Implementasi *Model View Presenter* dan *Object Relational Mapping* NHibernate pada Aplikasi *eStop Card* berbasis *Web* (Studi Kasus: PT. XYZ Jakarta)

Alessandro Christoforus B [1]
andobaramuli@gmail.com

Adi Nugroho [2]
adi.nugroho@staff.uksw.edu

Angela Atik Setiyanti [3]
setya_angela@yahoo.co.id

Abstract

The development of technology has shown significant progress, especially in application architecture design pattern. Model View Presenter (MVP) pattern is one of the most important pattern used for extracting business logic outside of User Interface (UI) elements and by that enabling unit test without the need testing tools, and specifically geared towards for page event model in ASP.NET. Together with MVP, Object Relational Mapping (ORM) NHibernate is implemented to build a web base application. ORM is a technique to map an object to a database. It can be concluded that the implementation MVP and ORM to build a web base application resulting a well-performed and easily maintained application as well as it effectiveness when performing unit test, and also a clear separation layer between model, view, presenter.

Keywords: *Model View Presenter (MVP), Object Relational Mapping (ORM).*

1. PENDAHULUAN

Dalam pengembangan perangkat lunak aplikasi, *design pattern* merupakan solusi yang dibuat untuk menyelesaikan masalah yang bersifat teknis dengan menyediakan suatu arsitektur aplikasi. *Design pattern* tidak hanya berupa suatu pola desain yang secara langsung diterjemahkan ke dalam barisan kode program, tetapi merupakan suatu deskripsi atau model untuk menyelesaikan masalah yang dapat digunakan pada situasi-situasi berbeda. Hal ini secara khusus menunjukkan hubungan antara kelas dan obyek dalam arsitektur aplikasi.

Arsitektur suatu aplikasi merupakan desain standar yang digunakan untuk mengatasi isu-isu dalam rekayasa perangkat lunak, seperti keterbatasan performa perangkat keras komputer, memperkecil resiko bisnis, dan isu yang lainnya. Suatu arsitektur aplikasi dapat

[1] Mahasiswa*Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana, Yogyakarta*
[2] *Staf pengajar, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana, Yogyakarta*
[3] *Staf pengajar, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana, Yogyakarta*

dipisahkan ke dalam beberapa inti *layer* atau komponen aplikasi. Dalam membangun suatu arsitektur aplikasi, ada penerapan suatu teknik pemrograman. Hal ini memberikan kesempatan kepada para pengembang aplikasi untuk mengembangkan aplikasi dengan performa terbaik.

Banyak teknik pemrograman yang dapat digunakan dalam mengembangkan aplikasi. Salah satu teknik yang digunakan untuk mengembangkan aplikasi adalah *Model View Presenter* (MVP). Teknik MVP diimplementasikan dengan tujuan untuk memisahkan tanggung jawab dari masing-masing bagian yaitu *Data Representation, User Interface* (UI), dan *Business Logic* aplikasi dalam tiga bagian besar yaitu *model*, *view*, dan *presenter*. Dalam melakukan implementasi *Model View Presenter* (MVP) di dalamnya ditambahkan suatu teknik *Object Relational Mapping* (ORM). Teknik ORM digunakan untuk memetakan setiap atribut dari suatu obyek ke dalam tabel pada basis data. Penelitian ini ditujukan untuk melakukan implementasi teknik pemrograman *Model View Presenter* (MVP) serta teknik *Object Relational Mapping* (ORM) *NHibernate* dalam membuat aplikasi *eStop Card* berbasis *web* pada modul *Master Management* di PT. XYZ.

2. KAJIAN PUSTAKA

Setelah bertahun-tahun melakukan perawatan ribuan baris kode *ASP Spaghetti*, pada akhirnya Microsoft mengeluarkan suatu *platform* untuk pengembangan *web* yaitu ASP.NET. Secara langsung, ASP.NET memberikan pemisahan dasar antara *presentation layer* dan *business logic layer* dengan memperkenalkan *code behind page*. Meskipun telah diperkenalkan dengan baik dan sempurna untuk suatu aplikasi, *code behind page* masih memiliki sejumlah kekurangan dalam aspek sebagai mediator antar *layer* yang menyebabkan banyaknya tanggung jawab yang harus dikerjakan, sulit untuk menggunakan *presentation logic* antar *code behind page* tanpa mendaftarkan *helper/utility*, sulit melakukan *unit testing* pada *code behind page* [1].

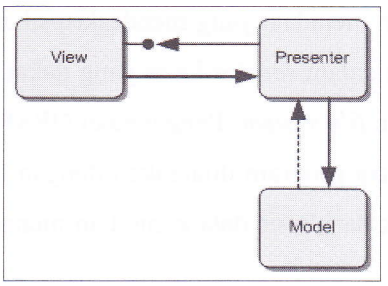
Pada jurnal dengan judul *Model View Presenter design pattern* menjelaskan bahwa MVP adalah suatu tipe *User Interface* (UI) *design pattern* yang mengijinkan pemisahan antarmuka pengguna dan lapisan akses data [2]. Salah satu keuntungan dengan menggunakan teknik MVP adalah mudah melakukan pengujian *action* sederhana tanpa menggunakan *tool* khusus pengujian. Semua *logical process* diletakkan dalam *presenter* dan interaksi antara *view* dan *presenter* hanya melalui *interface class*, sehingga setiap obyek tiruan dapat digunakan untuk pengujian selama itu mengimplementasikan *view interface* [3].

NHibernate in Action, menjelaskan tentang model pemrograman dimana suatu aplikasi dapat dilakukan penambahan perancangan dan penerapan suatu kelas *persistent*

untuk bagaimana kelas tersebut mengidentifikasi *business object* (entitas) dan memetakannya ke dalam basis data menggunakan teknik ORM dan pendekatan XML [4]. Berdasarkan beberapa teori dan konsep yang dikemukakan oleh beberapa orang tersebut, maka peneliti mencoba melakukan suatu implementasi MVP dan ORM dalam membuat aplikasi *eStop Card* berbasis *web* pada modul *Master Management*.

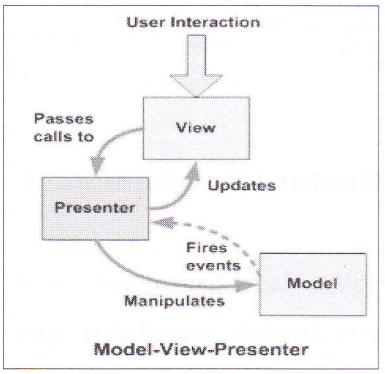
2.1. *Model View Presenter* (MVP)

MVP adalah teknik pemrograman yang secara khusus mengarah ke *page event model* seperti ASP.NET [1]. Arsitektur MVP terlihat pada Gambar 1.



Gambar 1 Arsitektur MVP[5]

Model bisa dikatakan sebagai *domain level object*, dan dikenal sebagai *business object*. Tugas utama *model* adalah menjaga data dan fungsi/*method* pada aplikasi untuk secara konsisten dapat diakses. *View* adalah bagian yang bertanggungjawab terhadap representasi antarmuka dengan pengguna. *Presenter* adalah bagian yang bertanggungjawab untuk memanipulasi *model* sebagai respon terhadap *input* dari pengguna. Proses interaksi pengguna akhir aplikasi yang dibangun dengan menggunakan arsitektur MVP, dapat dilihat pada Gambar 2.



Gambar 2 Alur kerja MVP [6]

Pada Gambar 2, dimulai dengan pengguna melakukan permintaan dan interaksi langsung dengan *view* dan *event* ini diambil alih oleh beberapa *life-cycle event* pada *view/page* kemudian *presenter* melakukan *get* atau *set* data dari *bussines logic*. *Presenter* mengisi *view* dengan memberikan *properties* dan tampilan yang diberikan merupakan respon yang dikirimkan kepada pengguna. Pemisahan fokus *layering* direalisasikan pada *IView Interface*. Oleh sebab itu *presenter* dapat mengetahui apa yang menjadi tugasnya dan bertindak sesuai dengan fungsinya.

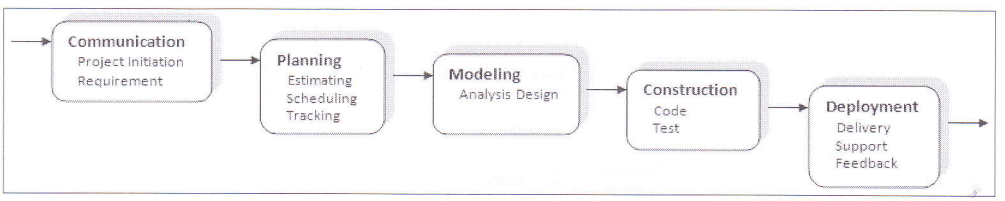
2.2. *Object Relational Mapping* (ORM) NHibernate

ORM adalah teknik otomasi dan transparansi dari *object persistence* ke dalam tabel pada basis data, menggunakan metadata yang mendeskripsikan pemetaan antara obyek dan basis data [7]. ORM berperan dalam *service layer* yang dekat dengan sumber data, baik itu dari *database*, *webservice*, dan *file system*. Penggunaan ORM dibuat untuk melakukan aksi terkait komunikasi obyek ketika program dijalankan dengan basis data seperti menyimpan obyek, mengambil data obyek dari basis data kemudian menampilkannya, menghapus, dan mengubah obyek.

3. PERANCANGAN SISTEM

3.1. Motede Pengembangan Sistem

Dalam penelitian ini, pengembangan aplikasi dilakukan menggunakan metode *waterfall*. Pada *waterfall*, pengembangan aplikasi dilakukan dengan fasenya terorganisasi dalam *linear order* sehingga dapat menjamin pengembangan aplikasi dengan adanya dokumentasi dan desain untuk memastikan kualitas, tahan uji, dan perawatan dari aplikasi yang dikembangkan.



Gambar 3 *Waterfall Model* [8]

Suatu pendekatan *waterfall* terdapat bagian atau sarana yang di dalamnya dilakukan secara berurut atau langkah demi langkah, menuliskan spesifikasi suatu sistem kemudian adanya suatu persetujuan, serta pengguna mempunyai batasan tertentu dalam suatu sistem. Dalam hal ini dimaksudkan pendekatan *waterfall* dilakukan secara bertahap dan berskala, kalau sudah berada pada tahap berikutnya, berarti tidak bisa kembali ke tahap sebelumnya.

3.2. Analisis Kebutuhan

Analisis kebutuhan dalam perancangan aplikasi *eStop Card* merupakan penyesuaian dari dokumentasi aplikasi di PT. XYZ, yang mendeskripsikan tentang proses bisnis dari *Stop Card*, dokumen yang digunakan, dan aturan bisnis sistem. **Proses Bisnis** dari *Stop Card* di PT. XYZ adalah serangkaian aktifitas yang mencatat dan mengevaluasi tingkat kewaspadaan setiap karyawan yang bekerja di fasilitas perusahaan. Pencatatan tingkat kewaspadaan karyawan dilakukan dengan menuliskan aktifitas yang terlihat yaitu berupa kesalahan dari seorang karyawan dan dicatatkan pada satu *manual stop card*. Terdapat 3 (tiga) bagian besar dalam proses *Stop Card* yaitu *Manual Stop*, *Master Management*, dan *Report*. Dan dalam pengembangan aplikasi *eStop Card* berbasis *web*, analisis kebutuhan dibatasi pada bagian *Master Management*.

Dokumen yang digunakan sebelum menggunakan sistem terintegrasi adalah dokumen *stop card* serta laporan statistik dari Departemen *Health Safety Environmental* dan Departemen *Operation*. Dokumen *stop card* adalah dokumen formal yang mencatat dan mengevaluasi tingkat kewaspadaan karyawan dalam bekerja di fasilitas perusahaan. Sedangkan laporan statistik yang digunakan dalam bentuk *Microsoft Office Excel* berisi informasi berupa jumlah *manual stop card* yang dilaporkan, tingkat kewaspadaan karyawan perusahaan, aktifitas yang dilakukan karyawan dari suatu departemen.

Aturan bisnis sistem dari aplikasi *eStop Card* dibatasi pada modul *Master Management*. *Master Management* merupakan modul dengan level tertinggi dalam aplikasi yang akan dikembangkan. Pada modul *Master Management*, seorang pengguna yang terautentikasi oleh sistem dapat melakukan tambah, lihat, ubah, dan hapus data pada sembilan submodul dari modul *Master Management*. Dalam modul *Master Management* pengguna dapat mengatur submodul:

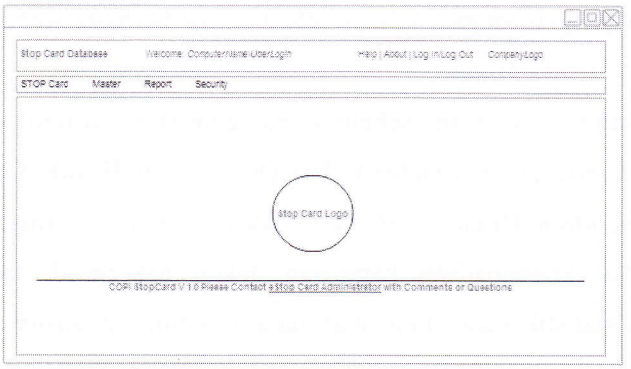
- *Departement Data*: departemen target observasi dan departemen *observer*.
- *Location Data*: lokasi target observasi.
- *Workgroup Data*: kelompok target observasi.
- *Company Data*: perusahaan *observer* yang mana tidak memiliki fungsional ID.
- *Category and Sub Category Data*: kategori dan sub kategori dari halaman 1 – *STOP Category* dan halaman 2 – *STOP root cause in STOP Card Page*.
- *Status*: status observasi.
- *Intervention level*: level
- *Activity Observed*: aktifitas observasi

3.3. Perancangan

Tahap perancangan dilakukan untuk memberikan gambaran awal yang jelas tentang apa yang harus dikerjakan dalam tahap pembuatan aplikasi. Rancangan yang dibuat juga dapat dipakai sebagai dokumentasi aplikasi.

3.3.1. Rancangan Tampilan Aplikasi

Gambar 4 adalah bentuk dasar dari tampilan aplikasi. Aplikasi berbasis *web* dengan tampilan dasar yang terdiri dari *header*, *menu drop down*, dan *content*.

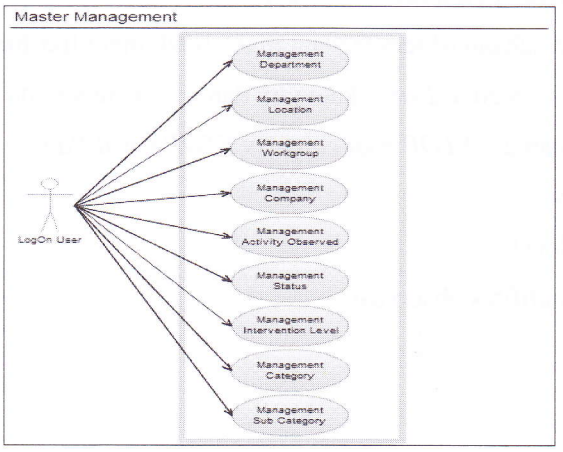


Gambar 4 Rancangan Tampilan Aplikasi

3.3.2. Rancan gn Dia gam *Use Case*

Use Case ini didesain secara keseluruhan, dengan menggambarkan aktor yang terlibat dalam satu atau lebih proses. Semua proses dalam sistem membutuhkan campur tangan aktor yang terautentikasi oleh sistem.

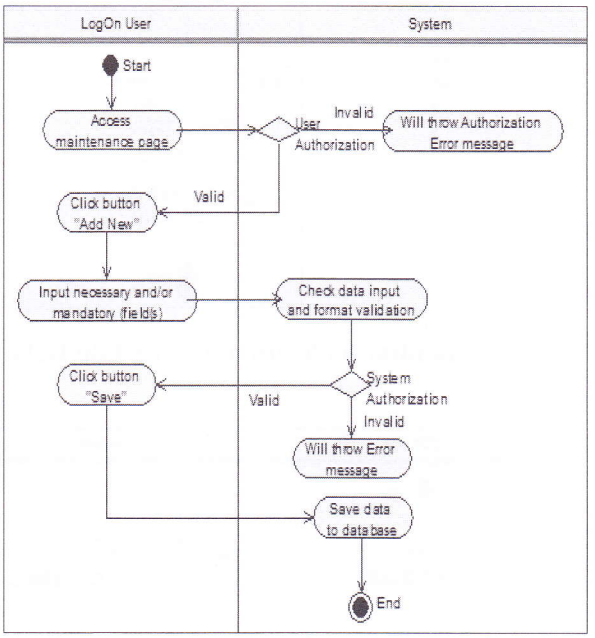
Rancangan diagram *Use Case* pada modul *Master Management* digunakan untuk menggambarkan fungsi-fungsi mengelola data oleh pengguna dalam submodul: *Departement, Location, Workgroup, Company, Category, Sub Category, Status, Intervention Level, Activity Observed*, terlihat pada Gambar 5.



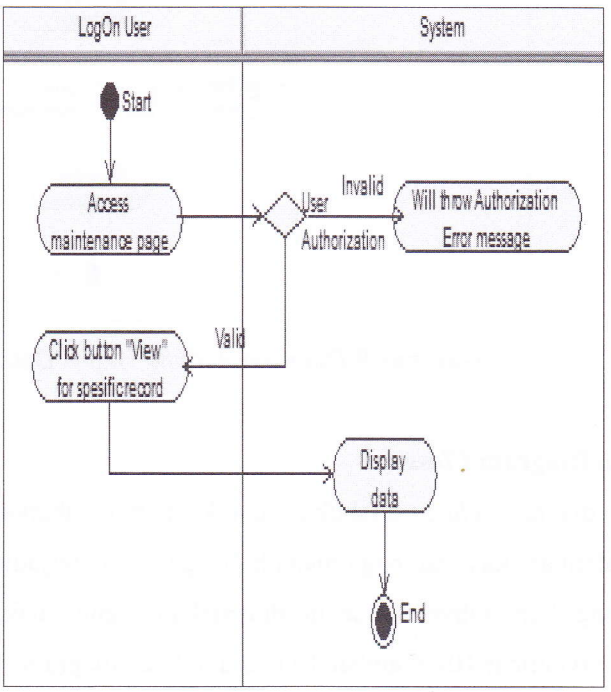
Gambar 5 Diagram *Use Case Master Management*

3.3.3. Rancangan Diagram *Activity*

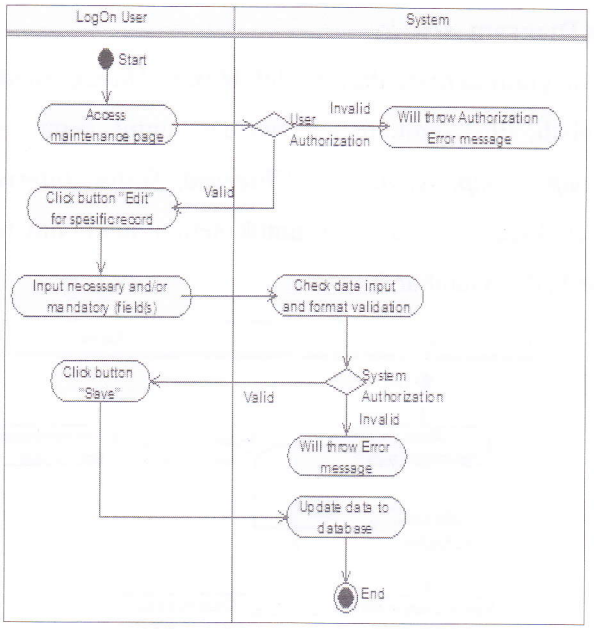
Rancangan diagram *activity* dari modul *Master Management* hanya terbagi dalam 4 aktifitas yaitu tambah, lihat, edit, dan hapus data untuk semua submodul: *Department, Location, Workgroup, Company, Activity Observed, Status, Intervention Level, Category, Sub Category*. Oleh karena itu, *activity* untuk setiap submodul terlihat pada Gambar 6, Gambar 7, Gambar 8, dan Gambar 9.



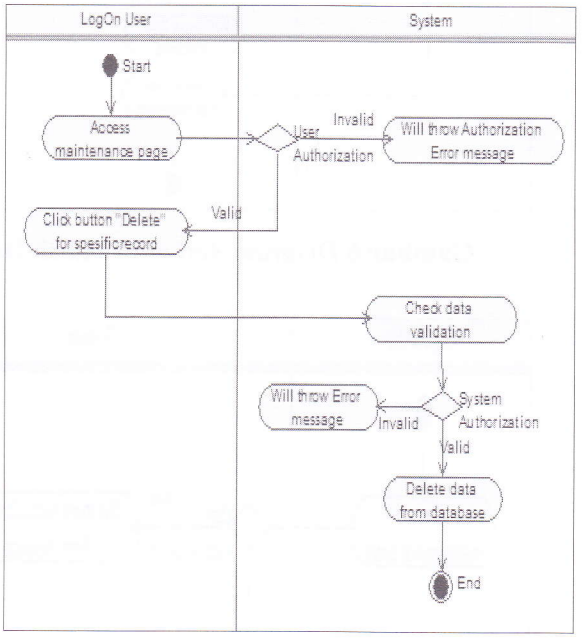
Gambar 6 *Diagram Activity* Tambah Data



Gambar 7 *Diagram Activity* Lihat Data



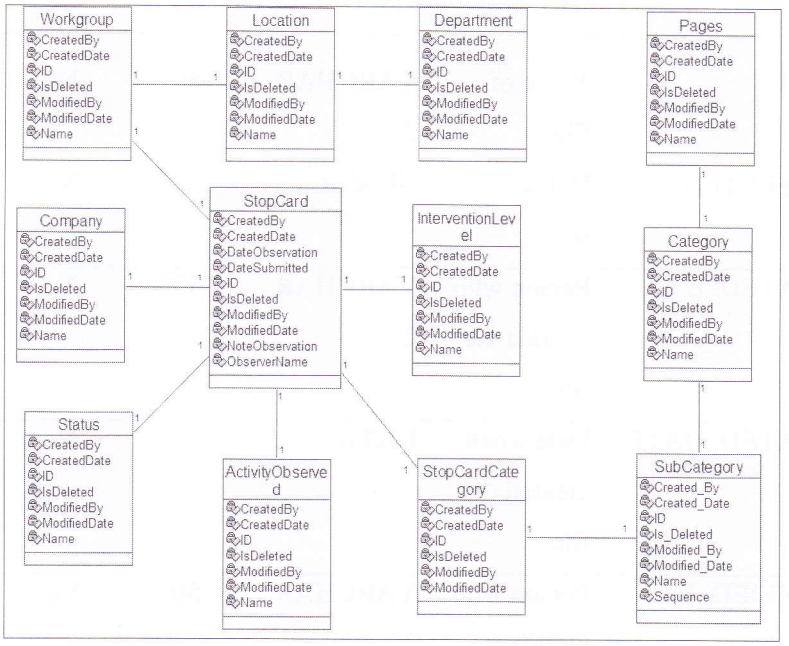
Gambar 8 *Diagram Activity* Edit Data



Gambar 9 *Diagram Activity* Hapus Data

3.3.4. Rancangan Diagram *Class*

Rancangan diagram *class* dilakukan untuk menggambarkan obyek-obyek yang menjadi bagian dalam aplikasi dan bagaimana hubungan yang terjadi antar obyek yang satu dengan obyek yang lain. Obyek tersebut dinyatakan menggunakan pendekatan *object relational mapping* (Gambar 10). Gambar 10 menunjukkan diagram *class eStop Card*.



Gambar 10 Diagram *Class eStop Card*

Diagram pada Gambar 10 menunjukkan representasi dari aplikasi yang akan dibuat. Dan dalam rancangan basis data aplikasi e*Stop Card* berbasis *web* terdapat tabel-tabel yang digunakan untuk menyimpan data dari modul *Master Management*. Tabel-tabel tersebut adalah MS_DEPARTMENT, MS_LOCATION, MS_WORKGROUP, MS_PAGES, MS_CATEGORY, MS_SUB_CATEGORY, TRX_STOP_CARD_CATEGORY, TRX_STOP_CARD, MS_ACTIVITY_OBSERVED, MS_STATUS, MS_COMPANY, MS_INTERVENTION_LEVEL. Tabel 1 merupakan salah satu tabel pada basis data, yaitu tabel MS_DEPARTMENT.

Tabel 1 MS_DEPARTMENT

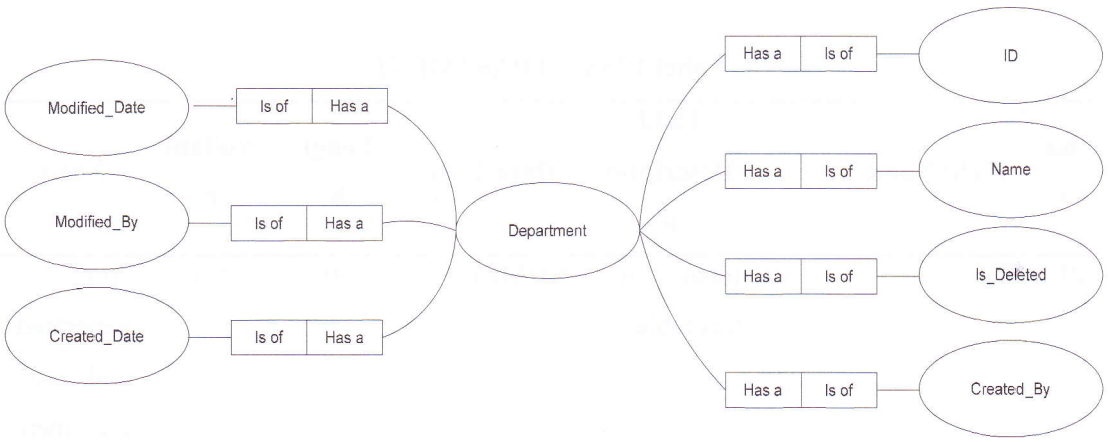
Key	Field Name	Field Description	Data Type	Length	Nullable	Notes
PK	ID	Identity of this table	NUMBER	20	No	Auto generated and auto increment Primary Key of this table

Tabel 1 (lanjutan) MS_DEPARTMENT

NAME	Name of file	VARCHAR 2	50	No
IS_DELETED	Delete status	NUMBER	1	No
CREATED_BY	Person who created the file	VARCHAR 2	50	No
CREATED_DATE	Date when created the file	DATE		No
MODIFIED_BY	Person who modified the file	VARCHAR 2	50	Yes
MODIFIED_DATE	Date when modified the file	DATE		Yes

3.3.5. Rancangan Diagram *Object Relational Mapping* (ORM)

Rancangan diagram ORM modul *Master Management* dilakukan untuk menggambarkan atribut-atribut suatu obyek. Rancangan diagram ORM ini dapat dilihat pada salah satu submodul *Master Management* yaitu *Master Department*.



Gambar 11 Diagram ORM *Master Department*

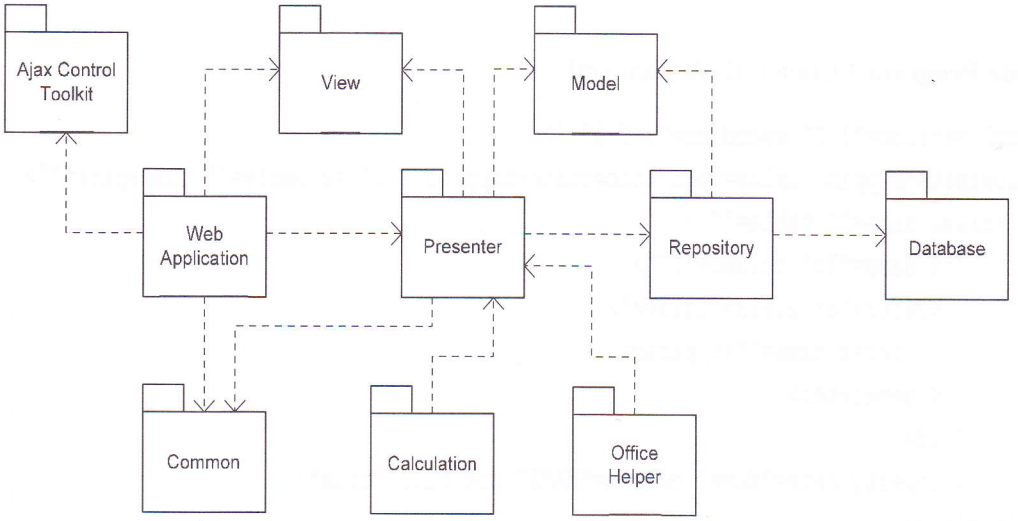
Gambar 11, menunjukkan diagram ORM dari *Master Department* yang memiliki atribut-atribut yaitu *id*, *name*, *is_deleted*, *created_by*, *modified_date*, *modified_by*, *created_date*.

Mapping atribut-atribut tersebut, dibuat dari dari suatu berkas *xml* yang kemudian akan disimpan pada tabel dalam basis data.

3.4. Implementasi Konsep MVP dan ORM

Sistem ini dikembangkan dengan menggunakan teknik *Model View Presenter* (MVP), aturan bisnis berjalan sesuai kondisi yang ditentukan oleh pihak perusahaan. Untuk implementasi MVP dan ORM diambil contoh pada modul *Master Management* dimana telah terdapat aturan bisnis sesuai analisis kebutuhan.

Pada MVP, *view* menunjukkan representasi yang akan ditunjukkan/ditampilkan kepada pengguna, *model* sebagai definisi struktur *object*, *repository* menyediakan pemodelan data, menerima masukan dan fungsi-fungsi lainnya untuk pengaturan akses pada basis data, dan pada *presenter* terdiri dari logika untuk melakukan kontrol terhadap *view* dan *model*. Untuk melihat komponen secara lengkap yang digunakan dalam implementasi MVP, dapat dilihat pada Gambar 13.



Gambar 12 Ilustrasi Komponen dalam Arsitektur Aplikasi

Gambar 12 memperlihatkan ilustrasi hubungan antar komponen. *Web Application* merupakan *Graphical User Interface* (GUI) yang dibuat untuk tampilan ke pengguna, *View* merupakan *class* yang menampung nilai dan untuk menghubungkan GUI dengan *Presenter*. *Presenter* merupakan bagian yang digunakan untuk memasukkan *business logic*, perpindahan data dari *View* ke *Model* atau sebaliknya. *Model* merupakan *class* yang digunakan untuk menampung data dari *Presenter* dan *Repository*. *Repository* merupakan bagian yang digunakan untuk *logical process* basis data dalam melakukan tambah, lihat, ubah, dan hapus data. *Database* merupakan tempat penyimpanan data. *Ajax Control Toolkit* merupakan *class* yang digunakan untuk operasi AJAX pada GUI. *Common* merupakan

class yang umum dipakai sebagai penunjang dalam membuat aplikasi. *Office Helper* merupakan *class* yang dibuat untuk Interop (cara mengakses *Excel*). Selain itu, implementasi konsep terhadap penggunaan ORM ketika ada definisi data berupa *person* yang memiliki atribut *id* dan *name*. Tanpa menggunakan ORM, untuk menyimpan obyek tersebut ke dalam basis data dengan menjalankan perintah SQL:

INSERT INTO Person (id, name) VALUES ('1','Ando');

Dengan menggunakan ORM, perintah tersebut akan digantikan dengan fungsi misalkan save() (bergantung pada *library* ORM yang disediakan), sehingga pemanggilnya adalah:

```
Person prs = new Person('1','Ando');
prs.insert();
```

ORM yang akan memetakan fungsi insert() sama halnya dengan perintah SQL "*INSERT*". *Library* yang akan digunakan adalah NHibernate pada .NET *platform*. Pemetaan entitas ke dalam tabel menggunakan ORM NHibernate juga dibantu dengan berkas *xml* yang berfungsi untuk pemetaan atribut-atribut dari obyek tersebut ke dalam basis data. Contoh penggunaan *hbm.xml* terlihat pada Kode Program 1.

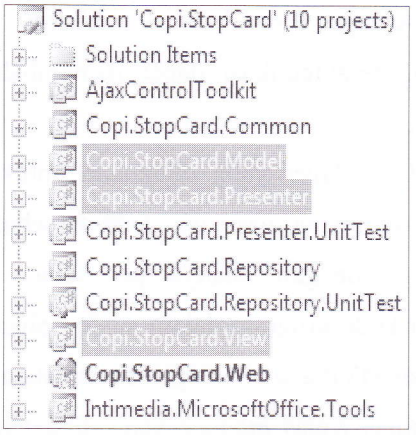
Kode Program 1 Contoh Kode hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="" namespace="">
  <class name="" table="" >
    <id name="Id" column="ID">
      <generator class="native">
        <param name=""></param>
      </generator>
    </id>
    <property name="Name" column="NAME" not-null="true"/>
  </class>
</hibernate-mapping>
```

4. IMPLEMENTASI DAN ANALISIS HASIL

Hasil dan Pembahasan Implementasi *Model View Presenter* dan *Object Relational Mapping*

Saat ini konsep MVP sudah diimplementasikan ke dalam sistem. Secara fisik program, komponen-komponen yang digunakan, terlihat pada Gambar 13.



Gambar 13 Paket Komponen Implementasi MVP

Bagian-bagian dari implementasi *Model View Presenter* itu dipisahkan ke dalam paket-paket yang berbeda yaitu *Model*, *View*, dan *Presenter*, agar dapat terlihat dengan jelas dimana posisi serta tugas dari masing-masing komponen tersebut. Pemisahan dari masing-masing komponen tersebut dimaksudkan agar pengerjaan aplikasi dibagi-bagi ke dalam beberapa komponen/bagian sehingga kedepan memudahkan pengembang aplikasi dalam melakukan *maintain*. Selain itu ada juga penambahan komponen yang dapat menunjang kinerja dari MVP yaitu *AjaxControlToolkit*, *Common*, *Presenter.UnitTest*, *Repository*, *Repository.UnitTest*, *Web*, dan *OfficeHelper*. Dalam aplikasi ini, untuk pengaturan *login* pengguna ditentukan berdasarkan pengguna yang terautentikasi oleh *service* dari perusahaan dalam arti hanya pengguna yang tercatat dalam basis data yang bisa masuk dan menggunakan aplikasi ini. Kode Program 2 merupakan perintah yang berfungsi mengatur *service* autentikasi pengguna ke sistem utama perusahaan.

Kode Program 2 Autentikasi Pengguna dalam *System Admin*

```
public static bool IsUserInSystemAdminGroup()
{
    UserService userService = new UserService();
    string userId = HttpContext.Current.User.Identity.Name;
    if (userId.Contains("\\"))
    {
        userId = userId.Split(Convert.ToChar("\\")).Last();
    }
    string authenticateGroup = ConfigurationSettings.AppSettings["SystemAdminGroup"];
    string domain = ConfigurationSettings.AppSettings["Domain"];

    userService.Credentials = System.Net.CredentialCache.DefaultCredentials;
    bool isAuthenticate = userService.IsUserInGroup(userId, authenticateGroup, domain);

    return isAuthenticate;
}
```

Dalam tahap pengembangan aplikasi, untuk tiap kali *running* diasumsikan pengguna telah *login* dengan menetapkan nilai *true* pada kontrol *login*, seperti kode program COPSiteNavigationHeader1.EnableLoginStatus=true; agar tidak perlu dilakukan *login*

pengguna secara berulang-ulang. Pada halaman *Master Management* yaitu modul *Department*, pengguna yang terautentikasi dapat menambah, melihat, mengubah, dan menghapus data *Department*.

Terdapat pemisahan *logic layer* yang dilakukan untuk mengatur bagaimana *user interface* berperilaku dan bagaimana *user interface* tersebut berinteraksi dengan *core application*. Selain itu juga dengan adanya pemisahan *layer* dapat memudahkan *programmer* dalam merawat kode program dalam suatu aplikasi. Langkah awal membuat aplikasi dengan menggunakan MVP adalah dengan menuliskan *test class*, dimana *test class* ini adalah suatu kelas *interface*. Kelas *inteface* yang dibuat merupakan persyaratan awal untuk suatu kode dapat dilakukan *compile* dengan benar. Misalnya dalam kasus ini bagaimana menuliskan *test class* dalam melakukan implementasi MVP pada submodul *Department*. Kode Program 3 merupakan kelas *interface* dari *IMasterDepartmentView*.

Kode Program 3 *IMasterDepartmentView*

```
public interface IMasterDepartmentView : IMainView
{
    string Name { set; get; }
}
```

Implementasi *Interface IMasterDepartmentView* pada Kode Program 3 merupakan turunan dari *IMainView*, dengan pemberian dan pengembalian suatu nilai dari variabel *Name*. Dengan melakukan *implement* pada *IMasterDepartmentView*, maka pada *web page* memiliki suatu tanggung jawab untuk *implement* suatu *method* atau *properties* yang telah didefinisikan dalam *interface* tersebut. Terdapat dua *properties* pada *interface IMasterDepartmentView* dan itu adalah *set* dan *get*, yang dapat memberikan dan mengembalikan suatu implementasi dari variabel *string Name*. Selanjutnya perlu dibuat *ASPX* untuk halaman *Master Department*, yaitu *DepartmentMain.aspx* pada Kode Program 4.

Kode Program 4 *DepartmentMain.aspx*

```
<asp:Content ID="Content" ContentPlaceHolderID="CphMain" runat="server">
    <uc1:COPMessageBox ID="COPMessageBox" runat="server" Display="Normal" Template="Panel" />
    <br />
    <uc2:TextBoxRow ID="TbrName" runat="server" Caption="Name" IsMandatory="true" />
    <br />
    <uc3:SubmitActionButtons ID="MabAction" runat="server"
                    OnSaveClick="ABMain_SaveClick"
                    OnClearClick="ABMain_ClearClick"
                    OnBackClick="ABMain_BackClick"
                    OnDeleteClick="ABMain_DeleteClick"
                    OnResetClick="ABMain_ResetClick" />
</asp:Content>
```

Berikutnya perlu dibuat *MasterDepartmentPresenter*, dimana pada kelas ini terdapat beberapa hal penting yaitu *constructor*, *method*, dan *validation*. Instansiasi aktual dari suatu *presenter* merupakan tempat yang menggantikan *code behind* untuk *web page*. Walaupun demikian hal ini bisa menjadi masalah jika tidak ada suatu *reference* kepada *service layer*. Untuk menyelesaikan masalah tersebut, perlu ditambahkan suatu *overloaded method* pada kelas *MasterDepartmentPresenter* dengan perintah pada Kode Program 5.

Kode Program 5 *Overloaded Method MasterDepartmentPresenter*

```
public MasterDepartmentPresenter()
    : this(new DepartmentRepository())
{
}
```

Pada konstruktor kelas *MasterDepartmentPresenter* dibuat suatu obyek dari kelas *DepartmentRepository*, yang digunakan untuk menerima *input* parameter dari kelas *MasterDepartmentView*, sehingga pada konstruktor tersebut ditambahkan *method*, yang nantinya akan terdapat *implement* untuk *DepartmentRepository* secara lengkap, terlihat pada Kode Program 6.

Kode Program 6 *Constructor* pada *Presenter*

```
public MasterDepartmentPresenter()
    : this(new DepartmentRepository())
{
}

public MasterDepartmentPresenter(IDepartmentRepository departmentRepository)
{
    _departmentRepository = departmentRepository;

}
```

Selain itu juga, dalam kelas *presenter* ini terdapat *method* yang digunakan sebagai *service implementation*, misalnya salah satu *method* OnSave() yang digunakan untuk menyimpan data ke basis data, dengan perintah pada Kode Program 7.

Kode Program 7 Fungsi OnSave()Pada Kelas *Presenter*

```
public override bool OnSave()
    {
        try
        {
            Department entityToSave = new Department();

            if (View.CurrentPageMode == PageMode.Create){
                if (CreateValidation()){
                    entityToSave = InsertViewToEntity(entityToSave);
                    _departmentRepository.Add(entityToSave);

                    OnClear();
                    View.SuccessMessage = MessageBoxMessages.SaveSuccessful;
                    return true;
                }else{

                    View.ErrorMessage = MessageBoxMessages.SaveUnsuccessful + "<br/>" + MessageBoxMessages.DataIsAlreadyExist;
                }
            }else{
                if (EditValidation()){
                    entityToSave = InsertViewToEntity(entityToSave);
                    _departmentRepository.Update(entityToSave);

                    OnClear();
                    View.SuccessMessage = MessageBoxMessages.EditSuccessful;
                    return true;
                }else{
                    View.ErrorMessage = MessageBoxMessages.SaveUnsuccessful +"<br/>" + MessageBoxMessages.DataIsAlreadyExist;
                }
            }
        }
        catch (CopiCustomException ex)
        {
            View.ErrorMessage = MessageBoxMessages.SaveUnsuccessful + "<br/>" + ex.Message;
        }
        catch (Exception ex)
        {
            throw new CopiCustomException(ex.Message);
        }
        return false;
    }
```

Validation pada kelas *MasterDepartmentPresenter* digunakan untuk melakukan pengecekan data melalui obyek yang ada pada *repository*. *Presenter* membutuhkan 3 (tiga) hal tersebut (*constructor, method, validation*) dalam kerangka performa kerja dari *presenter* itu sendiri.

Komponen *model* digunakan sebagai definisi struktur obyek dengan mendefinisikan atribut-atribut dari obyek tersebut dan yang akan dipetakan ke dalam tabel pada basis data, dengan menggunakan pendekatan hbm.xml dan kelas yang dibuat (Kode Program 9) untuk melakukan *mapping object* ke xml.

Kode Program 8 *Class Department* untuk *Mapping Object* ke hbm.xml

```
public class Department{
        private int _id;
        public virtual int Id{
            get { return _id; }
            set { _id = value; }
        }

        private string _name;
        public virtual string Name{
            get { return _name; }
            set { _name = value; }
        }

        private string _createdBy;
        public virtual string CreatedBy{
            get { return _createdBy; }
            set { _createdBy = value; }
        }

        private DateTime _createdDate;
        public virtual DateTime CreatedDate{
            get { return _createdDate; }
            set { _createdDate = value; }
        }

        private string _modifiedBy;
        public virtual string ModifiedBy{
            get { return _modifiedBy; }
            set { _modifiedBy = value; }
        }

        private DateTime _modifiedDate;
        public virtual DateTime ModifiedDate{
            get { return _modifiedDate; }
            set { _modifiedDate = value; }
        }

        private int _isDeleted;
        public virtual int IsDeleted{
            get { return _isDeleted; }
            set { _isDeleted = value; }
        }
    }
```

Setelah ada kelas yang tersedia pada Kode Program 8, selanjutnya diterapkan suatu *Object Relational Mapping* (ORM) dengan NHibernate menggunakan hbm.xml, ditunjukkan pada Kode Program 9.

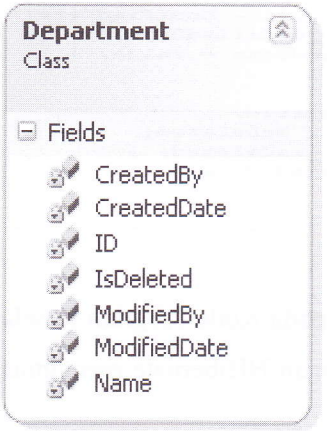
Kode Program 9 ORM *Nhibernate-Mapping* ke Tabel MS_DEPARTMENT

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Copi.StopCard.Model" namespace="Copi.StopCard.Model">
  <class name="Copi.StopCard.Model.Department,Copi.StopCard.Model" table="MS_DEPARTMENT" >
    <id name="Id" column="ID">
      <generator class="native">
        <param name="sequence">SEQ_DEPARTMENT</param>
      </generator>
    </id>

    <property name="Name" column="NAME" not-null="true"/>
    <property name="CreatedBy" column="CREATED_BY" not-null="true"/>
    <property name="CreatedDate" column="CREATED_DATE" not-null="true"/>
    <property name="ModifiedBy" column="MODIFIED_BY" not-null="false"/>
    <property name="ModifiedDate" column="MODIFIED_DATE" not-null="false"/>
    <property name="IsDeleted" column="IS_DELETED" not-null="true"/>
  </class>
</hibernate-mapping>
```

Penggunaan NHibernate tidak hanya melakukan *mapping* dari kelas-kelas pada .NET ke tabel pada basis data (dan juga dari tipe data .NET ke tipe data SQL) tapi juga menyediakan suatu *data query, retrieval facilities,* dan dengan signifikan dapat mengurangi waktu pengembangan pada *data handling* di ADO.NET. Aturan ORM ini dimaksudkan sebagai teknik *mapping* suatu representasi data dari *object model* ke dalam *relational data model* berdasarkan SQL-based schema. Tujuan dari ORM dengan NHibernate ini adalah untuk meringankan tugas dari pengembang perangkat lunak terhadap *data persistence* terkait tugas utama dari seorang pengembang perangkat lunak terhadap sistem yang dikembangkan.

Konsep ORM diimplementasikan dalam komponen *repository* yang berkaitan dengan pendekatan xml. Suatu obyek yang merupakan representasi dari kelas memiliki berbagai macam atribut, terlihat pada Gambar 17.



Gambar 14 *Object Department*

Misalnya saat dilakukan penyimpanan data ke dalam basis data dengan merepresentasikan sebuah obyek yang dapat dilihat pada Kode Program 10.

Kode Program 10 *Inserting* Data ke Tabel pada Kelas *Repository*

```
obj = obj.Add(Expression.InsensitiveLike("Name", "%" + objFilter.FieldValue + "%"));
```

Obyek yang ditambahkan ke dalam basis data tersebut merupakan suatu parameter *input* dari pengguna aplikasi yang dijadikan sebagai sebuah obyek dimana obyek tersebut diimplementasikan dalam sebuah kelas asbtrak *IDepartmentRepository* seperti pada Kode Program 11.

Kode Program 11 Kelas *abstract* untuk *IDepartmentRepository*

```
public abstract Department RetrieveByName(string name);
```

Pada dasarnya, melakukan pemetaan suatu obyek ke dalam basis data sama halnya dengan melakukan pemetaan suatu *collection* dengan menentukan *key* dan *value.*

Dalam melakukan penyesuaian terhadap *unit testing,* MVP mendukung adanya *unit testing*. Kode Program 12, menunjukkan *unit test* yang diletakkan pada komponen *presenter.*

Kode Program 12 *Script Test* untuk *Unit Test* pada *MasterDepartmentPresenterTest.cs*

```
[TestMethod()]
public void OnSaveTest()
{
    MasterDepartmentPresenter target = new MasterDepartmentPresenter();
    bool expected = false;
    bool actual;
    MasterDepartmentViewTest view = new MasterDepartmentViewTest();
    view.CurrentPageMode = 1;
    view.Name = "HSE DEPT.";
    target.View = view;
    actual = target.OnSave();
    Assert.AreEqual(expected, actual);
}
```

Melakukan *unit testing* pada teknik MVP dapat dilakukan pada komponen *presenter* karena seluruh logika bisnis diletakkan pada komponen tersebut, selain itu *presenter* adalah sebagai penengah antara *view* dan *model.* Sehingga cukup dengan menguji kelas-kelas atau fungsi-fungsi yang ada dalam *presenter* maka dapat diketahui apakah proses yang ada sudah sesuai dengan aturan atau belum.

5. KESIMPULAN

Implementasi *Model View Presenter* (MVP) dan *Object Relational Mapping* (ORM) menghasilkan suatu aplikasi *web* yang terstruktur dan mudah untuk dilakukan pemeliharaan, karena pengerjaan aplikasi dibagi-bagi ke dalam beberapa komponen/bagian dengan pemisahan *layer* yang jelas. Selain itu juga, efektif dalam melakukan *unit test*, karena hal ini dapat dilakukan cukup dengan meletakkan suatu *script test* pada komponen *presenter.*

Daftar Pustaka

[1] McCafferty, Billy., 2007., *Model View Presenter with ASP.NET.*, (http://www.codeproject.com diakses tanggal 7 Februari 2012)

[2] PAU, Valentin Corneliu, et al., 2010., *Model View Presenter Design Pattern.*, (http://search.ebscohost.com diakses tanggal 7 Februari 2012)

[3] Saltarello, Andreaa., & Esposito, Dino., 2009., *Microsoft .NET: Architecting Application for the Enterprise (PRO-Developer)*.

[4] Kuaté, Pierre Henri., et al., 2008., *NHibernate in Action.*, Manning Publications.

[5] Microsoft Team, 2012., *Model-View-Presenter Pattern*, Microsoft Library. (http://msdn.microsoft.com diakses tanggal 7 Februari 2012)

[6] Wenzel, Joel., 2011., *MVVM vs MVP vs MVC: The differences explained.* (http://joel.inpointform.net diakses tanggal 10 Februari 2012)

[7] Bauer C, King G., 2007., *Java Persistence with Hibernate.*, United States: Manning.

[8] Pressman, Roger S., 2005., *Software Engineering: A Practioner's Approach*. Sixth Edition., New York: McGraw-Hill Companies, Inc.