|  |
| --- |
|  |
| 개발규칙 가이드 |
|  |
|  |
| **홍길동** |
|  |
|  |

|  |
| --- |
|  |

목차

[1. 개발 표준 소프트웨어 아키텍처 2](#_Toc77577297)

[1.1 화면 및 UI 구성 2](#_Toc77577298)

[1.2 데이터베이스 처리 2](#_Toc77577299)

[1.3 표준 개발규칙 사용 목표 2](#_Toc77577300)

[2. 아키텍처 구성과 패턴 4](#_Toc77577301)

[3. 명명의 정의 7](#_Toc77577302)

[1.1 개발 명명 표기유형 종류 7](#_Toc77577303)

[2 작명 시 주의사항 8](#_Toc77577304)

[2.1 변수, 클래스, 메서드 명명 규칙 8](#_Toc77577305)

[3. 네임스페이스 10](#_Toc77577306)

[4. 클래스의 정의 10](#_Toc77577310)

[4.1 개체 구성의 실체 10](#_Toc77577315)

[4.3 클래스 접근권한 10](#_Toc77577322)

[4.4 클래스의 상속 10](#_Toc77577323)

[5. 공통 명명규칙 11](#_Toc77577324)

[5.1 명명 표기법 기본 11](#_Toc77577330)

# 개발 표준 소프트웨어 아키텍처

## 화면 및 UI 구성

개발 플랫폼에서 .NET Framework 역할은 응용 프로그램 만들 수 있는 플랫폼을 제공합니다. 이 플랫폼을 기반으로 Microsoft 플랫폼에서 사용되는 응용 프로그램을 개발하므로 문제 없는 소프트웨어를 개발하는 것을 목표로 아키텍처를 구성합니다.

소프트웨어 아키텍처는 시스템 구조들로 각 구조의 역할 부분에 소프트웨어 컴포넌트들을 구성하여 화면에 표시되는 속성들과 내부 구조들 간의 관계로 이루어 졌습니다.

화면에 표시되는 것을 User Interface라 명명하며, 이 부분에 표시되는 부분은 ASP.NET, Windows 응용 프로그램, RIA 응용 프로그램이 있습니다. 이 응용 프로그램일 기반으로 화면에 표시되는 구성요소를 선택하여 화면 구성을 완료합니다.

## 데이터베이스 처리

Database 는 WCF 서비스를 이용한 데이터 전송을 구성하는 것을 원칙으로 합니다. XML Web Service 의 경우 대 용량 데이터 전송 등의 취약점을 해결하기 위한 부분이며, .NET Framework 기반의 플랫폼에서 데이터 베이스에 접속하여 데이터를 처리 할 경우 매우 효율적으로 증명된 데이터 전송 방법입니다. MS SQL Server 기반의 데이터를 처리 하기 위하여 WCF 서비스를 이용합니다.

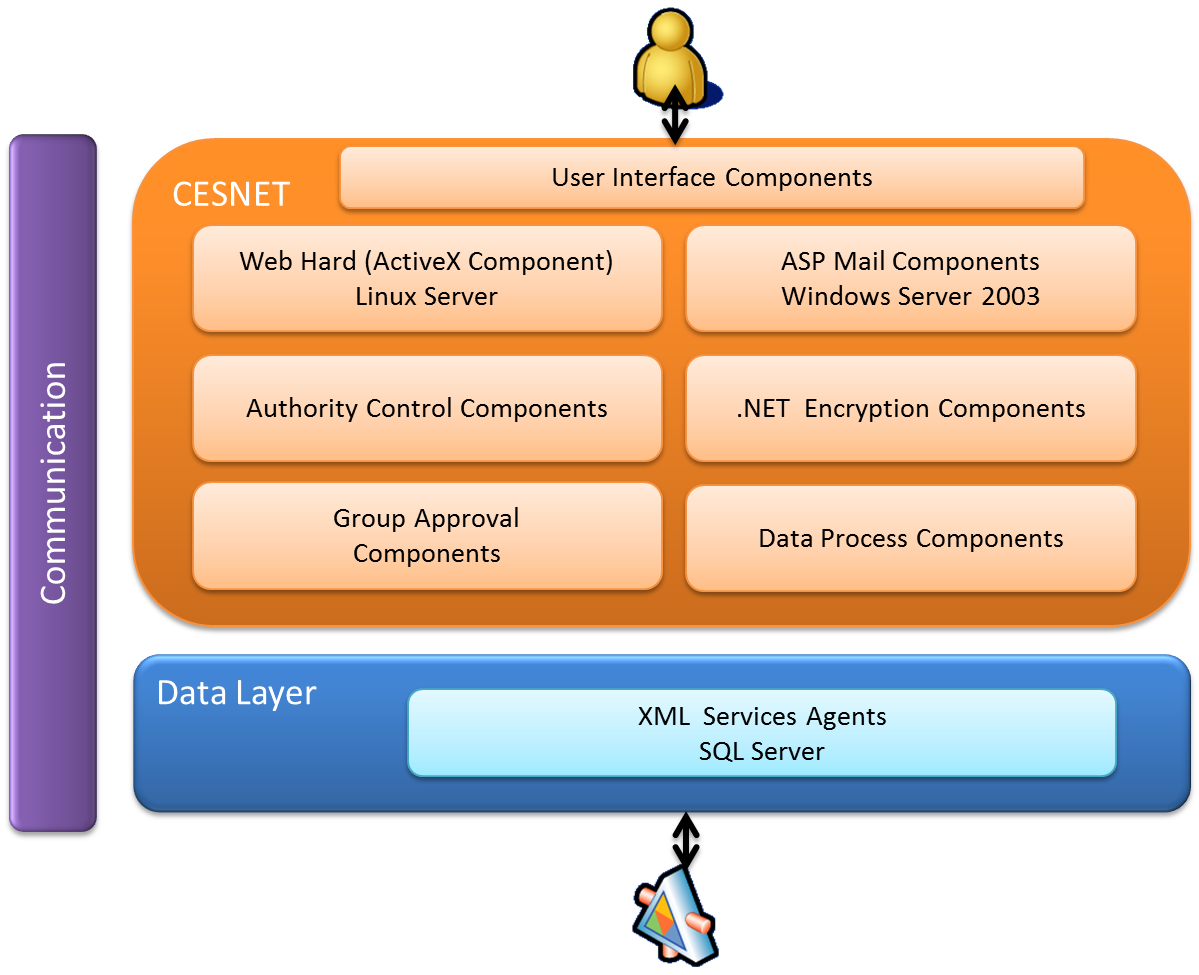
Linq 사용에 대하여는 개발자와 데이터베이스 쿼리를 만드는 사람의 협의하여 구성합니다. 기본 권장사항은 Linq와 저장프로시저 중 하나를 선택하여 구성합니다.

## 표준 개발규칙 사용 목표

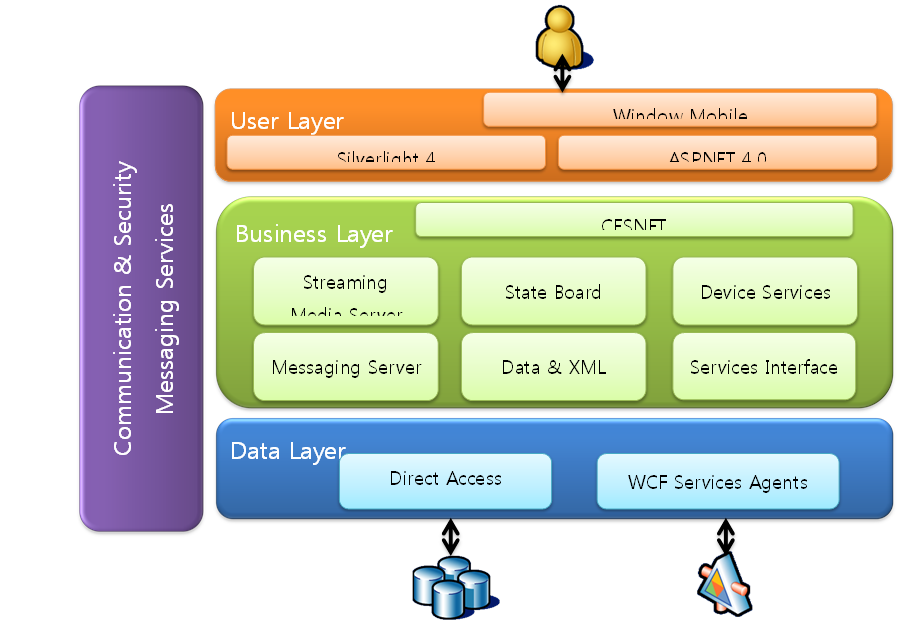
표준 개발 규칙을 사용하여, 사용자 요구사항에서부터 개발 완료까지 일련의 프로세스를 자동화 함으로 업무 처리에 있어서 자연스럽게 업무 패턴을 적용함으로써 개발 완료까지 소요되는 시간 및 인력 투자에 대한 비용 절감과 적재 적소에 인력을 조정함으로써 업무의 효율성을 높이는 것을 목표로 합니다. 새로운 시스템 개발에 공용 플랫폼과 표준 프레임 워크를 적용함으로써 유지보수의 편의성을 증대하여 개발 완료된 응용 프로그램의 유지보수 시간을 최대한 줄임으로써 유지보수 비용과 인력을 줄여 비용 절감과 총 소요비용의 절감을 목표로 합니다.

# 아키텍처 구성과 패턴

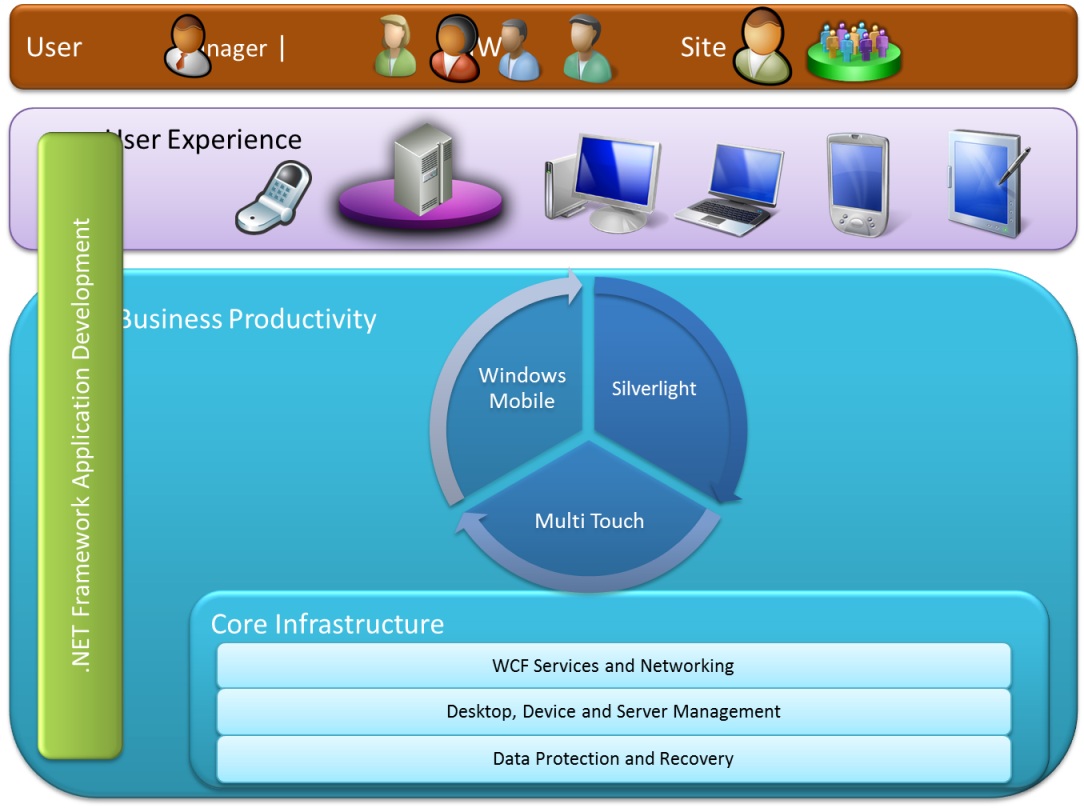
기존 구성의 아키텍처는 다음과 같이 구성되어 있습니다.



일반적인 구성 환경으로 개발을 진행하였으며, 다음과 같은 구조로 변경합니다.



개발 구조를 변경함으로 개발 시간 단축과 필요 구성요소의 재사용으로 중점으로 구조를 변경하여 개발자들은 개발에 필요한 시간을 최대한 줄이면서 화면 구성과 업무 담당자와의 협업 시간도 줄일 수 있습니다.



XXX 플랫폼의 가치는 Windows 기반의 플랫폼에서 어느 클라이언트에서도 별도의 개발을 구성하는 것이 아니라 하나의 표준 화면과 하나의 표준 개발 환경으로 Windows 환경 Web환경, 모바일 환경을 지원하는 것을 목표로 합니다.

# 명명의 정의

프로젝트 진행 시에 개발자들이 지켜야 하는 몇 가지 규칙을 정합니다. 이 문서에서 정의하는 규칙은 개발자들이 지켜야 하는 필수 규칙을 정의합니다.

이 문서에 정의되어 있는 규칙은 개발 시에 체크인 시에 적용되는 정책으로 개발 전에 미리 숙지해야 합니다. 체크인 정책 적용 시 명명 정의를 지키지 못할 경우 개발 시 문제가 발생할 수 있습니다.

## 개발 명명 표기유형 종류

1. Pascal Case

* 각 간어의 첫 문자는 대문자로 시작
* 예제) MyClass

1. Camel Case

* 첫 문자는 소문자로 시작하여 그 이후의 단어는 대문자로 시작
* 예제) totalCount

1. Upper Case

* 모둔 문자는 대문자로 표시
* 예제) COUNT

1. Hungarian Notation

* Pascal Case 유형 앞에 변수 타입의 약어를 표시
* 예제) intCount

# 작명 시 주의사항

이 기종간의 언어와 상호 연동 시에 표기법의 혼동을 막기 위하여 대소문자 구분을 엄격히 제한하여 지켜야 하며, 언어의 특성으로 대소문자를 구별할 수 있습니다. 대 부분의 프로젝트는 C#으로 이루어지며 대소문자를 구별합니다.

* C/C++, C# : 대소문자 구분
* Basic : 대소문자 구분하지 않음

## 변수, 클래스, 메서드 명명 규칙

프로젝트에서 변수 명, 클래스 명, 메서드 명등을 정의하기 위하여 다음과 같은 규칙을 지켜야 합니다.

변수 명 : Camel Case

* + totalCount

메서드 명 : Pascal Case

* + MyMethod()

클래스 명: Pascal Case

* + MyClass()

구조체 명: Upper Case

* + MyStruct

상수 명: Upper Case

* + MyCONST = 1

컴포넌트명: 대소문자를 구분하지 않음

예제) System.IO

대소문자로 구분되는 파라미터, 속성, 메서드는 사용하지 안습니다.

예제) private int UserLogin(string userID, userPassword)

int ThreadTotal{ get ; set ;}

void UserLoginSetting {}

명명 시에는 .NET Framework 에서 사용되는 클래스 명, 메서드, 내장 함수 명을 사용하지 않습니다.

명명 시에는 “동사”+”명사” 또는 “명사” + “명사”를 기본으로 합니다.

# 네임스페이스



* 어셈블리에 정의된 개체를 구성하는 어셈블리는 여러 개의 네임스페이스로 정의 가능하며, 각 네임스페이스는 참조 가능합니다.
* 네임스페이스를 정의할 때에는 모호성을 방지하여야 합니다.
* 참조 단순화를 구현하기 위하여 네임스페이스는 10자이상을 넘지 않도록 정의하고 10자를 넘길 경우 협의 후에 약어를 이용하여 10자 이내로 작명합니다.

# 클래스의 정의



개체지향 프로그래밍에서 매우 중요한 클래스는 해당 개체의 근간이 되며, 관련 항목을 하나의 단위로 그룹화한 것입니다.

## 개체 구성의 실체

* 메서드
* 속성
* 이벤트



## 클래스 접근권한

* Public : 제한 없음
* Internal : 동일한 어셈블리 내
* Protected : 파생된 클래스까지
* Protected internal : 현재까지 프로젝트 / 참조된 클래스에서 파생된 클래스 까지
* Private : 현재 정의된 곳 내

## 클래스의 상속

* None : 제한 없음
* Abstract : 클래스를 다른 클래스의 기본 클래스로만 사용가능, 인스터스 불가능
* Sealed : 클래스 상속불가

# 공통 명명규칙



## 명명 표기법 기본

|  |  |  |
| --- | --- | --- |
| **식별자** | **유형** | **용례** |
| 네임스페이스(Namespace) | User Define | CESNET.”<팀명>” |
| 솔루션(Solution) | User Define | CSE+솔루션명 |
| 프로젝트(Project) | User Define | CESNET.”<팀명>”. Project |
| 라이브러리(cs) | Pascal | CESNET. + “기능” + lib |
| 클래스(Class) | Pascal | MessageHandler |
| 예외 클래스(Exception Class) | Pascal | HandleException  항상 Exception으로 종료 |
| 인터페이스(Interface) | Pascal | IDisposasble  항상I로 시작 |
| 메서드(Method) | Pascal | ToString |
| 프로퍼티(Property) | Pascal | BackColor |
| 이벤트(Event) | Pascal | ValueChange |
| 컨트롤(Control) | Hungarian | txtUserName |
| 변수(Variable) | Camel | documentLibaray |
| 상수(Constant) | Upper Case | MAX\_VALUE |
| 구조체(Structure) | Upper Case | MYSTRUCT |
| 파라미터(Parameter) | Camel | typeName |
| 열거형의이름(Enum type) | Pascal | ErrorLevel |
| 열거형의 값(Enum values) | Pascal | FatalError |
| 읽기전용 static 필드 | Pascal | RedValue |