | Action Item(s) | People | Deadline |
| 1. | Writing Report 1: Introduction | KienBT, ToanDX | 15/05/2012 |

**Weekly meeting minute 2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | Meeting minute | **Date** | 14/05/2012 |
| **Facilitator** | Detech Building | **Time** | 9:00 – 10:00 |
| **Location** | FPT University – 8th Ton That Thuyet Str – Tu Liem – Ha Noi | **Scribe** | KienBT |
| **Attendees** | DungHA (Supervisor),TrungNT(Supervisor), KienTT (Team Lead), KienBT, ToanDX, TungTS, AnhNTV | | |
| **Absent** |  | | |

| **Key Points Discussed** | | |
| --- | --- | --- |
| No. | Topic | Description |
| 1. | Check working process | Check assigned tasks last week |
| 2. | Discuss detail proposed system | Define main features and set priority |
| 3. | Review and update report 1 |  |

| **Action Plan** | | | |
| --- | --- | --- | --- |
| No. | Action Item(s) | People | Deadline |
| 1. | Prepare slide presentation and project Plan, learning MVC and JavaScript LinQ to SQL, Ajax | KienBT, ToanDX | 15/05/2012 |
| 2. | Continue learning technology and make demo | KienTT, TungTS |  |

**Weekly meeting minute 3**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | Meeting minute | **Date** | 21/05/2012 |
| **Facilitator** | Detech Building | **Time** | 9:00 – 10:00 |
| **Location** | FPT University – 8th Ton That Thuyet Str – Tu Liem – Ha Noi | **Scribe** | KienBT |
| **Attendees** | DungHA (Supervisor),TrungNT(Supervisor), KienTT (Team Lead), KienBT, ToanDX, TungTS, AnhNTV | | |
| **Absent** |  | | |

| **Key Points Discussed** | | |
| --- | --- | --- |
| No. | Topic | Description |
| 1. | Complete Master Plan | Make the schedule, ands milestones more detailed. |
| 2. | Review and update report 2 |  |

| **Action Plan** | | | |
| --- | --- | --- | --- |
| No. | Action Item(s) | People | Deadline |
| 1. | Prepare slide presentation | ToanDX | 23/05/2012 |
| 2. | Continue learning technology and make demo | KienTT, TungTS, ToanDX, KienBT |  |

# 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:

IO

***Naming rules***

* Use Pascal casing for Class names.

public class HelloWorld

{

...

}

* 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.
* 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>***

* Use appropriate prefix for the UI elements to identify them from the rest of the variables.

|  |  |
| --- | --- |
| **Control** | **Prefix** |
| Label | lbl |
| TextBox | txt |
| DataGrid | dtg |
| Button | btn |
| ImageButton | imb |
| Hyperlink | hlk |
| DropDownList | ddl |
| ListBox | lst |
| DataList | dtl |
| Repeater | rep |
| Checkbox | chk |
| CheckBoxList | cbl |
| RadioButton | rdo |
| RadioButtonList | rbl |
| Image | img |
| Panel | pnl |
| PlaceHolder | phd |
| Table | tbl |
| Validators | val |

* 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 A1724)

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.

# Risk management plan

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Risk Description | Risk type | Prob-ability  (1-5) | Effect  (1- 5) | Risk Strategy | Resolution |
| Member’ illness | People | 2 | 4 | Build a reasonable working schedule. | - PTL and PM monitor other members’ tasks and always ready to do these tasks instead of them if can  - Using buffer time to recover works |
| Bad attitude of team member: in-cohesive with team, breach of discipline | People | 3 | 5 | - Engage teamwork.  - Team-building | Create team’s policies: rewarding and punishment |
| Working productivity is not high | Technical | 4 | 4 | - Define standards to be followed to meet the performance criteria  - Training technology seriously and research all related problem | - Using buffer time to recover works  - Cross-review tasks to track working process |
| Design Difficulty: Interface is not attractive, reasonable | Technical | 3 | 3 | Before designing, research in existing system to find out their advantage and disadvantage. | Design the interface with the instructions of an expert. |
| Working progress is not meeting schedule | Time | 3 | 5 | Create buffer in plan | Using buffer time to recover works |

# Project success criteria

|  |  |  |
| --- | --- | --- |
| **Project success criteria** | **Measurement** | **Weight** |
| All high-priority functions defined in the requirement specification must be finished at the end of project (31/8/2012) | All high-priority functions are tested and released on time | 40 |
| Complete all tasks within the predetermined time | All team member’s tasks have to meet the deadline | 30 |
| Team working | Team members can support and help each other. Software process must be carried out correctly. | 15 |
| Number of defects per function is acceptable | Number of defects per function is not exceed 10 | 15 |

# Communication and Reporting

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Communication | Method / Tool | Frequency / Schedule | Information | Participants / Responsible |
| Internal Communication: | | | | |
| Project Meetings | Face to face | Weekly and when required | Project status, problems, risks, changed requirements | Team Members |
| Sharing of project data | Shared Project Server | When available | All project documents and reports | Team Members |
| Milestone Meetings | Face to face | Before milestones | Project status | Team Members |
| Final Project | Face to face | End of project | Project result, lessons | Supervisor |
| External Communication and Reporting: | | | | |
| Project Report | Excel sheet | On event | 5 reports | Project Team |
| Meetings with Supervisor | Face to face | 2 times / week | Project status, problems, risks, change requirements, schedules | Supervisor, Team Members |

# References

* + [1] Wikipedia Iterative and incremental development
  + [http://en.wikipedia.org/wiki/Iterative\_and\_incremental\_development](http://en.wikipedia.org/wiki/Iterative_and_incremental_development%20)
  + [2] .NET Naming Guidelines
  + [http://msdn.microsoft.com/en-us/library/ms229002.aspx](http://msdn.microsoft.com/en-us/library/ms229002.aspx%20)
  + [3] FxCop rules and Code Analysis for Managed Code Warnings
  + [http://msdn.microsoft.com/en-us/library/ee1hzekz.aspx](http://msdn.microsoft.com/en-us/library/ee1hzekz.aspx%20)
  + [4] ITPro portal Top ten software development risks
  + [http://www.itproportal.com/2010/06/14/top-ten-software-development-risks](http://www.itproportal.com/2010/06/14/top-ten-software-development-risks%20)
  + [5] Wikipedia Project management
  + <http://en.wikipedia.org/wiki/Project_management>