7th International Scientific Conference
Technics and Informatics in Education
Faculty of Technical Sciences, Čačak, Serbia, 25-27th May 2018

Session 2: IT Education and Practice

UDC: 004.42

Teaching Adaptability and Code Reuse of Web Applications with the N-tier Architecture: Case study in VS.NET

Ljubica Kazi[1*], Dragica Radosav[1], Zoltan Kazi[1], Evgeny Cherkashin[23], Madhusudan Bhatt[4], Amar Kansara[5]

[1] University of Novi Sad, Technical faculty "Mihajlo Pupin" Zrenjanin, Serbia

[2] National Research Irkutsk State Technical University, Irkutsk, Russia

[3] Irkutsk State University, Irkutsk, Russia

[4] University of Mumbai, R.D. & S.H. National College, Mumbai, India

[5] Parth Systems, Navsari, India

[*] ljubica.kazi@gmail.com

Abstract: *This paper presents results in exploring research in development of adaptable software and teaching in this field, with special concern on using n-tier web programming as an example. The model of teaching n-tier web programming is based on introducing software components organized in layers (data layer, business logic layer, service layer and presentation layer) and sub-layers within each layer. This model enables adaptability of software to changes, including changes of DBMS, business rules etc. The proposed model is compared with existing industry standard architectures, such as MVC (model-view-controller). The proposed approach is explained with a case study of n-tier ASPX/MSSQL web application.*

Keywords: *adaptable software; n-tier programming, web programming, teaching, MVC*

1. INTRODUCTION

Within higher education of software development, it is of a great importance to study issues that are common within industry environment. One of the most important groups of issues belongs to software change management. Therefore, it is very important to teach students how to create software architectures that provide easier changes of its subsystems. Such changes are frequent in business rules, as well in used data warehouse technologies (i.e. DBMS types, data formats) etc.

Aim of the research presented in this paper is to propose the n-tier architecture suitable for teaching adaptability of software. An example that illustrates the proposed approach is given with the n-tier ASP.NET web application.

This paper is structured as follows: second section explains theoretical background in software change/configuration management, third section describes related work in programming teaching, fourth section explains the proposed model with layers and sub-layers, the responding to changes and comparison to industrial standard MVC, fifth section presents an example that is used within classes of software engineering at University of Novi Sad, Technical faculty "Mihajlo Pupin" Zrenjanin, Serbia. Final section is conclusion.

2. ADAPTATION AND REUSE CONCEPTS

In the phase of software design and construction, there are two very important approaches (SWEBOK [1]):

- Design/Construction for reuse,
- Design/Construction with anticipation of change, i.e. suitable for adaptation.

Concepts of reuse and adaptation are very closely related.

"Construction for reuse creates software that has the potential to be reused in the future for the present project or other projects, taking a broad-based multisystem perspective. IT is desired to encapsulate reusable code fragments into well-structured libraries or components. The tasks related to software construction for reuse during coding include variability implementation with mechanisms such as parameterization, conditional compilation, design patterns etc". [1]

"Most software will change over time, and the anticipation of change drives many aspects of software construction; changes in the environments in which software operates also affect software in diverse ways. Anticipating change helps software engineers build extensible software, which means they can enhance a software product without disrupting the underlying structure."[1]

According to [1], with acceptance of eight software evolution laws, software is considered as constantly evolving, with complexity growing. Changes occur in the software during the core development or maintenance. Maintenance activities could be categorized as: corrective, adaptive, perfective and preventative.

Configuration management is "a discipline of technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements". [1] [2]

"Change request (CR) may be originated by anyone at any point in the software life cycle and it may include the suggested solution and requested priority. The type of change is usually recorded on the Software change request. The formal procedure of managing changes include submitting and recording change requests, evaluating the potential cost and impact of a proposed change and accepting, modifying, deferring or rejecting the proposed change. Changes may be supported by source code version control tools." [1]

Therefore, the concepts of adaptation and reuse being followed at the design phase of the software development provide the basis for the following phases, related to the construction and deployment, as well as to the post-delivery activities. These concepts show potentials in conserving programming experiences for future related projects.

3. RELATED WORK

Evolution of software engineering methods and tools influences teaching in this field, particularly constant improvement in content of corresponding courses. Other important aspect of Software Engineering Teaching is methodology of the teaching process.

In [3], the importance of practical engagement of software engineering students in teamwork within real-world projects is demonstrated with the case study. Usual teaching practice is to include real-world projects in the field of information systems development [4], i.e. enterprise application development [5]. The concept of practical enterprise application development with university teaching in software engineering presented in [5] includes lab assignments in object-oriented programming of n-tier web applications with use of web services. Since dealing with the realistic projects from the real world requires complex set of knowledge and skills, as well as going through the complete software life cycle, to gain appropriate level of details in knowledge and skills requires one project to be split as set of subprojects to be done within set of interrelated subjects. In [6], that approach of having one common project, which the student work on within many subjects (courses) is experimentally tested. Students implemented their practical work within a series of inter-related subject by following the software development life cycle: object-oriented systems analysis and design, data structures, software engineering, web programming, distributed web services and software development, software testing. This way, working at the same project from different aspects within diversity of teaching subjects/courses lead to the completing more complex and more realistic result.

Teaching software engineering within practical work on the real-world projects is closely related to software project management (SPM), where related topics are included within appropriate SPM courses/subjects. Contemporary industrial standard approach to SPM is implementation of agile principles and methods. Experiences in teaching SPM with agile paradigm are described in [7]. One of the agile methodologies is Extreme programming and empirical results in teaching with this methodology are presented in [8]. Other teaching experiences in software development are more focused on technologies in reaching the agility of development by using model-driven approach, such as [9], and even more, – using code generators based on model-driven development [10]. Teaching in software engineering is, in recent years, focused on contents that are related to modern technologies within professional environments, such as cloud computing [11]. Close collaboration of industry and higher education educators and curriculum creators is crucial for the benefit of students to be ready for the knowledge and skills requirements in the professional development environment. The need to improve college programs, in aim to prepare students for professional work, is emphasized in [12]. Particularly important contents are related with IT governance process, knowledge in business domain, ability to capture requirements by proactive asking right questions, quality of design based on experiences, peer or supervisory review of code with suggestions on quality improvements etc. Particularly important is the area of configuration management, changes of code, compliance with existing architecture and documentation [12].

4. THE PROPOSED N-TIER TEACHING MODEL

The proposed approach to teaching the adaptability and code reuse in application development is based on practical work in software development with n-tier object-oriented architecture and service-oriented architecture (similarly to [5], [13]), with applying component

paradigm (separate layers belong to independent components i.e. projects that are coded and build independently from other layers, but that could be integrated with referencing (References – adding link to DLL or Service References – adding alias to Web service URL). This model enables (if needed within change management) replacements of components, easy changing in particular component, teamwork development and improvement of overall software quality, as well as the code reuse.

4.1. Layers and sub-layers

The proposed model of the n-tier software architecture is presented in the Table 1. The first column presents an n-tier software layer and second column is related to sub-layer. The proposed sub-layers implementation is presented in the third column with the example of VS.NET (Visual Studio .NET) web application and other executable components in the architecture.

Table 1. *The proposed model of n-tier web application architecture in VS.NET*

LAYER	SUB-LAYER	VS.NET executable component
Presentation Layer	User interface	ASPX
	Presentation logic	DLL
Services Layer	Web services	ASMX
	Mapping library	DLL
Business logic layer	Business rules class library	DLL
	Business objects class library	DLL
Data layer	Semantic-oriented class library	DLL
	Technology-dependent class library	DLL
Relational database	Stored procedures Views	**DBMS**
	Tables	
Data files	XML	

Executable components from Table 1. are:

- ASPX – ASP.NET Web form file extension
- ASMX – VS.NET web service (SOAP type) file extension
- DLL – Dynamic Link Library, file extension of executable form of class library

Presentation layer consists of:

- User interface, implemented in the example as Web forms application ASPX. Implementing web forms requires studying html + css (which are the essential part of any web page, as well as undelying structure of ASPX web form), with necessary programming part (C#) that enables access, data retrieval and basic instantiation and using presentation logic classes, i.e. invocation of appropriate class methods.
- Presentation logic, encapsulated within a separate component, i.e. Class Library (DLL). It contains the code for data capturing and transformations, validation and preparation for presenting in the user interface. The presentation logic instantiates classes from lower layers, such as services layer, business logic layer and data layer.

Services layer consists of:

- Web services, which are consulted as remote raw data retrieval services (from other data sources, such as XML or other database) or remote business logic services, such as enforcing business rules processing or retrieving constraints (stored in the XML form) needed for local business rules processing.
- Mapping service, encapsulated within Class Library (DLL), provides integration of other layers, i.e. data exchange between classes based on different data models or coding systems.

Business logic layer consists of:

- Business objects, implemented as separate Class Library (DLL), where classes are named by the terms of the business domain actors, entities or business processes and the implementation logic is based on using methods from the data layer classes.
- Business rules are implemented enforced from separate business-rules classes or as business objects methods and their mutual interactions.

Data layer consists of:

- Semantic-oriented class library (DLL) which consists of three types of classes. For each table in relational database there are three classes: a) Single-occurance class (contains private attributes that correspond to table fields from the relational database, as well as corresponding public properties implemented with set/get methods), b) List class, which contains typed list of "single-occurance" classes, c) DB class, which contains methods for basic CRUD (Create, Read, Update, Delete) operations upon appropriate table from the database. These methods could be implemented with using diversity of methods (using SQL client classes and methods with SQL queries or stored procedures or using technology-dependent class library).
- Technology-dependent class library (DLL) which is semantic-independent. The technology dependence occurs because of using standard classes (from appropriate DB API) for the connection to the database and executing active SQL queries (directly or with using stored procedures). The technology dependent classes consist of the database connection, table and transaction classes,

which are implemented upon standard libraries: SQLClient (for MSSQL server DBMS), OleDB (for MS Access DBMS) or ODBC (other DBMS types). This technology-dependent class library encapsulates the API-dependent source code with general names (of classes, attributes and methods) and therefore provides ease of replacement with other DLL with the same names, but different underlying technology-dependent implementation. Example: General class name is Connection, while DB API for MSSQL DBMS (SQLClient) name for connection is SQLConnection.

In broader definition, data layer could include:

- Relational database, that consists of data tables related with foreign key constraints, as well as stored procedures and views. Stored procedures and views consist of basic CRUD (CreateReadUpdateDelete) SQL queries, which provide faster queries processing. Including stored procedures and views in Semantic-oriented class library methods improves data retreival and processing performances.
- Data files, such as XML, which represent source of raw data or constraints queried from business rules.

4.2. Responding to changes

The proposed model supports changing in:

- The interface type, since presentation logic captures the logic of the application and invoking lower layers
- Coding standards between different modules and applications, with mapping service
- Data responsibilities among diversity of institutions, which is provided by using web services
- Business rules, which is enabled with allocating business rules algorithms within appropriate business objects or separate classes related to business rules enforcing,
- Business rules constraints, provided by web services or externally stored constraints in XML,
- DBMS type, which is enabled by separation of semantics and technology. By changing DBMS, it is sufficient to replace the technology-dependent DLL with other technology-dependent DLL with the same name of class library, classes, attributes and methods. This way, semantic-oriented class library will not require any changes in the source code, but only to be rebuild, because of the change in the referencing DLL.

4.3. Comparison of the proposed model with standard pattern MVC

In contemporary industrial software development, some of the most important principles include:

- Agile development, i.e. quick adaptation of software to user requirements that evolve during the development process, which require an architecture suitable for frequent changes
- Teamwork development, which leads to the need of separation of the software into modules that could be assigned to diversity of teams or team members, with parallel development (which supports decrease in development tim) and specialization of teams/members, which encourages quality of production
- Standard orientation, which requires all team members follow standard procedures, coding conventions, industrial models and design patterns etc. with the outcome in quality of development products and services.

One of the standard software structuring approach include MVC architectural design pattern. The concept of responding to client's request in MVC in ASP.NET (according to [13]) is presented in the Figure 1. Client's htpp request is received by the controller class (derived class from standard System.Web.MVC), which queries model classes for data read, update, insert or delete. After data retrieval or change, the model gives data to the controller, which prepares the necessary output and chooses appropriate view as response (in the form of web page to be presented to client).

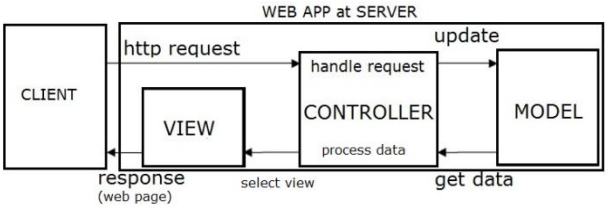

Figure 1. *MVC structure and collaboration of components in responding to http request (according to [13])*

Comparison of the proposed n-tier model and MVC regarding adaptability to changes and reausability of code is presented at Table 2 and explained below:

- Adaptability to changes is provided as in MVC the view is separated from application logic and data. This allows making changes in separate elements. The issue is that the controller captures all the application logic and the model integrates data model semantics and DBMS-dependent technology in CRUD support (example: Entity framework generated classes include both semantics from relational database tables and underlying technology of MS SQL Server incorporated in generated Entity framework code). The proposed n-tier model promotes separation of sub-layers and separation of semantics from the (DBMS-dependent) technological implementation. In this way, the proposed n-

tier approach is more suitable for change implementation and supports adaptability more flexible.

– Reusability of code is provided thanks to MVC approach, where the created model and controller could be used with other type of view. One issue with MVC is that the controller captures all the application logic, which makes it harder to separate, reuse and adapt particular parts. Another issue with MVC is related with very tight dependency of semantic and technology in the model component. In the proposed n-tier model, reusability is supported for all layers and sub-layer, particularly in the data layer, by separation of technology-dependent and semantically independent class libraries, which allows using the same universal technology-dependent class library for any other project. Business rules and business objects classes are reusable for similar software solution, which is particularly imporant for software product lines support. Even the presentation logic, captured within the class library, could be reused with any other user interface.

Table 2. *Comparison of MVC with the proposed n-tier model*

LAYER	SUB-LAYER	MVC
Presentation layer	User interface	View
	Presentation logic	Controller
Service layer	Web service	
	Mapping service	
Business logic layer	Business rules	
	Business objects	Model
Data Layer	Semantic-oriented class library	
	Technology-oriented class library	

5. EXAMPLE OF VS.NET APPLICATION

Illustration of the proposed model is given in this section, with the example constructed within Software engineering classes at University of Novi Sad, Techical faculty „Mihajlo Pupin" Zrenjanin.

Semantics of the example is related to records on teaching staff at the Unviersity school. The web application is developed to provide basic enterprise software functions, such as data entry, tabular presentation with filtering, updating, deleting and printing.

Within implementation of these software functions, students created user-interface project of Web forms type ASPX and parallely created class library projects for appropriate sub-layers. One DLL file of a sub-layer class library project is attached (using References section in VS.NET IDE) in the projects of other sub-layers. Web services (ASMX) are created as separate projects and attached to other sub-layers by using Service Reference section in VS.NET (assigning alias to URL of previously activated Web Service in localhost). Figure 2. shows the user interface of the ASPX application – the page for the entry of data related to the new teaching staff member.

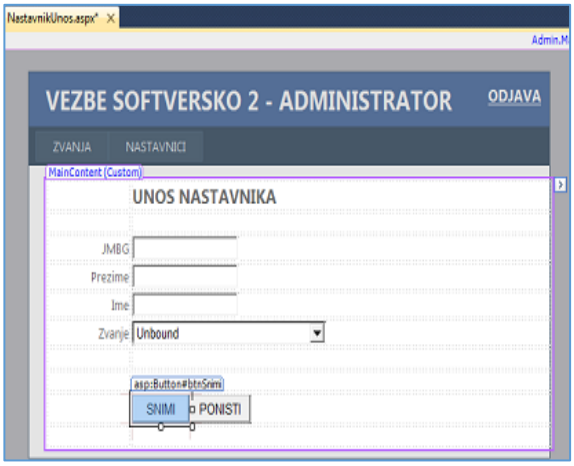

Figure 2. *Teaching staff data entry page of an example user interface*

Drop-down list (combo) gets, at the ASPX page (code presented at Figure 3.), the data from the class, which presents the presentation logic class dedicated to support this page (i.e. form in user interface).

```
private void NapuniCombo()
{
DataSet dsZvanja = new DataSet();
dsZvanja=objFormaNastavnikUnos.DajPodatkeZaCombo(
);
int ukupno = dsZvanja.Tables[0].Rows.Count;
ddlZvanje.Items.Add("Izaberite...");
for (int i = 0; i < ukupno; i++)
{
ddlZvanje.Items.Add(dsZvanja.Tables[0].Rows[i].It
emArray[1].ToString());
}
}
```

Figure 3. *The procedure for filling drop-down list at the page from Figure 2.*

The concept of presentation logic classes organization is that each page at user interface has appropriate class in the presentation logic, that support that page. Figure 4. presents list of classes in the presentation logic that are developed, within classes in Software engineering, for the purpose of this example.

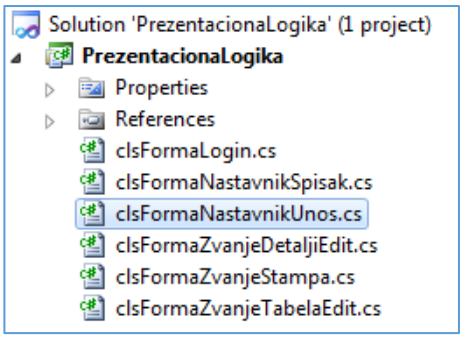

Figure 4. *List of classes within presentation logic*

For the page, presented in Figure 2, the presentation logic class is presented with the code (only attributes and methods signatures are presented) presented at Figure 5. The role of the presentation logic class is demonstrated with this example:

- Getting the data from the user interface in the exact form as they are entered,
- Transformation of data suitable for lower sub-layers,
- Preparation of data for presentation (such as data needed for drop-down list filling)
- Verification of the entered data (checking mandatory fields, uniqueness and invocation of methods from classes from lower layers, such as to check business rules or to start saving of the data).

```
namespace PrezentacionaLogika
{
    public class clsFormaNastavnikUnos
    {
        // atributi
        private string pStringKonekcije;
        private string pJMBG;
        private string pPrezime;
        private string pIme;
        private string pNazivZvanja;

        // property
        public string JMBG...

        public string Prezime...

        public string Ime...

        public string NazivZvanja...

        // konstruktor
        public clsFormaNastavnikUnos(string NoviStringKonekcije)...

        // private metode

        // public metode
        public DataSet DajPodatkeZaCombo()...

        public bool DaLiJeSvePopunjeno()...


        public bool DaLiJeJedinstvenZapis()...

        public bool SnimiPodatke()...

        public bool DaLiSuPodaciUskladjeniSaPoslovnimPravilima()...
    }
}
```

Figure 5. *The code of the presentation logic class*

Uniqueness of the data that were assigned from the user interface to presentation logic attributes is checked within the appropriate method of the presentation logic class, which invokes method from the DB class from the data layer (code presented at Figure 6).

Checking compliance of the data assigned to the presentation logic attributes with the business rules is performed within the method in presentation logic that invokes the method from the business logic layer (code presented at Figure 7).

```
public bool DaLiJeJedinstvenZapis()
{
    bool JedinstvenZapis = false;
    DataSet dsPodaci = new DataSet();
    clsNastavnikDB objNastavnikDB = new clsNastavnikDB(pStringKonekcije);
    dsPodaci =objNastavnikDB.DajNastavnikaPoJMBG(int.Parse(pJMBG));

    if (dsPodaci.Tables[0].Rows.Count == 0)
    {
        JedinstvenZapis = true;
    }
    else
    {
        JedinstvenZapis = false;
    }

    return JedinstvenZapis;

}
```

Figure 6. *Method of presentation logic for data uniqueness checking by invoking DB class method*

```
public bool DaLiSuPodaciUskladjeniSaPoslovnimPravilima()
{
    // POSLOVNO PRAVILO:
    // Na fakultetu ne moze biti zaposleno vise nastavnika u odredjenom
    // zvanju nego sto je dozvoljeno maksimalnim brojem prema Sistematizaciji
    // radnih mesta.
    bool UskladjeniPodaci = false;

    clsPoslovnaPravila objPoslovnaPravila = new clsPoslovnaPravila(pStringKonekcije);

    //izracunavanje ID zvanja
    clsZvanjeDB objZvanjeDB = new clsZvanjeDB(pStringKonekcije);
    string IDZvanja = objZvanjeDB.DajSifruZvanjaPoNazivu(pNazivZvanja);

    UskladjeniPodaci = objPoslovnaPravila.DaLiImaMestaZaZaposljavanje(IDZvanja);

    return UskladjeniPodaci;
}
```

Figure 7. *Method of presentation logic for business rules checking by invoking class method from the business logic class library*

Saving data, that were assigned from the user interface to presentation logic attributes, is performed with appropriate method from the presentation logic class, which invokes the appropriate method from the DB class from the data layer (code presented at Figure 8).

```
public bool SnimiPodatke()
{
    bool uspehSnimanja=false;

    clsNastavnikDB objNastavnikDB = new clsNastavnikDB(pStringKonekcije);

    clsNastavnik objNoviNastavnik = new clsNastavnik();
    objNoviNastavnik.JMBG = int.Parse(pJMBG);
    objNoviNastavnik.Prezime = pPrezime;
    objNoviNastavnik.Ime = pIme;

    clsZvanje objZvanje = new clsZvanje();

    clsZvanjeDB objZvanjeDB = new clsZvanjeDB(pStringKonekcije);
    objZvanje.Sifra = objZvanjeDB.DajSifruZvanjaPoNazivu(pNazivZvanja);
    objZvanje.Naziv = pNazivZvanja;

    objNoviNastavnik.Zvanje  = objZvanje;

    uspehSnimanja = objNastavnikDB.SnimiNovogNastavnika(objNoviNastavnik);


    return uspehSnimanja;
}
```

Figure 8. *Method of presentation logic for saving data by invoking DB class method*

The business rule that is checked is in this example related to the constraint on the maximal number of employees for certain teaching position, formulated in the form of

IF (condition) THEN (action).

IF (number of existing teaching staff is lower than maximal allowed number of teaching staff) THEN (hire a new teaching staff member).

Business rules application algorithm is presented at the Figure 9. and appropriate code is presented at Figure 10.

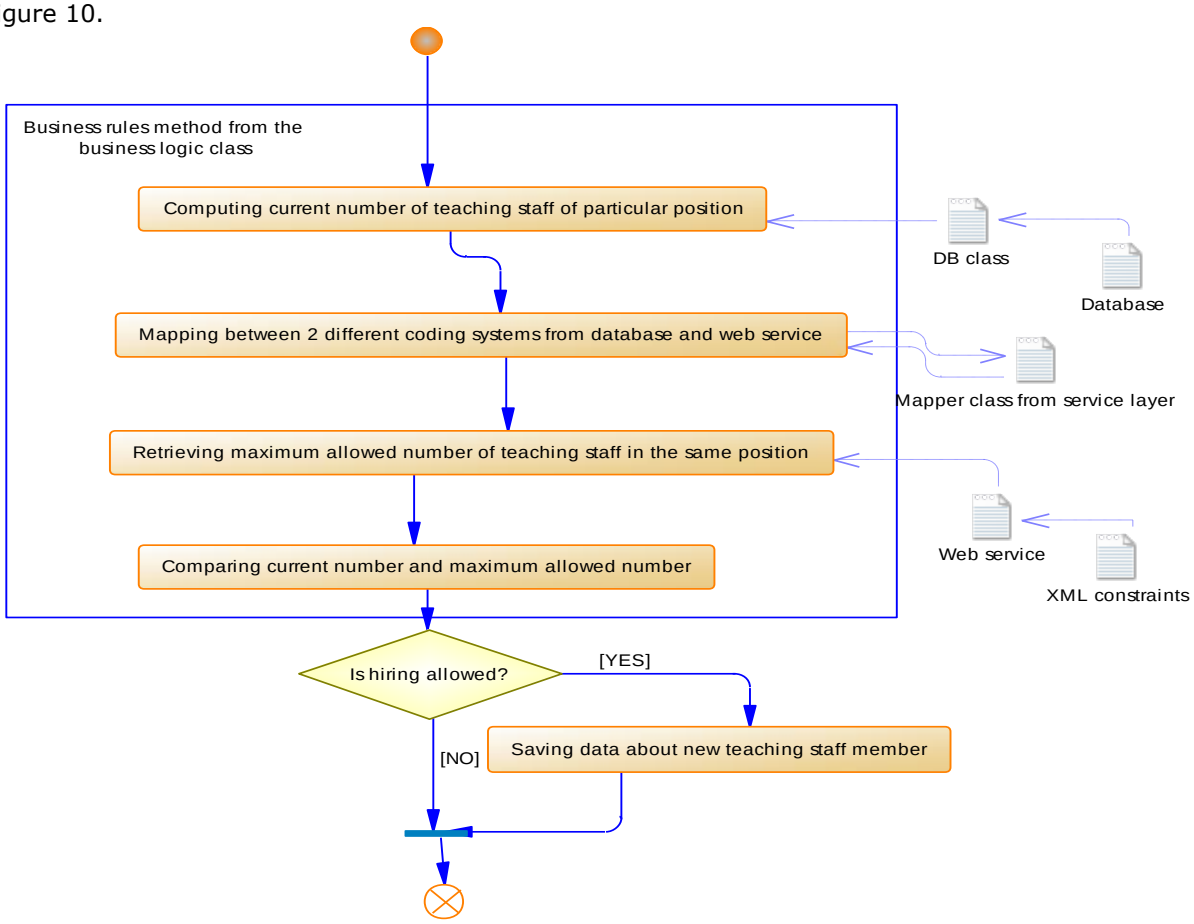

Figure 9. *Activity diagram of the business rule with invocation of methods from other layers*

```
public bool DaLiImaMestaZaZaposljavanje(string IdZvanjeIzBazePodataka)
{
    bool imaMesta = false;
    // ############################################################
    // 1. IZRACUNAVANJE TRENUTNOG BROJA NASTAVNIKA U BAZI PODATAKA
    int UkupnoNastavnika = 0;
    clsNastavnikDB objNastavnikDB = new clsNastavnikDB(pStringKonekcije);
    UkupnoNastavnika = objNastavnikDB.DajUkupnoNastavnikaZaZvanje(IdZvanjeIzBazePodataka);
    // ############################################################
    // 2. MAPIRANJE SLOJEVA - uskladjivanje ID Zvanja iz raznih delova programa
    // Web servis ima d - docenta, baza podataka ima 1 - docent
    string IdZvanjeWS = "";
    clsMaper objMaper = new clsMaper(pStringKonekcije);
    IdZvanjeWS = objMaper.DajSifruZvanjaZaWebServis(IdZvanjeIzBazePodataka);
    // ############################################################
    // 3. IZDVAJANJE MAX BROJA NASTAVNIKA ZA ODG ZVANJE
    WSKadrovskiPodaci.Service1 objOgranicenja = new WSKadrovskiPodaci.Service1();
    int MaxBrojNastavnika = objOgranicenja.DajMaxBrojNastavnika(IdZvanjeWS);
    // ############################################################
    // 4. UPOREDJIVANJE TRENUTNOG BROJA I MAX BROJA NASTAVNIKA
    if (UkupnoNastavnika < MaxBrojNastavnika)
    {
        imaMesta = true;
    }
    else
    {
        imaMesta = false;
    }
    return imaMesta;
}
```

Figure 10. *Method for business rules checking within the business logic class*

6. CONCLUSION

Constant change in the information technologies requires adaptation of the teaching contents and improvement of teaching methodology. In the software engineering teaching, contemporary technologies and industry practices lead university teaching toward agile development, teamwork, programming of distributed software systems etc.

Aim of this paper is particularly focused on the teaching issues in development of software that is suitable for adaptation to changes. Design and construct reusable code is closely related to the suitability for code change and the teaching model that is proposed in this paper addresses both issues.

This paper proposes an n-tier model to be used in teaching code adaptation and reusability of code, while comparing the proposed model with the MVC, the industrial standard approach. As an illustration of the proposed model, an example of n-tier VS.NET web application is presented with user interface page, code segments and an example of a business rule compliance checking algorithm. The presented example is used at Software engineering classes at bachelor (undergraduate) level of study in Information Technology Engineering at University of Novi Sad, Technical faculty "Mihajlo Pupin" Zrenjanin, Serbia.

The presented n-tier model is not a complete model of enterprise-oriented professional software, but is suitable for the undergraduate study in enterprise-oriented software development. More complex teaching aspects of professional software engineering should include workflow and business rules management systems, with automated reasoning over externally stored rules written in formal language and data obtained from the database and the user interface. Such advanced (non-covered in this model) topics are included at master-level of studies at the University of Novi Sad, Technical faculty "Mihajlo Pupin" Zrenjanin, Serbia.

Comparison of the proposed model is made regarding teaching practice in India and Russia and the proposed model is recognized as similar that is taught in these countries. The proposed model is also compared with industry experiences in India and it is concluded that it enables students to understand software component paradigm and components integration, as well as techniques for developing a reusable and adaptable software in a way that prepares students for the industry practice, by making students understand basic structures and principles and empowering them to understand industrial frameworks.

REFERENCES

[1] IEEE (2014). *SWEBOK, Guide to the Software Engineering Body of Knowledge, Version 3.0*

[2] ISO/IEEE (2010). *ISO/IEC/IEEE 24765:2010 Systems and Software Engineering—Vocabulary, ISO/ IEC/IEEE.*

[3] Gnatz, M., Kof, L., Prilmeier, F., Seifert, T. (2003). A Practical Approach of Teaching Software Engineering, *Proceedings of 16th Conference on Software Engineering Education and Training*, IEEE Computer Society, ISBN 0-7695-1869-9, pp 120-128.

[4] Ivanovic, M., Todoric-Vukasin, D., Budimac, Z. (2009). Information system topics for studies in software engineering, *Proceedings of the International Conference on Computer Systems and Technologies – CompSysTech*, ACM, 2009, pp IV.15-1, IV.15.6

[5] Withhinghill, D., Lutes, K. (2011). Teaching Enterprise Application Development: Strategies and Challenges, *Proceedings of the SIGITE Conference*, *ACM*, October 20-22, 2011, West Point, New York, USA

[6] Iyengar, S.R. (2009). Teaching Enterprise Software Development in Undergraduate Curriculum, *Proceedings of the SIGITE Conference, ACM*, October 22-24, 2009, Fairfax, Virginia, USA

[7] Fitsilis, P., Lekatos, A. (2017). Teaching software project management using agile paradigm, *Proceedings of 21st Pan-Hellenic Conference on Informatics*, Larissa, Greece, September 2017 (PCI'17).

[8] Bergin, J., Caristi, J., Dubinsky, Y., Hazzan, O., Wiliams, L. (2004). Teaching Software Development Methods: The Case of Extreme Programming, *Proceedings of SIGCSE'04, ACM*, March 3-7, 2004, Norfolk, Virginia

[9] Kuzniarz, L., Martins, L.E.G. (2016). Teaching Model-Driven Software Development: A Pilot Study, *ITiCSE'16 Working Group Reports*, ACM, July 09-13, 2016, Arequipa, Peru

[10] Schmidt, A., Kimmig, D., Bittner K., Dickerhof, M. (2014). Teaching Model-Driven Software Development: Revealing the "Great Miracle" of Code Generation to Students, *Proceedings of the Sixteenth Australasian Computing Education Conference (ACE2014),* Auckland, New Zealand.

[11] Trinta, F.A.M., Santos, E. (2017). Teaching Software Development for the Cloud: An Experience Report, *Proceedings of the SBES'17*, September 20-22, 2017, Fortaleza, CE, Brazil

[12] Vandever, K. (2008). Teaching the Business of Software Development, *SIGCSE Bulletin*, Vol. 40. No 2, 2008.

[13] Microsoft. Application Architecture Guide 2nd edition. https://msdn.microsoft.com/en-us/library/ee658109.aspx [visited: April 10, 2018].