პროგრამული აპლიკაციის დამუშავების სასიცოცხლო ციკლი Visual Studio.NET Framework-ის ახალ ვერსიებში

გია სურგულაძე, თინათინ კაიშაური,
გულბაათ ნარეშელაშვილი, გიორგი მაისურაძე

საქართველოს ტექნიკური უნივერსიტეტი

რეზიუმე

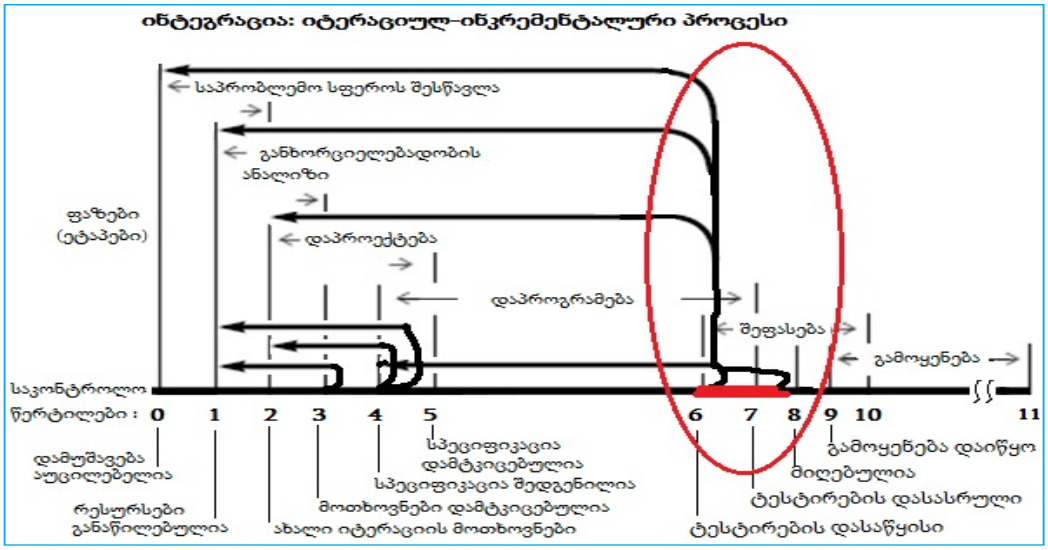
განიხილება პროგრამული აპლიკაციების შექმნის სასიცოცხლო ციკლის ძირითადი ეტაპების (მოდელირება, დაპროექტება, რეალიზაცია, რეფაქტორინგი, ტესტირება და თანხლება) პროცესების ავტომატიზაციის საკითხები Visual Studio.NET 2013/15-ის ვერსიებში. კერძოდ, წარმოდგენილია უნიფიცირებული მოდელირების ენის შესაძლებლობების რეალიზაცია ამ ინტეგრირებულ გარემოში. აგებულია შესაბამისი UML-დიაგრამები უშუალოდ VisualStudio.NET-ში. განხორციელებულია კლასთა დიაგრამიდან C#-კოდის გენერაცია და გაანალიზებულია რევერსიული პროგრამირების შესაძლებლობები.

საკვანძო სიტყვები: დაპროგრამება. პროგრამის სასიცოცხლო ციკლი. UML. Visual Studio.NET. რევერსიული პროგრამირება. კლასთა დიაგრამა. C#-კოდი.

1. შესავალი

სრულყოფილი, საიმედო და მოქნილი პროგრამული უზრუნველყოფის (Software Engineering) სწრაფად დაპროექტება, რეალიზაცია, დანერგვა და შემდგომი თანხლება სისტემის დამკვეთ ორგანიზაციაში მეტად მნიშვნელოვანი ამოცანაა და მისი ეფექტურად გადაწყვეტა ბევრადაა დამოკიდებული როგორც საპროექტო-დეველოპმენტის გუნდის შემადგენლობასა და გამოცდილებაზე, ასევე IT-ინფრასტრუქტურასა და CASE-ინსტრუმენტებზე [1].

პროგრამული სისტემის მენეჯმენტის საკონტროლო (0-12) წერტილებში (ნახ.1), ეტაპების მიხედვით ხორციელდება იტერაციული სამუშაოები (დაბრუნება უკანა წერტილებში განმეორებითი პროცედურების ჩასატარებლად), განსაკუთრებით ტესტირების ეტაპზე, სისტემის ფუნქციონალობის სისრულის დაზუსტების ან გაფართოების მიზნით [2,3].

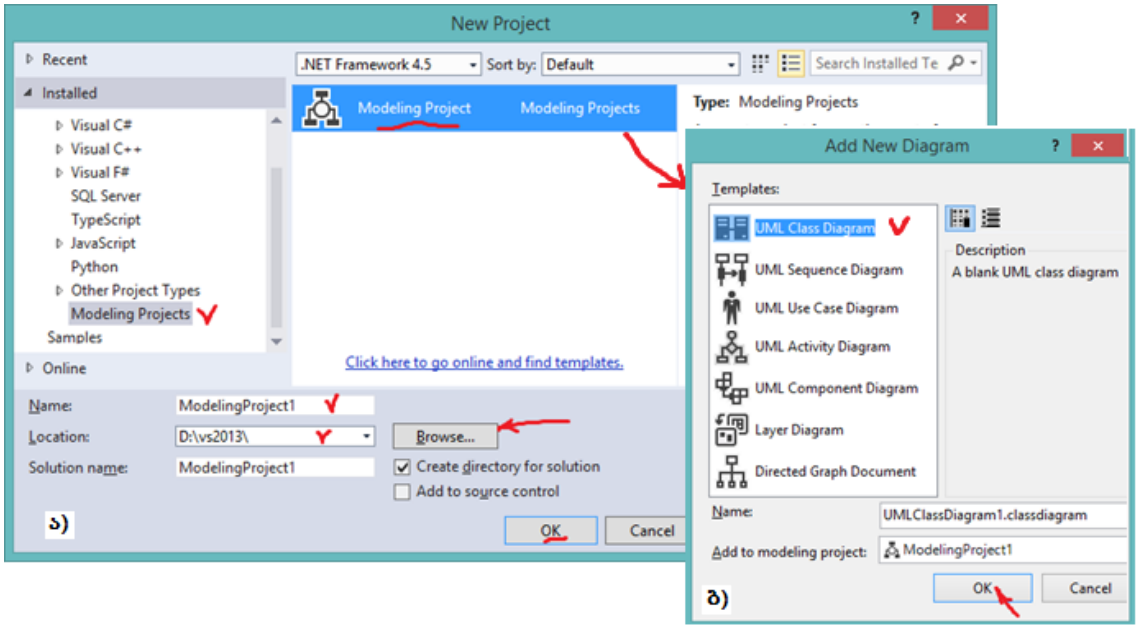


ნახ.1. პროგრამული სისტემის სასიცოცხლო ციკლი ტესტირებით

Visual Studio 2013/15-ის ვერსიათა ინტეგრირებული გარემოსთვის დამახასიათებელია ერთ-ერთი მნიშვნელოვანი შესაძლებლობა, პროგრამული პროექტის დამუშავების სასიცოცხლო ციკლის ყველა ეტაპის მხარდაჭერა, რასაც წინამდებარე სტატიაში განვიხილავთ [4].

დიდი პროექტებისათვის, რომელშიც რესურსები და დროითი ფაქტორები, შედარებით კრიტიკული არაა, ხდება ობიექტ-ორიენტირებული მიდგომის ყველა ეტაპის და ფაზის გამოყენება შესაბამისი საკონტროლო წერტილების აუცილებელი მონიტორინგით და რეპორტებით. ამ დროს სრული მოცულობით ხორციელდება უნიფიცირებული მოდელირების ენის (UML) და შესაბამისი CASE ინსტრუმენტული საშუალების, მაგალითად, Enterprise Architect, Rational Rose, Visual Paradigm ან სხვა პაკეტების გამოყენება [5,6].

Visual Studio.NET Framework 4.5/4.6 ახალი ვერსიებისთვის UML დიაგრამების აგების გრაფო-ანალიზური საშუალებები თითქმის მთლიანადაა ჩადებული (მოდელირების უმრავლესი ამოცანებისთვის აღარაა საჭირო ზემოთ ნახსენები CASE-ინსტრუმენტები). 2-ა და 2-ბ ნახაზებზე ილუსტრირებულია ახალი პროექტის აგების მაგალითი Visual Studio.NET 2013 გარემოში.



ნახ.2. Visual Studio.NET 2013 გარემოში UML მოდელების პროექტის შექმნა

Visual Studio 2013 -ში, ისევე როგორც სხვა ინტეგრირებულ ინსტრუმენტებში, არსებობს პროგრამული აპლიკაციის სტრუქტურის მოდელირების მხარდაჭერა UML ენაზე. მოდელირების ეს ენა შეიძლება გამოყენებულ იქნას პროექტის დამუშავების სხვადასხვა ეტაპზე, მაგალითად საპრობლემო სფეროს შესწავლის და სიტემის მოთხოვნლების განსაზღვრის ეტაპზე, ინტერაქტიული სცენარების ასაგებად ობიექტ-ორიენტირებული ანალიზის ეტაპზე, კლასთა-იერარქიის მოდელირებისათვის ობიექტ-ორიენტირებული დაპროექტების ეტაპზე და ა.შ. [5].

UML ენა საშუალებას იძლევა დაპროექტდეს კლასების იერარქია აბსტრაქტულ ტერმინებში, წარმოადგინოს იგი მოდელის სახით, ხოლო მოდელი - დიაგრამის სახით. შემდეგ ამ დიაგრამის საფუძველზე შეიძლება გენერირებულ იქნას კოდი არჩეულ ენაზე (მაგალითად, C#-ზე). ამგვარად, UML ენის გამოყენებით შეიძლება მოდელირებისა და პროექტირების ეტაპებიდან მოხდეს გადასვლა რეალიზაციის ეტაპზე. გენერირებული კოდი მომავალში შეიძლება განხილულ იქნას როგორც

საფუძველი პროექტის შემდგომი რეალიზაციისათვის. ასევე შესაძლებელია პირიქით, უკვე დამუშავებული კოდით გენერირდეს ამ კოდის მოდელი UML ენაზე, რაც, საერთო ჯამში, რევერსიული პროგრამირების კარგ ინსტრუმენტად ითვლება.

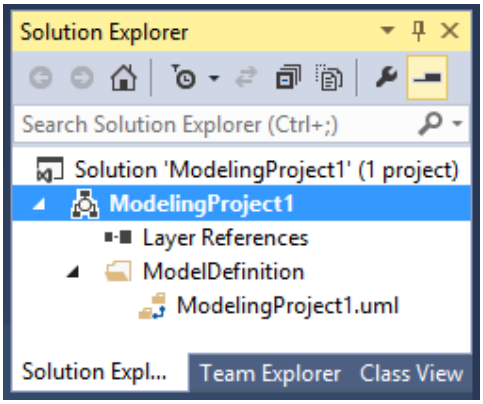
2. ძირითადი ნაწილი

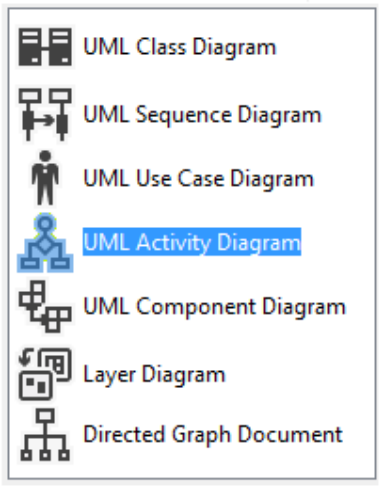
2.1. UML დიაგრამების აგების საშუალებები

გავიხილოთ Visual Studio 2013 სამუშაო გარემოში UML-ის დიაგრამების აგების ინსტრუმენტის შესაძლებლობები. როგორც 2-ა,ბ ნახაზებზეა ნაჩვენები, სპეციალური სახის პროექტი აიგება Modeling Project-ის მიხედვით:

File / NewProject და პროექტის ტიპი: Modeling Project

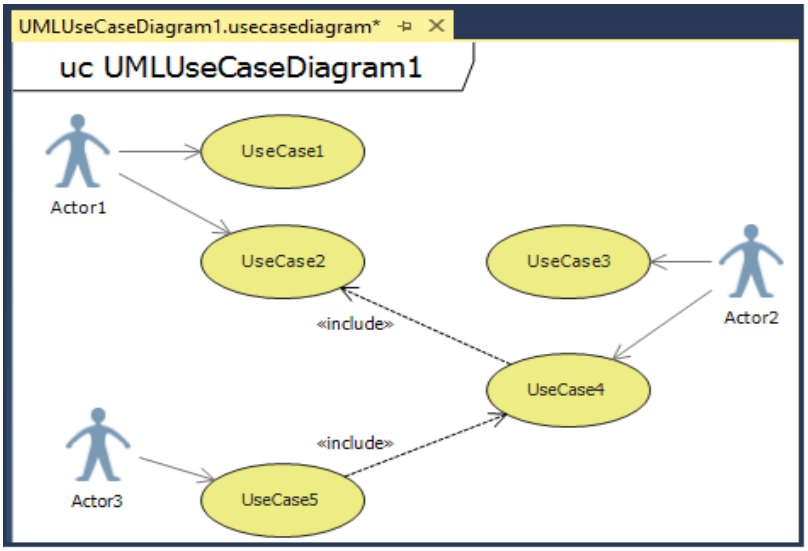
Solution Explorer-ში მივიღებთ შემდეგ სურათს (ნახ.3):



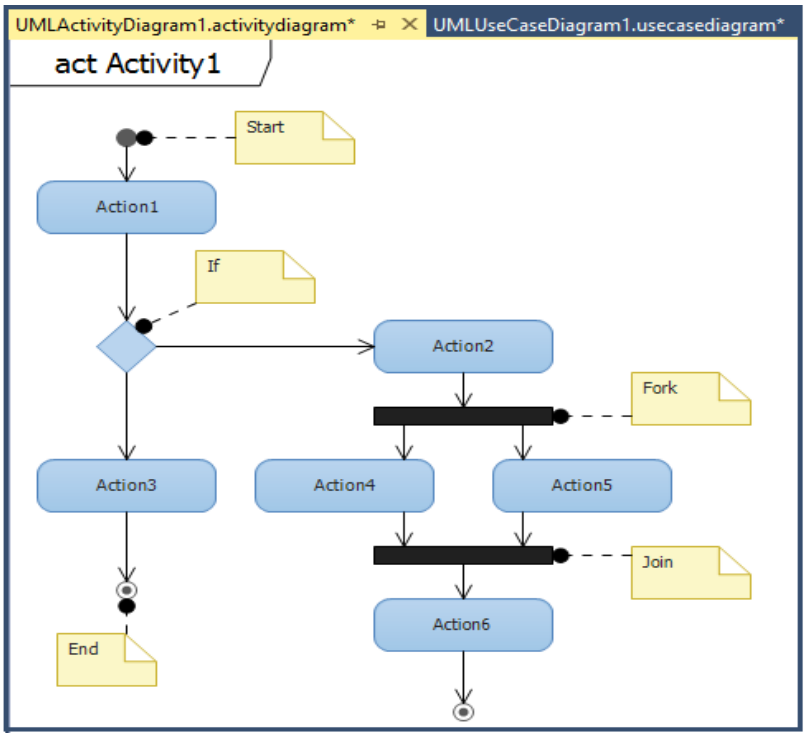


ნახ.3. Solution Explorer-ში შეიქმნა ModelingProject1 **ნახ.4. ასაგები დიაგრამის არჩევა**

ამ პროექტზე Add New Diagram-ით მივიღებთ მე-4 ნახაზზე ნაჩვენებ მენიუს, საიდანაც, მაგალითად, ავირჩევთ ჯერ UML Use Case Diagram, შემდეგ კი – UML Activity Diagram სტრიქონს. მე-5,6 ნახაზებზე ნაჩვენებია, შესაბამისად, ჩვენ მიერ აგებული საილუსტრაციო მოდელები.



ნახ.5. Use Case დიაგრამის მაგალითი VisualStudio.NET 2013 გარემოში

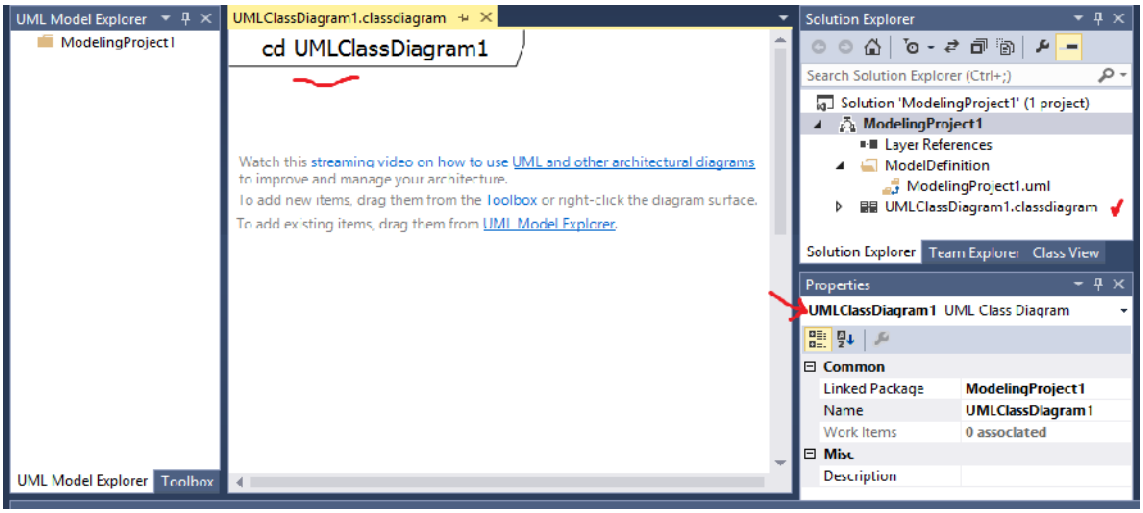


ნახ.6. Activity დიაგრამის მაგალითი VisualStudio.NET 2013 გარემოში

ამგვარადვე შესაძლებელია დანარჩენი დიაგრამების აგებაც VisualStudio.NET 2013 გარემოში, როგორებიცაა მაგალითად, მიმდევრობითობის (Sequence), კომპონენტების (Component) და სხვა დიაგრამები. განსაკუთრებით საყურადღებოა კლასთა დიაგრამების მოდელირება და მის საფუძველზე პროგრამული კოდის ავტომატიზებული გენერაცია, რასაც მომდევნო პარაგრაფში დეტალურად განვიხილავთ.

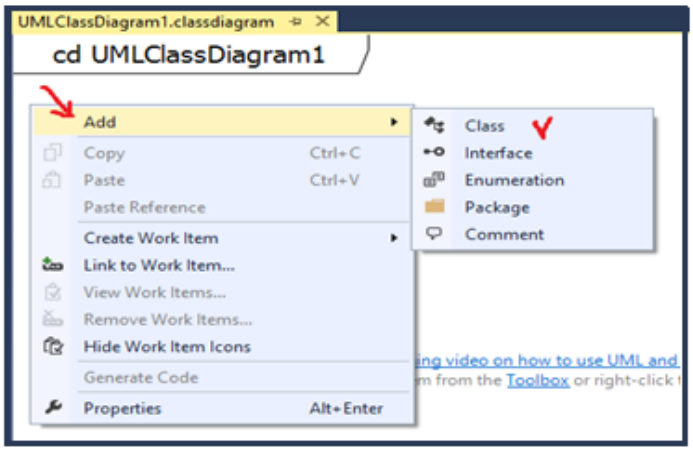
2.2. კლასების დიაგრამის აგება და კოდის გენერაცია

შევქმნათ ახალი კლასთა დიაგრამა, თავიდან ცარიელი (ნახ.7).

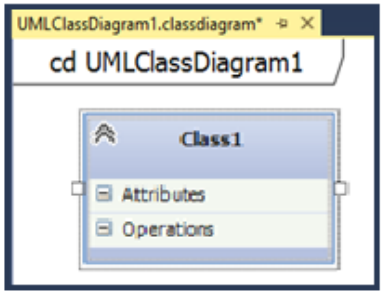


ნახ.7. კლასების (ცარიელი) დიაგრამის შექმნა

კლასის დასამატებლად დიაგრამაზე ვირჩევთ მენიუს პუნქტს Add / class (ნახ.8). ახალი კლასი ემატება ავტომატურად სახელით Class1. იგი შედგება ატრიბუტებისა და ოპერაციებისგან (ნახ.9).

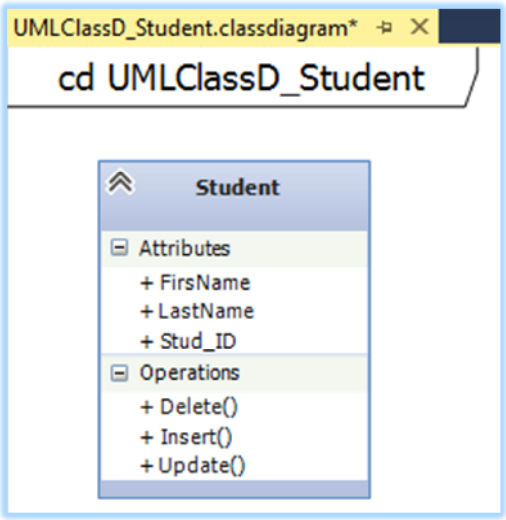


ნახ.8. დიაგრამაზე ახალი კლასის დამატება



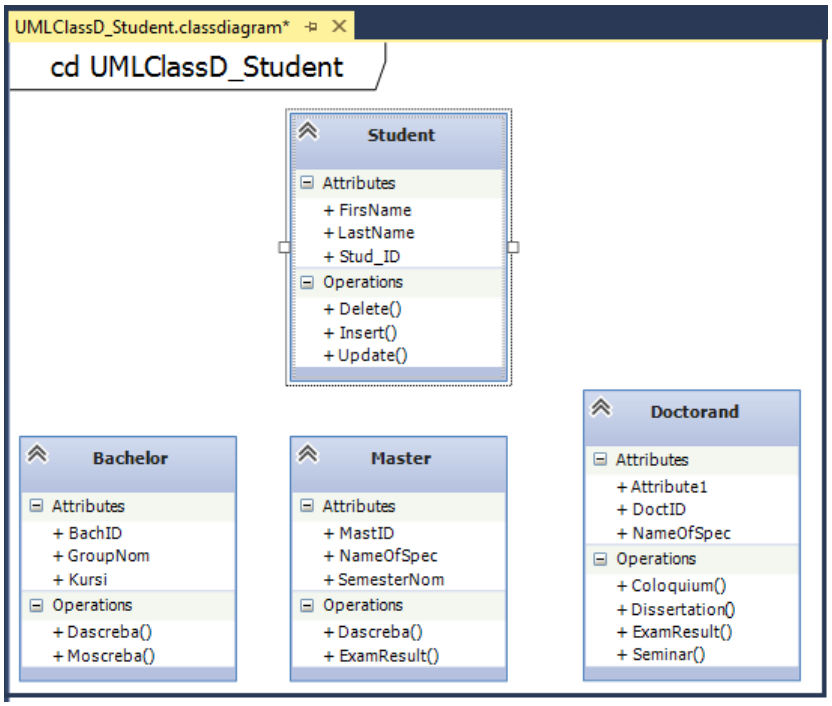
ნახ.9. კლასი ატრიბუტებით და ოპერაციებით

შევცვალოთ მოდელში კლასის სახელი Student-ით, ატრიბუტის სახით დავამატოთ St_ID, FirstName, LastName, ხოლო ოპერაციის სახით Input, Update და Delete. დასამატებლად გამოიყენება კონტექსტური მენიუ (მაუსის მარჯვენა ღილაკით) და პუნქტი Add. შედეგი მოცემულია მე-10 ნახაზზე.



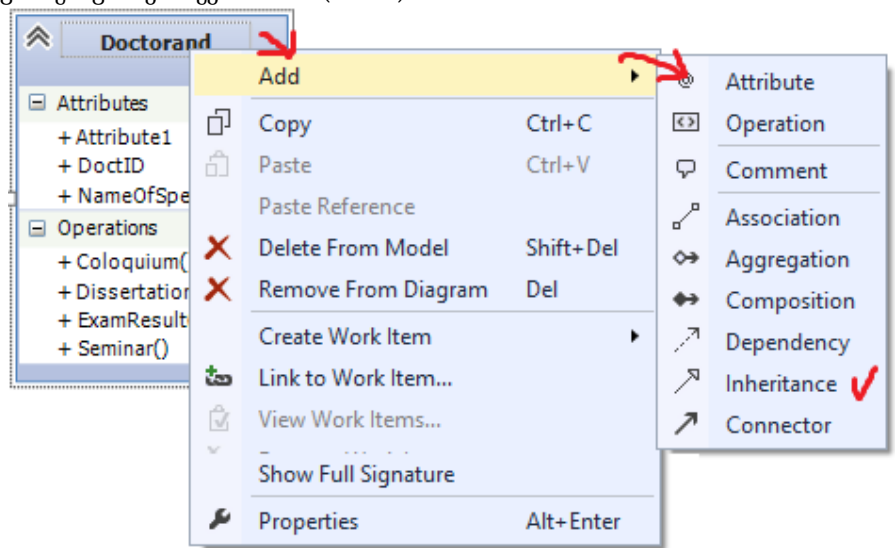
ნახ.10. Student კლასის შედეგი

დავამატოთ „შვილობილი" კლასები: Bachelor, Master და Doctorand, მათთვის დამახასიათებელი თვისებებით (ატრიბუტებით), როგორც მე-11 ნახაზზეა ნაჩვენები.



ნახ.13. სამი კლასის დამატება

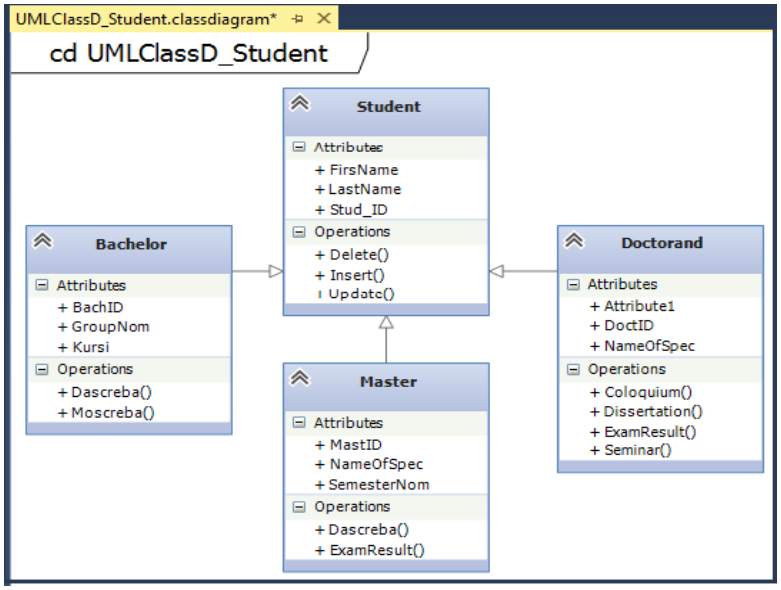
ახლა დიაგრამაზე უნდა ავსახოთ ინფორმაცია იმის შესახებ, რომ Bachelor, Mastyer და Doctorand კლასები არის Student კლასის მემკვიდრეები. ვდგებით თვითოეულზე კლასზე და კონტექსტურ მენიუში ვირჩევთ Add-ს (ნახ.14).

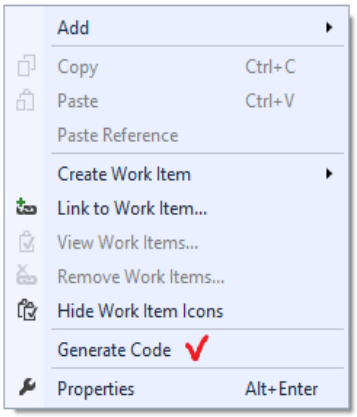


ნახ.14. მემკვიდრეობითობის კავშირის არჩევა

აქ უნდა ამოვირჩიოთ კავშირის ელემენტი და დავამატოთ დიაგრამაზე: Attribute - ახალი ატრიბუტი, Operation - ახალი ოპერაცია, Comment - კომენტარი; Association – ასოციაცია; Aggregation – აგრეგაცია; Composition – კომპოზიცია; Dependency – დამოკიდებულება; Inheritance – მემკვიდრეობითობა; Connector- კონექტორი. ჩვენ გვაინტერესებს Inheritance.

Add / Inheritance - არჩევით დიაგრამაზე დაემატა მემკვიდრეობითობის კავშირი ისრით მიმართული Doctorand-იდან Student-ისკენ. საბოლოოდ მივიღებთ მთლიან დიაგრამას (ნახ.16). ახლა შეიძლება პროგრამული კოდის გენერირება, რომელსაც შემდომში გამოვიყენებთ. კონტექსტურ მენიუში ვირჩევთ პუნქტს - Generate code (ნახ.17).

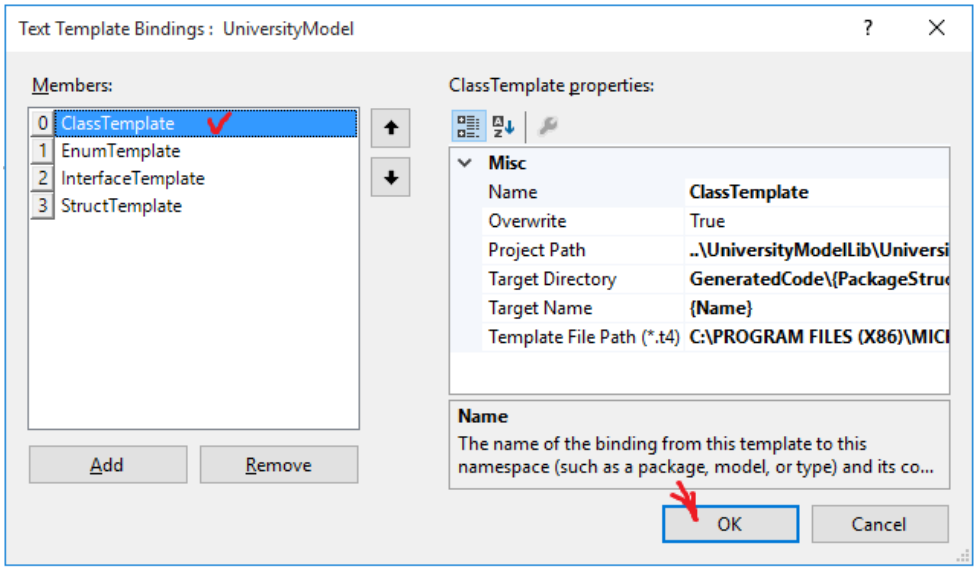




ნახ.17. კოდის გენერაციის ფუნქცია

ნახ.16. მემკვიდრეობითი კლასთაშორისი კავშირები – Inheritance

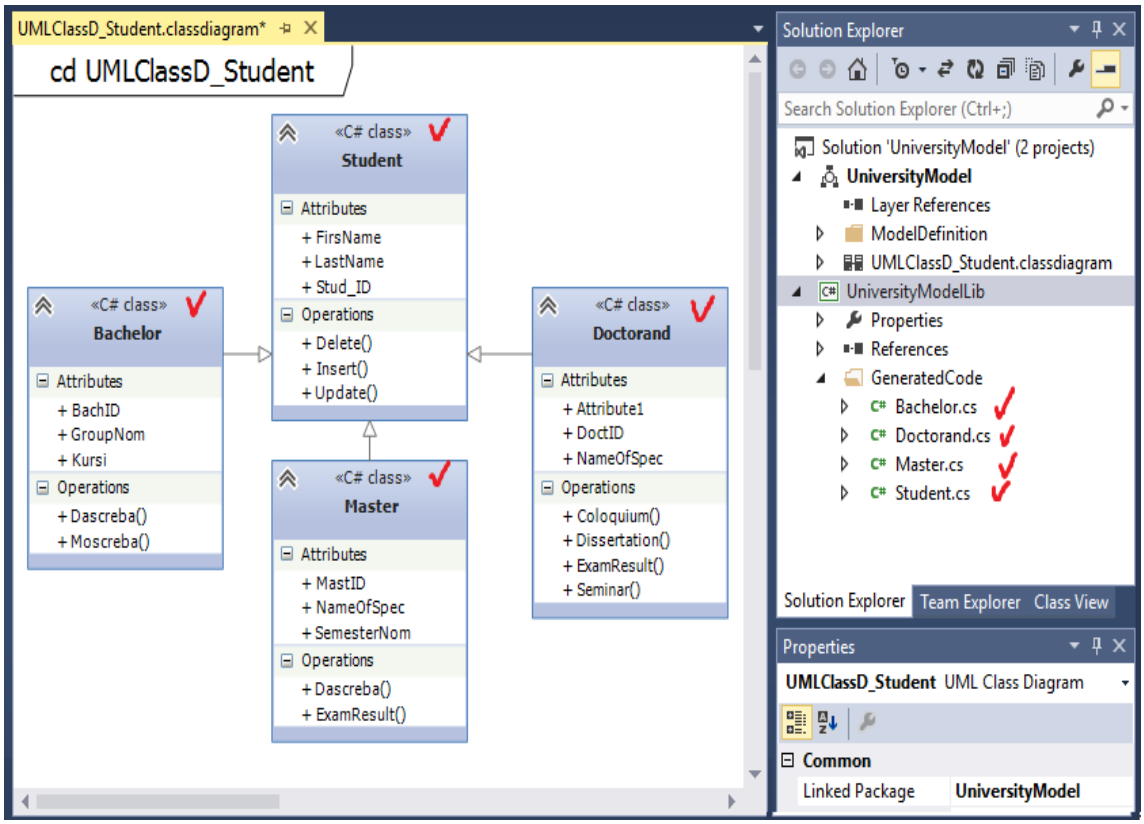
კოდის გენერატორი გვთხოვს დავაზუსტოთ თუ რომელი შაბლონით (Template) მოხდება გენერაცია. ვირჩევთ შაბლონს კლასისთვის (ნახ.18).



ნახ.18. ClassTemplate – შაბლონის არჩევა კოდის გენერაციისთვის

კოდის გენერაციის შემდეგ Solution Explorer-ში გამოჩნდება ორი ახალი სტრიქონი, C#-ის ფაილებისთვის: Student.cs, Bachelor.cs, Master.cs და Doctorand.cs (ნახ.19).

გავხსნათ ახალი Student.cs, Bachelor.cs, Master.cs და Doctorand.cs ფაილები. პროგრამის ტექსტები მოცემულია 1-4 ლისტინგში.



ნახ.19. Solution Explorer-ში გამოჩნდა შექმნილი C#-კოდები

```
//----ლისტინგი_1:  Student.cs ---------------------------------------------
// <auto-generated>
//     This code was generated by a tool
//     Changes to this file will be lost if the code is regenerated.
// </auto-generated>
//-----------------------------------------------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
public class Student     // „მშობელი“ კლასი
  {
       public virtual object LastName { get; set; }
       public virtual object FirsName { get; set; }
       public virtual object Stud_ID  { get; set; }
       public virtual void Insert() {throw new System.NotImplementedException();}
       public virtual void Update() {throw new System.NotImplementedException();}
       public virtual void Delete() {throw new System.NotImplementedException();}
  }

//-----  ლისტინგი_2:  Bachelor.cs --------------------------------
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
public class Bachelor : Student  // „შვილობილი“ კლასი
  {
       public virtual object BachID { get; set; }
       public virtual object GroupNom { get; set;}
       public virtual object Kursi {get; set; }
       public virtual void Dascreba(){throw new System.NotImplementedException();}
       public virtual void Moscreba(){throw new System.NotImplementedException();}
  }

//----- ლისტინგი_3:  Masterr.cs ----------------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
public class Master : Student   // „შვილობილი“ კლასი
  {
     public virtual object MastID {get; set;}
     public virtual object NameOfSpec {get;   set;}
     public virtual object SemesterNom {get;set;}
     public virtual void Dascreba(){throw new System.NotImplementedException();}
     public virtual void ExamResult(){throw newSystem.NotImplementedException();}
  }

//------- ლისტინგი_4:  Doctorand.cs ----------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
public class Doctorand : Student   // „შვილობილი“ კლასი
 {
   public virtual object DoctID {get; set;    }
   public virtual object NameOfSpec {get; set;}
   public virtual object Attribute1 {get; set;}
   public virtual void Seminar(){throw new System.NotImplementedException();}
   public virtual void Coloquium(){throw new System.NotImplementedException();}
   public virtual void ExamResult(){throw new System.NotImplementedException();}
   public virtual void Dissertation(){throw new System.NotImplementedException();}
 }
```

აქ ატრიბუტები (მაგალითად, FirsName, LastName, StudID და სხვა) რეალიზებულია კლასის თვისებების (propeties) სახით get და set მეთოდებით. ხოლო მეთოდების (მაგალითად, Insert, Update, Delete, Dascreba, Moscreba, Seminar და სხვა) სახშობები წარმოდგენილია ვირტუალური მეთოდების სახით და რეალიზებულია როგორც გამონაკლისის გენერაცია. მაგალითად,

```
   public virtual void Insert()
     {
       throw new System.NotImplementedException();
     }
```

```
public virtual void Update()
  {
   throw new System.NotImplementedException();
  }
  ...
public virtual void Seminar()
  {
   throw new System.NotImplementedException();
  }
```

ლისტინგის კოდებიდან კარგად ჩანს მემკვიდრეობითობის (inheritance) კავშირი (გამოკვეთილია კომენტარით): „მშობელი"-„შვილობილი" კლასებით.

3. დასკვნა

გამოყენებითი პროგრამული სისტემების სასცოცხლო ციკლის ეფექტური მართვის თვალსაზრისით ობიექტ-ორიენტირებული მიდგომის გამოყენებისას განსაკუთრებული მნიშვნელობა აქვს დიდი პროექტების UML მოდელების (დიაგრამების) და პროგრამული კოდების თავსებადობის და მოდიფიკაციის საკითხებს. გენერირებული ფაილები შეიძლება გამოყენებულ იქნას დამუშავების მომდევნო ეტაპებზე. სასიცოცხლო ციკლის საწყისი ეტაპების შედეგების ხარისხი, როგორიცაა საპრობლემო სფეროს შესწავლა და სისტემის ბიზნეს-სპეციფიკაციების განსაზღვრა, უშუალოდ იჩენს თავს სისტემის დეველოპმენტის შემდეგ, ტესტირების ეტაპზე. იტერაციულ-ინკრემენტალური პროცესი სისტემის მომდევნო ვერსიის ასაგებად ეფექტურად განხორციელდება რევერსიული ინჟინერინგის ((reverse engineering) გამოყენებით. ანუ, პროექტის შეცვლის აუცილებლობის შემთხვევაში, შეიძლება ცვლილებები ჩატარდეს UML-დიაგრამებში, საიდანაც კოდები ავტომატურად იქნება გენერირებული. შესაძლებელია პირიქითაც, რეალიზებული კლასებიდან მოხდეს UML-დიაგრამების გენერირება, რაც მეტად მნიშვნელოვანია პროგრამული სისტემის კვლევისა და მისი ხარისხის მართვის სრულყოფისათვის.

ლიტერატურა:

1. სურგულაძე გ., ურუშაძე ბ. (2014). საინფორმაციო სისტემების მენეჯმენტის საერთაშორისო გამოცდილება (BSI, ITIL, COBIT). სტუ. „ტექნიკური უნივერსიტეტი". თბილისი.

2. სურგულაძე გ., ბულია ი. (2012). კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. სტუ. „ტექნიკური უნივერსიტეტი". თბილისი.

3. სურგულაძე გ., გულიტაშვილი მ., კივილაძე ნ. (2015). Web-აპლიკაციების ტესტირება, ვალიდაცია და ვერიფიკაცია. სტუ. „IT-კონსალტინგის ცენტრი". თბილისი.

4. Create UML modeling projects and diagrams. (2015) https://msdn.microsoft.com/en-us/library/dd409445.aspx.

5. გოგიჩაიშვილი გ., ბოლხი გ. (გერმ.), სურგულაძე გ., პეტრიაშვილი ლ. (2013). მართვის ავტომატიზებული სისტემების ობიექტ-ორიენტირებული დაპროექტების და მოდელირების ინსტრუმენტები (MsVisio, WinPepsy, PetNet, CPN). სტუ. „ტექნიკური უნივერსიტეტი". თბილისი.

6. თურქია ე. (2010). ბიზნესპროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. სტუ. თბილისი.

SOFTWARE DEVELOPMENT LIFE CYCLE IN THE LATEST VERSIONS OF VISUAL STUDIO.NET FRAMEWORK PACKAGE

Surguladze Gia, Kaishauri Tinatin,
Nareshelashvili Gulbaat, Maisuradze Giorgi

Georgian Technical University

Summary

Article discusses process automation issues for basic phases of software development life cycle (modeling, design, implementation, refactoring, testing and support) in versions of Visual Studio.NET 2013/15 package. In particular, implementing new possibilities of the unified modeling language are given within this integrated environment. Respective UML diagrams have been developed using Visual Studio.NET. The process of generating C# code from class association diagram has been investigated based on an experimental task and new possibilities of reverse programming have been analyzed.

ЖИЗНЕННЫЙ ЦИКЛ РАЗРАБОТКИ ПРОГРАММНЫХ АППЛИКАЦИЙ В НОВЫХ ВЕРСИЯХ ПАКЕТА VISUAL STUDIO.NET FRAMEWORK

Сургуладзе Г., Каишаури Т.,
Нарешелашвили Г., Маисурадзе Г.

Грузинский Технический Университет

Резюме

Рассматриваются вопросы автоматизации процессов основных этапов жизненного цикла разработки программных аппликаций (моделирование, проектирование, реализация, рефакторинг, тестирование и сопроваждение) для версий Visual Studio.NET 2013/15. В частности, представлены новые возможности реализации языка унифицированного моделирования в в этой интегрированной среде. Построены соответствующие UML-диаграммы непосредственно в VisualStudio.NET. На основе экспериментальной задачи проведено исследование процесса генерации C# кода из диаграммы ассоциации классов и проанализированы новые возможности реверсного программирования.