iAgile: A Tool for Database Generation Guided by Graphical User Interface

Shaimaa Galal[1] and Ehab Hassanein[2]

[1] Information systems department, Cairo university
Cairo, Egypt

[2] Information systems department, Cairo university
Cairo, Egypt

Abstract

The agile development of the database and software systems is highly productive activity; it reduces time consumed, cost and effort invested in project development, but many agile projects do not apply agile practices to database development and still consider it in a serial manner as heavy-weight methodologies exactly work, while agile methodologies were introduced to overcome the problems experienced with the heavy-weight methodologies. The Enhanced Early Development of Graphical User Interface Practice Framework was introduced to enable performing the database development process in an evolutionary manner. In this article a proposed tool will be presented to help generating the final software product through applying and automating this framework to support agility in both directions of coding and data modeling as well, using such a tool will provide a high level of customer collaboration and help data professionals to work in an agile manner to avoid the problem of having overbuilt systems along with automatically generating portions of the software code based on available modern software architecture models.

Keywords: Agile database development; Agile data modeling; Graphical user interface; Data access Layer generation.

1. Introduction to Agile Database Development

Most data-oriented techniques are serial in nature, requiring the creation of fairly detailed models before implementation is "allowed" to begin. These models are often base line and put under change management control to minimize changes (at the end, this will be actually called a change prevention process) [3]. Agile data modeling allows data professionals to adopt evolutionary approaches to all aspects of their work; examples of those techniques are [3]:

1. **Database refactoring:** evolve an existing database schema by implementing few changes at every time to improve the quality of its design without changing its semantics.
2. **Evolutionary data modeling:** Model the data aspects of a system iteratively and incrementally, just like all other aspects of a system, to ensure that the database schema evolves in step with the application code.
3. **Database regression testing**: Ensure that the database schema actually works.
4. **Configuration management of database artifacts:** Your data models, database tests, test data, and so on are important project artifacts that should be managed just like any other artifact.
5. **Developer sandboxes**: Developers need their own working environments in which they can modify the portion of the system that they are building and get it working before they integrate their work with that of their teammates.

The Enhanced Early Development of Graphical User Interface practice Framework (EEUID framework for short) applied the **evolutionary data modeling** technique on the Early Development of Graphical User Interface practice [8, 9] to provide data professionals with an agile data modeling technique that will have the following advantages:

1. Remove frustration that happens during the development process, as developers ignore data professionals' advice, standards, guidelines, and enterprise models, While developers often do not even know about these people and things in the first place, this problem illustrated in[3].
2. Accelerate the development process by automatically generating code portions that formulate essential parts of the system.
3. Increases the documentation level for the agile methodologies, which is considered a competitive advantage.

In this article, a tool is proposed to put the EEUID Framework in action, which in turn will help to reach an

evolutionary final product that is based on different available modern software architecture models.

2. Introduction to the Enhanced Early Development of Graphical User Interface

Agile methodologies have arisen since the last decade to fulfill the need of developing information systems more quickly in a competitive business environment [7].

There is a framework recently introduced – "the Enhanced Early Development of Graphical User Interface practice framework" [6] – the framework main objective is to generate a final software product through applying agility in both directions of coding and data modeling as well.

The EEUID process works in an evolutionary manner as shown in fig.1.

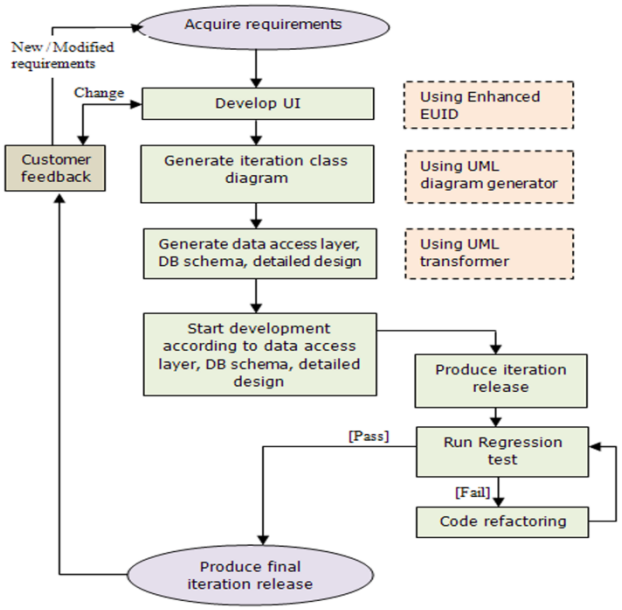

Figure 1: Enhanced EUID process

As shown in fig. 1, through each iteration start, requirements acquired from the customer, then GUI will be developed using the EEUID generator producing three outputs (HTML GUI structure, XML behavior file, XML class diagram description), hence class diagram is generated using UML diagram generator. From this point, the framework automatically generates the data access layer, DB schema and detailed design documents needed for this iteration, then developers can start the development process and produce release for this iteration. For each successive iteration a complete regression test should be applied for the iteration release if it fails, refactoring should be done, if it passes the test, iteration release is presented to the customer for feedback and starting the new iteration.

The framework supports four types of available modern software architecture models; the proposed tool in this article, will apply only one model which is the **GUI layer and object relational access layer above relational DB model** [6].

3. Tool Overview

The iAgile studio is a proposed tool to implement the EEUID framework components; it will help system analysts to automatically generate the database schema, the data access layer and the quality documentation needed for future maintenance and testing purposes.

According to the iAgile Layout and content editor displayed in fig. 2, the visual specification of the project pages is constructed through three sets of controls:

1. The Early Development of Graphical User Interface practice components (EUID components for short), which are components of several types that are used to build main parts of the web page. In the next sub section, further details about these components will be illustrated.
2. Typical web development controls: Examples including Textboxes, Comboboxes and images. These are used for further addition\modification of the web page controls. Meanwhile, when a user drags and drop any of these, it must be attached to a particular page component, then events can be added to different types of controls. The tool offers a set of specified predefined actions such as saving, searching, adding, deleting data or moving to another page as an initial set of most frequently used actions. For further actions, the system analyst should write the Test First examples that will be implemented manually in that iteration development phase.
3. iAgile studio tools: help to build more interactions on the page, such as *simple math calculator* that is used to produce an output in a field from performing simple math calculations on other inputs of the same page, e.g. calculating a person's age from his\her date of birth.

According to fig. 1, at the beginning of each project iteration, the system analyst should start to construct the web pages GUI through the previously mentioned sets of tools; upon completion of this step, the tool will be able to generate the system accumulated class diagram up to this project iteration making use of the generated XML behavior file and XML class diagram description file.

From this point, the system analyst can generate the database schema, the data access layer and the detailed design documents for this iteration according to the chosen software architecture model; consequently a phase of further development should start to complete the missing functionalities using the Test First Examples generated with the web pages GUI and finally proceed to next iteration after considering the customer feedback.

be used for viewing data only or for editing or deleting. The most famous form for editing or deleting records in that component is through adding links for each record in the datagrid.

4. **Navigator component:** There are several actions occurs when a link is clicked. For example, on clicking a link the user can move to another page or make changes on the same page.

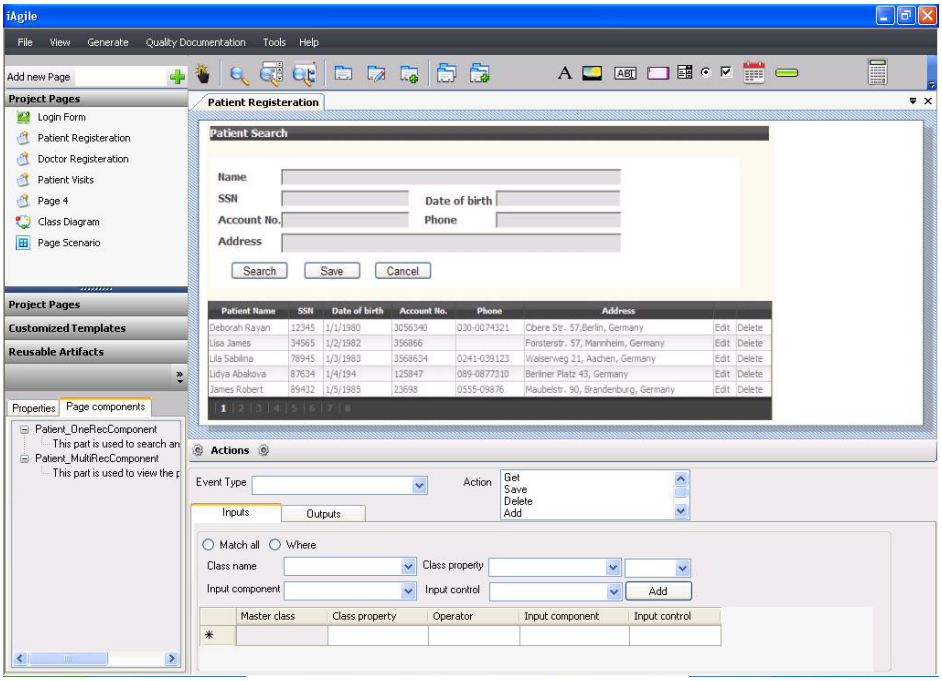

Figure 2: iAgile layout and content editor

3.1 The tool support for the early development of graphical user interface practice components

According to similarities between information systems web applications, the EUID components were proposed as essential constructs for each web page applying the EUID practice, there are four types of these components, which are [8, 9]:

1. **Search component:** It allows users to search for certain data. This may come in different forms, for example, a combobox or a set of editable controls with a search button.
2. **One record component:** It is responsible for adding, editing, or viewing a one record data. One famous form for that component is a set of controls in columnar representation.
3. **Multi record component:** The Multi record component can be referenced to be a datagrid where data is represented in tabular form. It can

iAgile supports these types of components via the idea of that for every screen there is at least a conceptual master class and potential set of dependant classes. The properties of each class will be added from every screen to those classes in order to form the final class definition.

4. iAgile Software Architecture Design

One extreme importance is illustrating the relationship between engineering principles and architectural view; this section will illustrate a number of insights into what iAgile software architecture might be. Fig.3 shows the component diagram of iAgile architecture, which implements the EEUID framework.

The software architecture is composed of:

1. *The Enhanced Early Development of Graphical user Interface package*: designed for designing the pages' UI via constructing each page of the system in the form of one or several components

and it stores the classes' details associated with each screen. This package will generate the pages' user interface in the form of HTML, the XML class description files and the XML behavior files. The XML class description file structure is shown in fig. 4, and the XML behavior file structure is shown in fig. 5.

2. *The UML diagram generator component:* processes the "XML class description files" in order to generate the current system class diagram that is crucial part of the system which will be used mainly to generate the database schema.
3. *The UML transformer package*: will use the class diagram and XML behavior file to generate the final expected outputs of the system.

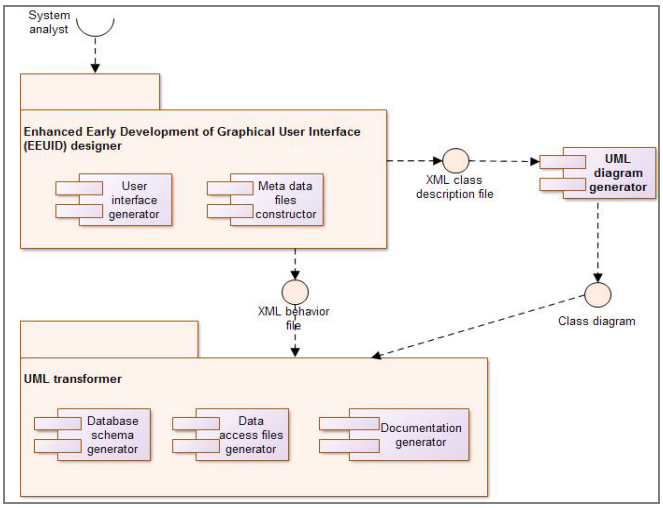

Figure 3: iAgile software architecture's component diagram

```
<?xml version="1.0" encoding="utf-8" ?>
<masterclass name="patients" classidentifier="patientsIID">
    <classproperties>
        <property name="Name" datatype="String"/>
        <property name="SSN" datatype="String"/>
        <property name="Dateofbirth" datatype="DateTime"/>
        <property name="acccountno" datatype="Double"/>
        <property name="Phone" datatype="String"/>
        <property name="Address" datatype="String"/>
    </classproperties>
    <classoperations>
        <operation name="Search">
            <input datatype="String"/>
            <input datatype="String"/>
            <input datatype="DateTime"/>
            <input datatype="Double"/>
            <input datatype="String"/>
            <input datatype="String"/>
            <output datatye="Patients[]"/>
        </operation>

    </classoperations>
    <dependantclasses>
        <class name="PatientVisits"/>
```

Figure 4: EEUID designer's XML class description file structure

```
<?xml version="1.0" encoding="utf-8"?>
<components>
    <component type="searchbutton" id="component1">
        <active_event id="event1" control="button1" active="onclick">
            <action id="action1">
                <desc>
                    when button 1 is clicked the patient data will be
                    fetched from the DB and viewed in the datagrid
                </desc>
                <input> component1.name_1</input>
                <input> component1.SSN_2</input>
                <input> component1.AccountNo_3</input>
                <input> component1.Dateofbirth_4</input>
                <input> component1.Phone_5</input>
                <input> component1.Address_6</input>
                <output>component2.datagrid1</output>
            </action>
        </active_event>
    </component>
    <component type="multirecord" id="component2">
        <active_event>
            <action id="action2" control="linkbutton2" active="onclick">
                <desc>
                    when edit link button is clicked the patient
                    data for the clicked row in this datagrid
                    should be saved.
                </desc>
                <input> component2.datagrid1.row1.PatientName_7</input>
                <input> component2.datagrid1.row1.SSN_8</input>
                <input> component2.datagrid1.row1.Dateofbirth_9</input>
                <input> component2.datagrid1.row1.AccountNo_10</input>
```

Figure 5: EEUID designer's XML behavior file structure

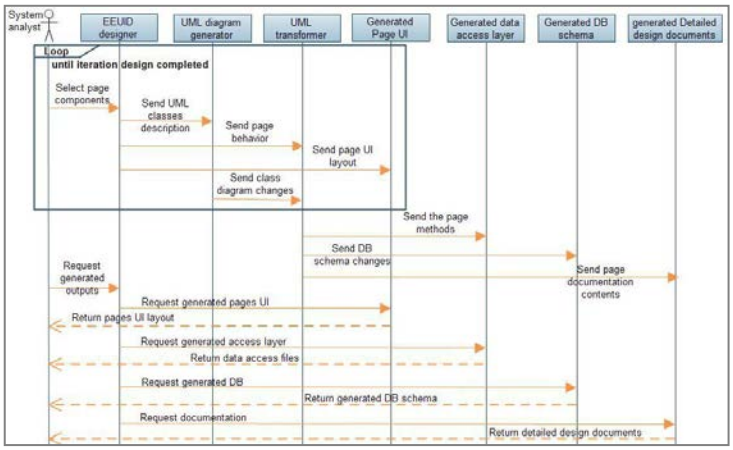

Figure 6: iAgile UML sequence diagram

A UML sequence diagram is shown in fig. 6 to illustrate iAgile process sequence. First, the system analyst starts to design the system pages, each time a component is added in any page, a class description for this page must be provided to indicate whether this class is master or dependant class and whether it have been used previously in other page or not, this is the most critical task to be done as all later steps will depend on this major step. Once the pages for this project iteration are completed, the system will be able to generate the final three outputs. All successive iterations are done the same way and modified outputs for the previous iteration will be generated smoothly.

4. iAgile in Action

This section will demonstrate a simple example of how iAgile will reach the expected final results. Consider a web page for "patients' registration" as shown in fig. 7.

To construct the web page in fig. 7, the page will contain two components: Insert_OneRecord component and Navigation_Edit_Multirecord component.

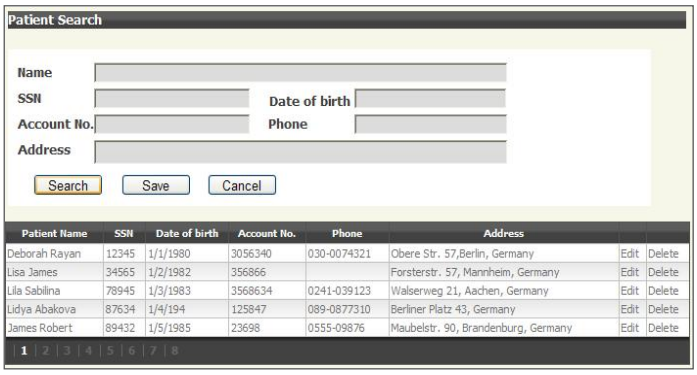

Figure 7: Patients' registration web page

First, the user drags Insert_OneRecord component to the content editor and a screen for master class details will appear. If the data on the web page appears for the first time in the system, then the web page contains a *new master class*, details for the class properties that will appear on the screen should be completed e.g. to add patient name class property, then details should be as follow: Control Type → Text Box, name → Patient Name, Data Type → String, Length → thousand. The rest properties should be filled as shown in fig. 8.

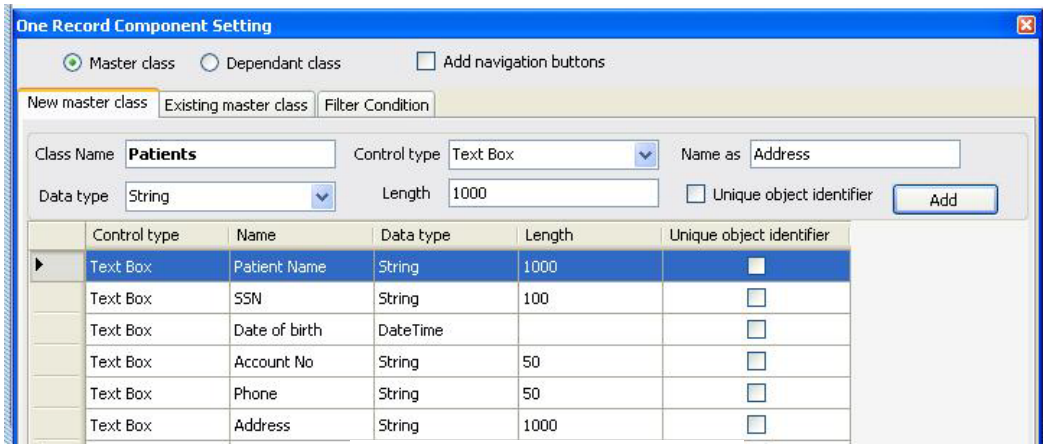

Figure 8: master class details

Second, the user drags Navigation_Edit_Multirecord component to have a data grid in the web page with edit and delete link buttons. Finally to add the extra button for the search criteria and view the results in the data grid, the user should drags a button and sets its action details from the panel in the most bottom of the screen shown on fig.2 to define the action inputs and outputs. After the system analyst finish constructing the web pages for this project iteration, a class diagram will be generated automatically making use of the iAgile metadata files represented in the XML behavior file and the XML class diagram description. iAgile will thereafter generate the three expected outputs, which are database schema, data access layer and detailed design documents, further details illustrated in next sections.

5.1 The tool support for the early development of graphical user interface practice components

Starting from the class diagram classes we need to transform objects into tables in the database. To store an object in a relational database, it should be flatten, i.e. create a data representation for this object. To retrieve the object, data retrieved from the database and then object is created — often referred to as restoring the object — based on that data [2]. There are existing approaches to generate the relational database (RDB) schema from a given class diagram [1, 2], but still it is complicated task as mappings can be done in several ways and choices for each mapping way to use should be defined.

From the previous generated class diagram, we can have RDB schema according to the following mapping choices [6]:

a) *Mapping Meta-data:* Class properties are mapped to table columns:

b) *Inheritance Relationships:* are best mapped by mapping each class to table, due to the simplicity in implementation and understanding.
c) *Relationships between classes:* will be maintained in relational databases through the use of foreign keys.
d) *Object Identifiers:* will be maintained in relational databases through the use of primary keys.

5.2 Data Access Layer

To generate the appropriate classes and files for object relational (OR) data access layer an adequate ORM tool should be chosen first, large number of ready-made tools (O/R mapping tools) are present in the market due to the popularity of using such technologies according to its advantages discussed in [6, 5, 12]. These tools allow mapping between objects and RDB. Examples of these tools are Hibernate/NHibernate, Entity Framework, Open Access, Subsonic, Light Speed, Data Objects.Net and Hera Framework. iAgile here implements the NHibernate tool as it is the most usable ORM tool for .NET and java as well [11].

iAgile automatically generates the object relational access layer files that fit for the relational database schema using the NHibernate engine, the following files will be generated for each web page in the project:

- *Domain object (classes)* – contains class properties getters and setters and main class operation.
- *Hibernate/NHibernate hbm files* – these files represent Mapping Metadata between objects properties and relational table columns along with relationships between classes.

And only one file for the *physical database driver configuration* that contains necessary pieces of information in order to connect to data source. For the web page in fig. 6, two files will be generated which are Patients.cs and Patients.hbm.

5.3 Detailed Design Documents

In Agile, documentation is sparse – often limited to the source code and a set of user stories or UML diagrams. While reducing the amount of documentation can increase productivity, it does come at some risk and cost. Documentation serves as a way to bring new members up to speed. It is useful when transitioning the project to a maintenance team. From a business perspective, documents form the basis for audits assuring proper quality procedures are followed. Documentation serves as a domain knowledge repository and is necessary to retain critical information over time [4, 10].

iAgile builds quality document templates according to the information stored during the web pages GUI building process, one of the templates that can be produced is the detailed design document for the designed web pages. This will help more documentation and give agile methodologies a competitive advantage. Sample of the detailed design document is shown in fig. 9.

Detailed Design Form

Web Page Name: Patients.html	**Date:** 01/05/2011
System Version No.: 1	**Screen Shot:** E:\Project1\ Patients.html
Database Type: SQL server 2005	**Database Name:** EMR
General Page Description: This page for searching patients information to start editing the electronic medical record	

Page Events Description

Control name	Event type	Inputs	Outputs	Mapping to DB	Description	Test first examples
Search button	OnClick	component1.name_1 <<TextBox>> component1.SSN_2 <<TextBox>> component1.AccountNo_3<<TextBox>> component1.Dateofbirth_4<<TextBox>> component1.Phone_5<<TextBox>> component1.Address_6<<TextBox>>	component2.datagrid1	**Inputs:** Table: Patients Columns: Name, SSN, AccountNo, , Dateofbirth, Phone, Address **Outputs:** Table: Patients Columns: Name, SSN, AccountNo, , Dateofbirth, Phone, Address.	When button 1 is clicked the patient data will be fetched from the DB and viewed in the data grid.	

Figure 9: Detailed Design Document

The proposed detailed design document contains two main sections. The first section contains general information as web page name, creation date, iteration number, database engine type, etc. The second section explains page events by stating each control causing event for this page and full details for inputs, outputs, mapping to database, description and associated test first cases [6].

6. Conclusion and Future Work

There are proven practices employed in Agile Methodologies that, when applied under the right circumstances result in lower-risk projects and ultimately better productivity and quality. In this article, a proposed tool – iAgile – was presented to automatically generate a full software product using the EEUID framework that is based on robust software architecture in which developers can refactor later in order to finalize the needed product through several iterations; this will lead to even more productivity and speed up the project delivery time.

iAgile did not stop at the software product generation stage, it also included automatic documentation generation as documentation is a part of the agile software development, but as all agile approaches include a minimum of documentation, it makes documentation the responsibility of the development team to ensure enough

documentation is available for future maintenance and testing.

Future work should include the following items for better development process:

a. Generation of the rest software architecture models as for any project there is not something called “best architecture model”, but there is “adequate architecture model” according to the project circumstances.
b. Generation of different documentation types used for quality and testing purposes that can be formulated from the iAgile studio meta-data files.
c. Supporting the bidirectionality of the Enhanced EUID framework illustrated in [6].
d. It is highly recommended integrating the tool with other automated testing tools as NUnit and JUnit, this will lead the agile family to gain more competitive advantages.
e. Enhancing the GUI generation part by integrating it with additional application of a picture editing tool, e.g. those used for sketching UI widgets. When more sophisticated UI behavior is necessary, embeddable objects such as Adobe Flash need to be generated separately. iAgile will help to model ordinary UIs, but otherwise needs to become part of an interrelated tool-chain.
f. Finally integrating the tool with task management systems will allow easier project planning and task assignments for the project managers, as after finishing each iteration GUI design, the project manager can assess how much work still need to be done and start assigning missing functionalities to developers.

Acknowledgments

I am truly and deeply grateful to my professor Dr.Ehab Ezzat who encouraged and helped me to produce this work and enhanced my personality as well.

References

[1] S. A., “Mapping objects to relational databases - What you need to know and why”, 2000.
[2] S. A., “Agile Database Techniques—Effective Strategies for the Agile Software Developer”, New York: 2003.
[3] S. A., and P. J. S., “Refactoring Databases: Evolutionary Database Design”, Addison-Wesley, 2006.
[4] C. M., and B. S., “The Impact of Agile Methods on Software Project Management”, in Engineering of Computer-Based Systems conference, 2005.
[5] A. D., and V. R., “Object-Relational Mapping Techniques for .Net Framework”, 2004.
[6] Sh. G., E. H., “Applying Agile Methodology on Database Generation Guided by Graphical User Interface”, in international conference on computational intelligence and software engineering, 2011.
[7] J. H., “Agile software development ecosystems”, Boston: 2002, Addison Wesley.
[8] C. L., E. H., and O. H., “Early development of graphical user interface (GUI) in agile methodologies”, in Informatics and Systems Conference, 2010.
[9] C. L., E. H., and O. H., “Early Development of Graphical User Interface (GUI) in Agile Methodologies”, Journal of Computational Methods in Sciences and Engineering, V. 9, No. 1, 2009.
[10] M. L., et al., “Empirical Findings in Agile Methods”, in Universe and First Agile Universe Conference on Extreme Programming and Agile Methods, 2002.
[11] NHibernate 3.1.0, http://www.hibernate.org/, Last date accessed 5.2011.
[12] W. Z.; N. R., “The Real Benefits of Object-Relational DB-Technology”, in British National Conference on Databases: Advances in Databases.