პროგრამული ინჟინერიის ტექნოლოგიის გაფართოებული გამოყენება მოდელებით მართვადი არქიტექტურით

ეკატერინე თურქია, დავით ჯიბუტი, სოფიო სტომადოვა
საქართველოს ტექნიკური უნივერსიტეტი

რეზიუმე

განხილულია პროგრამული ინჟინერიის ტექნოლოგიის გაფართოებული გამოყენება ორგანიზაციის ბიზნესპროცესების ფორმალიზებული აღწერიდან პროგრამული უზრუნველყოფის დაპროექტებამდე. მოცემულია პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის ეტაპებზე, პროგრამული ინჟინერიის ინსტრუმენტების გამოყენების დონეები და ამ დონეების ლოგიკური ურთიერთდაკავშირება მოდელებით მართვადი ტექნოლოგიით. პრაქტიკული მაგალითების სახით შემოთავაზებულია ბიზნესპროცესების დიაგრამიდან UML UseCase ფორმირება, შემდეგ მისგან აქტიურობის დიაგრამის ავტოგენერაცია, კლასების დიაგრამაში მეთოდის აღწერა აქტიურობის დიაგრამით და კოდის გენერაცია მეთოდის პირობითი და ციკლური ოპერატორების თანხლებით. წარმოდგენილი მაგალითებით ნაჩვენებია პროგრამული ინჟინერიის ტექნოლოგიის შესაძლებლობები - თუ რამდენად მჭიდროდ უახლოვდება ბუნებრივ ენაზე შედგენილი ბიზნესპროცესების პროცედურები მოდელირებას, ხოლო მოდელირება პროგრამულ კოდს.

საკვანძო სიტყვები: პროგრამული ინჟინერია. მოდელებით მართვადი ტექნოლოგია. UML. BPMN. კოდის გენერაცია.

1. შესავალი

ორგანიზაციის ბიზნეს-პროცესების შესრულების მხარდამჭერი პროგრამული უზრუნველყოფა მაქსიმალურად მიახლოებული უნდა იყოს ორგანიზაციის ბიზნეს-მოთხოვნებთან და სრულად უნდა შეესაბამებოდეს ორგანიზაციის ფუნქციონალურ ან არა-ფუნქციონალურ მოთხოვნებს. ეს მოთხოვნები შესაძლებელია შემდგომში დაკორექტირდეს. ავტომატიზებული ბიზნეს-პროცესების რეალურ გარემოსთან ადაპტირებისას. ფაქტობრივი და ავტომატიზებული ბიზნეს-პროცესების შესრულების თანხვედრა მუდმივად დგას ორგანიზაციის ეფექტურად დაგეგმვის ამოცანებში, რაც ტესტირების ფართო ციკლებით, სიმულაციური მეთოდებითა და ანალიტიკური კვლევის ინსტრუმენტებით მიიღწევა.

ავტომატიზებული სისტემებისა და ბიზნეს-პროცესების შესაბამისობის ანალიტიკური კვლევის ფართო სპექტრს გვთავაზობს პროგრამული ინჟინერიის ტექნოლოგია, პროგრამული სისტემის მოდელირების ენების სიმრავლით. ანალიტიკური მიზნებისთვის ამ მოდელების ურთიერთაღქმადობით სრულყოფისთვის განვითარდა მოდელებით მართვადი არქიტექტურა (MDA-Model Driven Architecture), რაც სხვადასხვა მოდელების ურთიერთ ტრანსფორმაციისა და შინაარსობრივად დაკავშირების საშუალებაა, მათ შორის, პლატფორმადამოუკიდებელი არქიტექტურით [1].

მიდმინარე საერთაშორისო კვლევები საინფორმაციო ტექნოლოგიებში მიმართულია პროგრამული ენების ბუნებრივ ენასთან მჭიდროდ მიახლოების რეალიზაციაში. მოდელების დაკავშირებისა და ურთიერთგარდაქმნის განვითარება განსაკუთრებით საინტერესოა და პროდუქტიული ბუნებრივი ენიდან ანალიტიკური მოდელების გენერაციის ინსტრუმენტის ფუნქციონირებისთვის, ასევე ანალიტიკური მოდელების საინჟინრო კვლევისთვის და ავტომატიზაციისთვის. ხშირად, ბიზნეს-პროცესის აღწერა ორგანიზაციებში პროცედურის

სახით ფორმდება ტექსტური ფორმით. ეს მიღებული პრაქტიკაა, თუმცა ბიზნეს-პროცესების ანალიტიკურად გამოყენებისთვის აუცილებელია მისი საინჟინრო გარდაქმნა. ამ პრობლემის გადაჭრისთვის პროგრამული ინჟინერია აფართოებს დიაგრამების ელემენტების თვისებების ფორმას სცენარების შექმნის საშუალებით, ხოლო მოდელებით მართვადი ტექნოლოგიით სცენარში გაწერილი პროცედურები ავტომატურად გენერირდება სხვადასხვა ტიპის დიაგრამებად [2].

2. ძირითადი ნაწილი

პროგრამული ინჟინერიის მოდელების აგებისა და რეალიზაციის ტექნიკური გადაწყვეტაა CASE (Computer-aided software engineering) ტექნოლოგია, სადაც პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის (SDLC-systems development life cycle) მხარდამჭერი ინსტრუმენტული საშუალებები კლასიფიცირდება ოთხ დონედ:

1. ზედა დონის ინსტრუმენტები (Upper Case Tools) - გამოიყენება დაგეგმვის, ანალიზის და დაპროექტების ეტაპზე;

2. ქვედა დონის ინსტრუმენტები (Lower Case Tools) - გამოიყენება იმპლემენტაციის, ტესტირებისა და მხარდაჭერის ეტაპებისთვის;

3. ინტეგრირებული ინსტრუმენტები (Integrated Case Tools) - გამოიყენება პროგრამული უზრუნველყოფის დამუშავებისა და ფუნქციონირების ნებისმიერ ეტაპზე, მათ შორის დოკუმენტირების, მენეჯერული და ანალიტიკური ფუნქციების დაგეგმვა-მოდელირებისთვის;

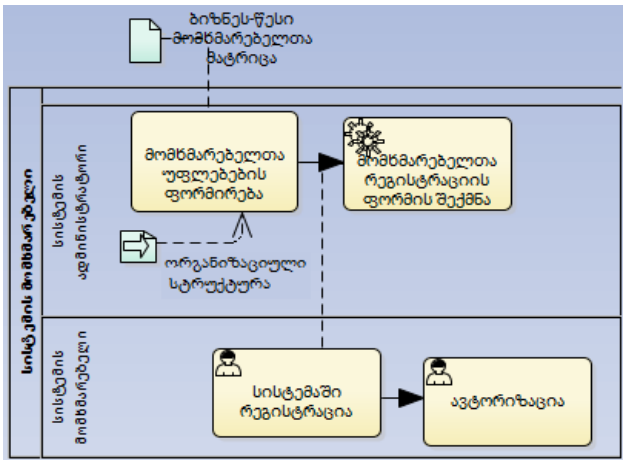
4. დამატებითი ინსტრუმენტები - ორგანიზაციული და სტრატეგიული დაგეგმვის, მოდელების სიმულაციის, ურთიერთასახვა-ტრანსფორმაციის, მოდელებიდან კოდის გენერაციის, სინქრონიზაციისა (მოდელში ან კოდში ელემენტების შეცვლის მყისიერი ურთიერთასახვა) და გამართვის (Debbug) მექნიზმებით.

მოდელებით მართვადი ტექნოლოგიით განისაზღვრა მოდელების ელემენტების იდენტურობა და სემანტიკური კავშირი, რის შედეგადაც უზრუნველყოფილი იქნა ზემოთ ჩამოთვლილი ინსტრუმენტების ე.წ. ლოგიკური გადაბმა [3].

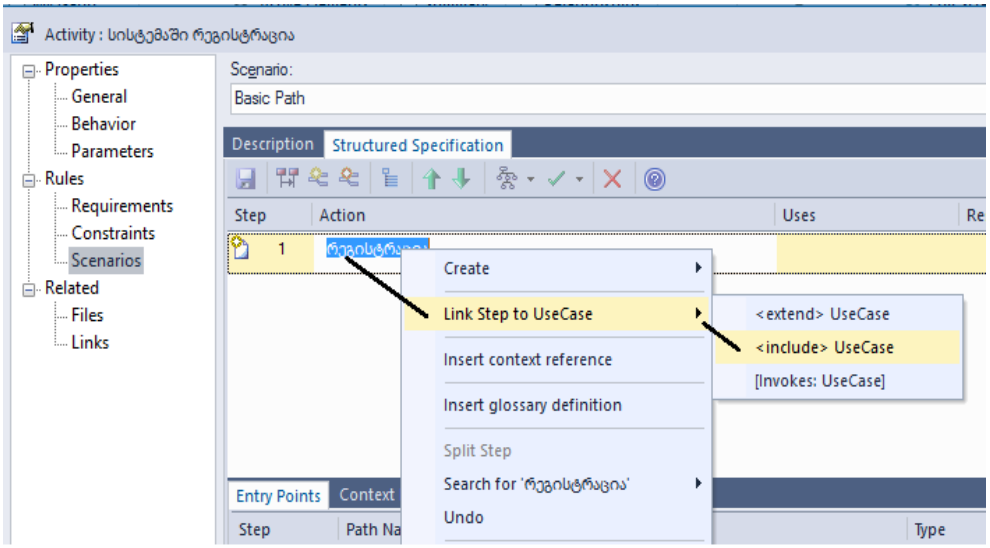
მაგალითად, ბიზნეს-პროცესების მოდელირების ნოტაციიდან UML (Unified Modeling Language) -UseCase დიაგრამის პარალელური აგება; UML-UseCase დიაგრამის პრეცედენტის სცენარული აღწერა და დაკავშირებული დიაგრამების - აქტიურობის, მიმდევრობითობის, მდგომარეობების, ტესტირების, მდგრადობის (robustness), ბიზნეს-წესების ავტოგენერაცია; კლასების დიაგრამაზე კლასის მეთოდების აღწერა არა მხოლოდ კარკასული (დასახელება, ტიპი, პარამეტრები), არამედ შინაარსობრივი ფორმით აქტიურობის, მიმდევრობითობის ან მდგომარეობების დიაგრამის სახით და პროგრამულ კოდად გენერაცია და სხვ. [4].

პროგრამული ინჟინერიის ზემოთ ჩამოთვლილი მიღწევების თვალსაჩინოებისთვის, განვიხილოთ რამდენიმე მაგალითი სარეალიზაციო პროგრამული სისტემის „მომხმარებელთა მართვის“ ნაწილისთვის Sparx Enterprise Architect ინსტრუმენტის გსმოყენებით.

1-ელ ნახაზზე მოცემულია სისტემის მომხმარებელთა მართვის ბიზნესპროცესის აღწერის BPMN (Business Process Model and Notation) დიაგრამა. მოდელებით მართვადი ტექნოლოგიის თანახმად განსაზღვრულია სემანტიკური კავშირი BPMN დიაგრამასა და UML-UseCase დიაგრამებს შორის. იდენტურია შემდეგი ელემენტები Pool/Line (BPMN) – Actor (UML UseCase), Activity (BPMN) – Case (UML UseCase). BPMN დიაგრამის ელემენტების თვისებებში სცენარის აღწერის ნაწილში შესაძლებელია თითოეული ”ქმედების” (Activity) მიბმა UML-UseCase დიაგრამასთან (ნახ.2).

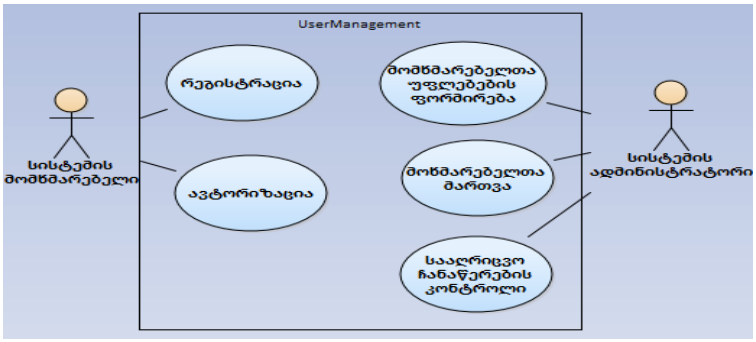


ნახ. 1. სისტემის მომხმარებელთა მართვის ბიზნეს-პროცესის აღწერის BPMN დიაგრამის ფრაგმენტი



ნახ.2. BPMN დიაგრამიდან UML-UseCase დიაგრამის შექმნის ფრაგმენტი

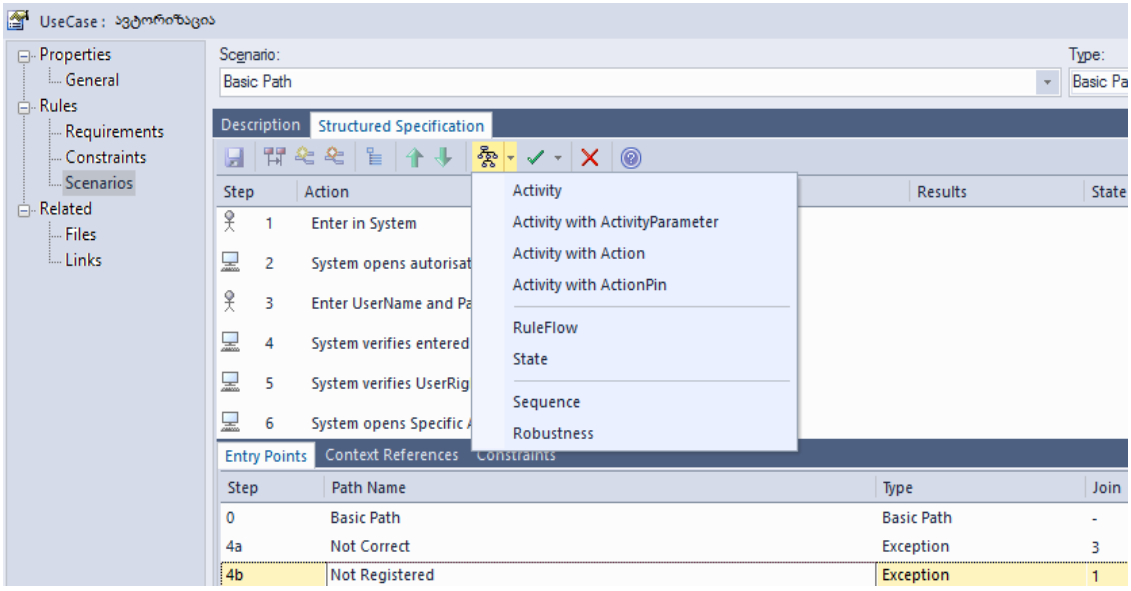
შედეგად ფორმირდება UML UseCase დიაგრამა (ნახ.3).



ნახ.3. BPMN დიაგრამიდან შექმნილი UML-UseCase დიაგრამა

შემოთავაზებულ UseCase დიაგრამის პრეცედენტზე „ავტორიზაცია” ფორმირებული სცენარი ნაჩვენებია მე-4 ნახაზზე (Sparx EA სისტემა ნაწილობრივ უჭერს მხარს ქართულ უნიკოდს, კოდის გენერაციის პრინციპებიდან გამომდინარე, ამდენად პროცედურა შედგენილია ინგლისურ ენაზე).

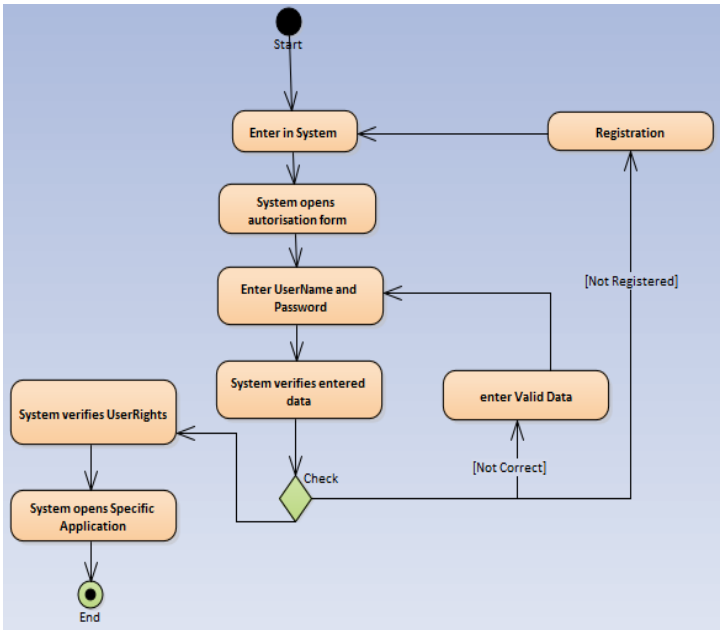
სცენარის პროცედურების მიხედვით დაგენერირებული აქტიურობის დიაგრამა ასახულია მე-5 ნახაზზე. პროგრამული ინჟინერიის სტანდარტის მიხედვით CASE მხარდამჭერი პროდუქტების მიმართ ერთ-ერთი აუცილებელი მოთხოვნაა პროგრამული კოდის პირდაპირი და რევერსიული გენერაცია.



ნახ.4. UseCase დიაგრამის პრეცედენტზე „ავტორიზაცია" სცენარის ფორმირების ფრაგმენტი

მოდელებით მართვადი ტექნოლოგიის საშუალებით პროგრამული კოდის პირდაპირი გენერაციის შესაძლებლობა მიეცა UML ქცევით მოდელებს, რომლებიც ამ შემთხვევაში აღწერს კლასის ერთი მეთოდის ლოგიკურ შინაარსს.

მაგალითად, „Autorisation" კლასში „CheckUser" მეთოდი ამოწმებს მომხმარებლის მიერ მითითებული მონაცემებით მის არსებობას მონაცემთა ბაზაში, იმ შემთხვევაში თუ კლიენტი არ იძებნება, იძახებს რეგისტრაციის „Registration" მეთოდს და იმახსოვრებს მონაცემებს. ამ ლოგიკით აგებული აქტიურობის დიაგრამა წარმოდგენილია მე-6 ნახაზზე, ხოლო 1-ელი ლისტინგი გვიჩვენებს SparxEA სისტემიდან C#.NET პლატფორმაში გენერირებულ კლასს „CheckUser" მეთოდით.

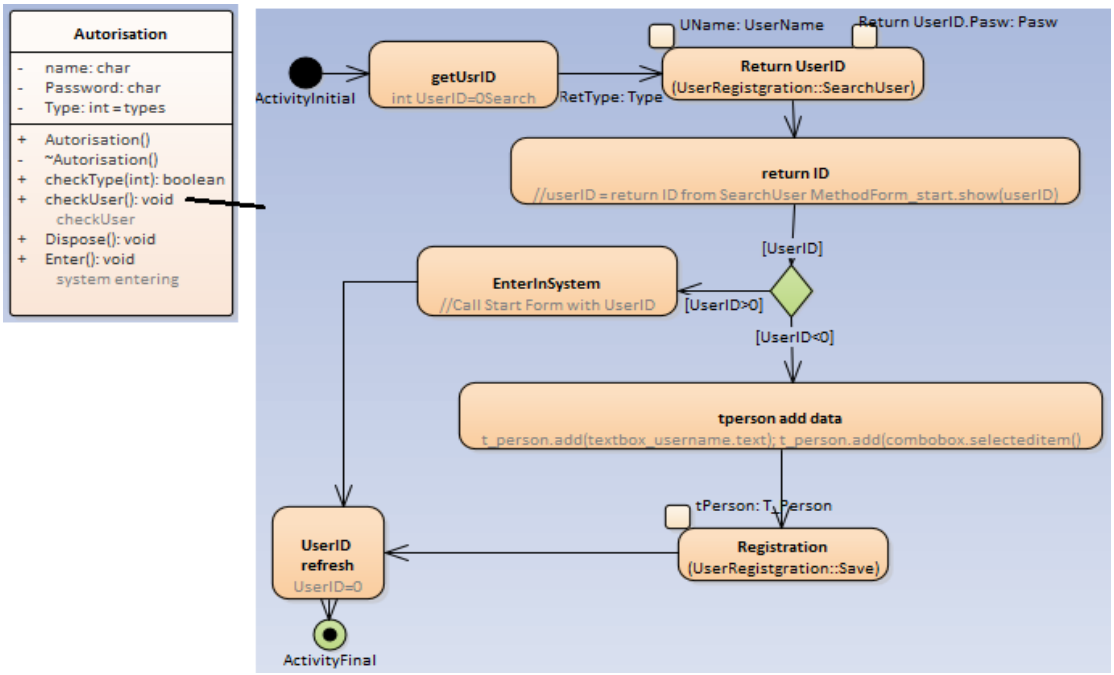


ნახ.5. სცენარის პროცედურების მიხედვით დაგენერირებული აქტიურობის დიაგრამა

მიუხედავად ნაშრომში მოცემული შედეგებისა, პროგრამული ინჟინერიის ტექნოლოგიაში აქტუალური რჩება და შემდგომი კვლევის მიმართულებებია არაფორმალიზებული დიაგრამებიდან ფორმალიზებული დიაგრამების გენერაცია, პროგრამული კოდის ქცევით მოდელებში ასახვა, პროგრამული კოდის ავტომატიზებული ტესტირება და სხვა.

3. დასკვნა

მაქსიმალურად მიახლოებისა და აღწერისთვის დღეს, პროდუქტიულია პროგრამული ინჟინერიის ტექნოლოგია სხვადასხვა ტიპის მოდელებისა და დიაგრამების სიმრავლით, პროგრამული კოდის პირდაპირი და რევერსიული გენერაციით.



ნახ.6. UML კლასის მეთოდის შექმნის აქტიურობის დიაგრამის ფრაგმენტი

ლისტინგი.1. SparxEA სისტემიდან .NET C# პლატფორმაში გენერირებული კლასის

```
//  Autorisation.cs
//  Implementation of the Class Autorisation
//  Generated by Enterprise Architect
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
namespace System {
        public class Autorisation {
private char name; private char Password; private int Type = types; private Textbox_pass
private Combobox_Posicion Posicion; private Textbox_Username UserName;
public UserRegistgration(){}
        /* Begin - EA generated code for  Activities and Interactions */
public void CheckUser( int userID)
                {
// behavior is a Activity
                        int UserID=0;
                        SearchUser(textbox_username.text,Combobox_Posicion.selecteditem);
                        Form_start.show(userID);
                        if (UserID>0)
                        {//Call Start Form with UserID  ;
                        }
                        else if (UserID<0)
                        {       List <Tperson> tperson=new List <Tperson>;
                        Datagridview Datagridview_user;
                        while (DataGridViewRow row in Datagridview_user.Rows)
                        {
                                tperson.add(row.Cells[0].Value.ToString());
                                tperson.add.row.Cells[1].Value.ToString();
                                // add data to list tperson;
                        }
                        Save(t_person);
                        }
                        UserID=0;
                }
}//end Autorisation
}//end namespace System
```

ფრაგმენტი მეთოდით „CheckUser“

პროგრამული კოდის გენერაციის ტექნიკამ მნიშვნელოვანი ბერკეტები შექმნა ორგანიზაციის ბიზნეს-პროცესების აგებისთვის. ამით უზრუნველყოფილია ერთის მხრივ პროგრამული სისტემის დამუშავების ანალიტიკოსების, დამპროექტებლების, შემფასებლებისა და ექსპერტების ერთობლივი მოქნილი მუშაობა, ხოლო მეორეს მხრივ დეველოპერების ტვირთის შემცირება ტერმინოლოგიის, პროგრამული სისტემის პაკეტების, კლასების, მეთოდების განთავსებისა და დოკუმენტირების მხრივ. პროგრამული ინჟინერიის გაფართოება მოდელებით მართვადი ტექნოლოგიით უზრუნველყოფს ბიზნეს-პროცესების ცვლილებების სწრაფ გავრცელებას ანალიტიკურ ინსტრუმენტებზე, მოდელების მანიპულაციასა და ბუნებრივ ენაზე აღწერილი პროცედურების საინჟინრო-ტერმინებში მიღებას.

ლიტერატურა:

1. Rodrigues A. (2015). Model-driven engineering: A survey supported by the unified conceptual model, Computer Languages, Systems & Structures V. 43, Portugal,
2. Koehler J., Hauser R., Kuste J. (2006). The Role of Visual Modeling and Model Transformations in Business-driven Development, Proceedings of the Fifth International Workshop on Graph Transformation and Visual Modeling Techniques, Austria.
3. http://www.c-sharpcorner.com/uploadfile/nipuntomar/computer-aided-software-engineering-tools-case/
4. Sparx Ea Users guide, http://www.sparxsystems.com/enterprise_architect_user_guide/__13.0/ model_ domains / codeengineering.html

EXTENDED USE OF SOFTWARE ENGINEERING TECHNOLOGY WITH MODEL-DRIVEN ARCHITECTURE

Turkia Ekaterine, Jibuti Davit, Stomadova Sophio
Georgian Technical University

Summary

The paper describes extended application of software engineering technology from formal description of company's business processes to software development. For this purpose, corresponding Use CASE tools levels of software life cycle and their logical connections with model-driven technology are reviewed. As a practical example, Use CASE diagram is constructed from business process diagram; from Use CASE diagram an activity diagram is auto-generated; description of method inside class diagrams using activity diagram; and code generation with conditional and cyclic operators. Presented examples show current possibilities of software engineering – how close can business processes described in natural language be to procedure modeling, and furthermore modeling to software code.

РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ ПРОГРАММНОЙ ИНЖЕНЕРИИ АРХИТЕКТУРОЙ УПРАВЛАЕМОЙ МОДЕЛЯМИ

Туркия Е., Джибути Д., Стомадова С.
Грузинский Технический Университет

Резюме

Рассматривается расширенное использование технологии программной инженерии бизнес-процессов организации с формализованного описания до проектирования программного обеспечения. Для этого, Рассматриваются / Обсуждаются этапы жизненного цикла программного обеспечения соответствующие Case Tools уровни и логические уровни соединения моделями управляемых технологий. Практическими примерами представлены с диаграмм бизнес-процессов формирование диаграмм UML Use Case, автоматическая генерация Диаграммы активности (activity) с диаграмм Use Case, в диаграмме классов описание метода / Способа с помощью диаграммы активности (activity) и генерация кода сопровождением методами условных и циклических операторов. Пприведенными примерами показано текущие возможности технологии программной инженерии – насколько тесно приближаются друг к другу моделирование к процедуре бизнес процессов созданных на естественном языке, а моделирование к программному коду.