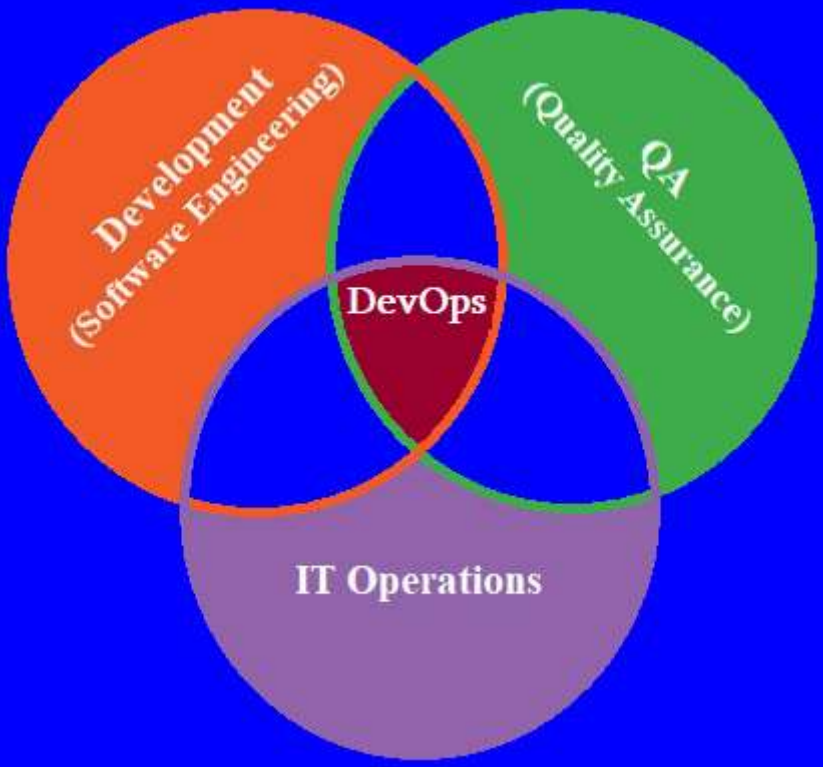
გია სურგულაძე, ეკატერინე თურქია

ინფორმატიკა - „პროგრამული ინჟინერია“

(საბაკალავრო პროექტის მეთოდმითითებანი
„პროგრამულ ინჟინერიაში“)



„სტუ-ს IT- კონსალტინგის სამეცნიერო ცენტრი”

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, ეკატერინე თურქია

ინფორმატიკა - „პროგრამული ინჟინერია“

(საბაკალავრო პროექტის მეთოდური მითითებანი

„პროგრამული ინჟინერიის“ კონცენტრაციით)

დამტკიცებულია:
სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრის“ სარედაქციო კოლეგიის მიერ ოქმი N2, 10.02.2021

თბილისი
2021

უაკ 004.5

წარმოდგენილა საქართველოს ტექნიკური უნივერსიტეტის ინფორმატიკისა და მართვის სისტემების ფაკულტეტის საბაკალავრო პროგრამის „ინფორმატიკა“ დამამთავრებელი ნაშრომის (საბაკალავრო პროექტის) შესრულების მეთოდური მითითებანი „პროგრამული ინჟინერიის“ კონცენტრაციის შესაბამისად. არსებული სილაბუსის საფუძველზე განსაზღვრულია საფინალო პროექტის მოთხოვნები, მისი თეორიული, პრაქტიკული და ექსპერიმენტული საკითხების გადაწყვეტის გათვალისწინებით. განხილულია გამოყენებითი პროგრამული ინჟინერიის სამი მიმართულება: სამაგიდო აპლიკაციების დეველოპმენტი (Desktop-App), ვებ-აპლიკაციების დეველოპმენტი (Web-App) და მობილური აპლიკაციების დეველოპმენტი (Mobile-App). პროგრამული პროდუქტების შექმნის მეთოდოლოგიური საფუძვლებია ობიექტ-ორიენტირებული, უნიფიცირებული მოდელირების ენა (UML) და მოქნილი ტექნოლოგიები (Agile Software Development), რომელთა საფუძველზეც ხორციელდება სხვადასხვა დარგის მხარდამჭერი საინფორმაციო სისტემების პროექტირება და პროგრამული რეალიზაცია. შემოთავაზებულია პროგრამული პროდუქტის აგების სასიცოცხლო ციკლის ეტაპების შესაბამისი საკითხები, პროგრამული სისტემების უსაფრთხოების, მათი ტესტირებისა და ხარისხის შეფასებით, აგრეთვე დისტრიბუციული (საინსტალაციო) პაკეტის შექმნით. წიგნი განკუთვნილია სტუ-ის „ინფორმატიკის“ საბაკალავრო პროგრამის სტუდენტებისათვის პროგრამული ინჟინერიის მიმართულებით.

რეცენზენტი:

პროფ. ნ. ამილახვარი (საქართველოს ტექნიკური უნივერსიტეტი, დოქტორი)

პროფ. დ. გულუა (ბახრეინის უნივერსიტეტი, ინფორმატიკის დოქტორი)

ISBN 978-9941-8-2927-7

სარჩევი

1. შესავალი:
საბაკალავრო პროექტის მიზანი

სტუ-ს „ინფორმატიკის“ საბაკალავრო პროგრამის დამამათავრებელი ნაშრომის შესრულება, „პროგრამული ინჟინერიის“ კონცენტრაციის შესაბამისად, ხელს უწყობს კვალიფიციური კურსდამთავრებულის ჩამოყალიბებას, საპროექტო დავალების დამოუკიდებლად განხორციელე-ბისას უნივერსიტეტში შესწავლილი კურსების ცოდნის ინტეგრირებული გამოყენების საფუძველზე. კერძოდ, ნაშრომში მოითხოვება კომპიუტერული სისტემების დიზაინისა და დეველოპმენტის თანამედროვე პროგრამული პლატფორმების, ენების და ტექნოლოგიების გამოყენების უნარ-ჩვევების გამოვლენა და მათი სრულყოფა. პროგრამული პროდუქტის შექმნის პროექტის გადაწყვეტისას სტუდენტი ატარებს საძიებო, კვლევით და პრაქტიკულ სამუშაოებს, სისტემური ანალიზის, პროგრამების სასიცოცხლო ციკლის ეტაპების და საჭირო რესურსების ეფექტიანი მართვის საფუძველზე.

2. საგნის შესწავლის შედეგად მიღებული ცოდნა და შეძენილი უნარები

საბაკალავრო პროექტის შესრულების პროცესში სტუდენტი ღებულობს ცოდნას და იძენს შემდეგ უნარებს [1]:

1. საპროექტო ობიექტის ბიზნეს-ანალიზის საფუძველზე განსაზღვრავს მიზნობრივი სისტემის/პროდუქტის სავარაუდო ინფრასტრუქტურას და აღწერს მისი კომპონენტების კვლევისა და შესაბამისი ალგორითმების შემუშავების მეთოდებს;
2. განსაზღვრავს პროექტირების/დეველოპმენტის წესებს და სისტემის/პროდუქტის სასიცოცხლო ციკლის საფუძველზე პროგრამირების მეთოდებს და ინსტრუმენტულ საშუალებებს;
3. ასახელებს სისტემის/პროდუქტის ინფორმაციული უსაფრთხოების წესებს;
4. ახდენს სისტემის/პროდუქტის კომპონენტების შერჩევას და მათი თავსებადობის დადგენას, პროექტირებას და ფუნქციური სქემების შედგენას;
5. აყალიბებს დასაბუთებულ დასკვნებს სისტემის/პროდუქტის მუშაობისუნარიანობის, მწარმოებლურობის, საიმედოობის და ხარისხის მდგომარეობის შესახებ;
6. აფასებს სისტემის/პროდუქტის ინფორმაციულ უსაფრთხოებას და ეფექტიანობას;
7. ამზადებს და წარადგენს სისტემაში/პროდუქტში მიმდინარე პროცესების შესახებ წერილობით და ზეპირ ანგარიშს.
8. ახდენს საკუთარი სწავლის პროცესის მართვას შემდგომი სწავლის საჭიროებების დადგენის მიზნით; პროფესიული ცოდნის, გამოცდილების გაღრმავებისა და გაუმჯობესების მიზნით, განსაზღვრავს საკუთარი თვითგანათლების მიმართულებებს;
9. ახდენს ეთიკის ნორმებზე დაფუძნებული პროფესიული საქმიანობის ღირებულებების შეფასებას და სხვებისთვის გაზიარებას.

3. ძირითადი და პროგრამული ინჟინერიის კონცენტრაციით გავლილი აკადემიური კურსები (წინაპირობები)

სტუ-ს „ინფორმატიკის“ მოდიფიცირებული (2018 წ.) და რეაკრედიტაციით განახლებული საბაკალავრო პროგრამებით (2020 წ.) სტუდენტებს გავლილი აქვთ შემდეგი კურსები (სპეციალობის აკადემიური საგნები: ძირითადი და არჩევითი (ა) [1]):

1. შესავალი ინფორმატიკაში
2. კომპიუტერული უნარები
3. დაპროგრამების საფუძვლები
4. მონაცემთა სტრუქტურები და ალგორითმები
5. კომპიუტერული საინჟინრო გრაფიკა
6. კომპიუტერის არქიტექტურის და ორგანიზაციის საფუძვლები
7. კომპიუტერული მათემატიკის საფუძვლები
8. ალბათობის თეორია და გამოყენებითი სტატისტიკა
9. შესავალი მონაცემთა ბაზებში და მათ გამოყენებაში
10. ოპერაციული სისტემების საფუძვლები
11. ობიექტზე ორიენტირებული დაპროგრამება (C++/C#-ის ბაზაზე) (ა)
12. ობიექტზე ორიენტირებული დაპროგრამება (Python-ის ბაზაზე) (ა)
13. ვებ-ტექნოლოგიების საფუძვლები (HTML, CSS, JavaScript)
14. მონაცემთა ბაზების დაპროექტება (SQL-სერვერის და My SQL-ის ბაზაზე) (ა)
15. მონაცემთა ბაზების დაპროექტება (SQL-სერვერის და No SQL-ის ბაზაზე) (ა)
16. პროგრამული ინჟინერიის საფუძვლები
17. ობიექტზე ორიენტირებული დაპროგრამება და აპლიკაციები (C++/C#-ის ბაზაზე)
18. ობიექტზე ორიენტირებული დაპროგრამება და აპლიკაციები (Python-ის ბაზაზე)
19. კომპიუტერული ქსელების საფუძვლები
20. ვებ-აპლიკაციების და მომხმარებელთა ინტერფეისების დაპროგრამება (XML, AJAX, Angular)
21. ინფორმაციული უსაფრთხოების საფუძვლები
22. მულტიპარადიგმული დაპროგრამების საფუძვლები Python ენის ბაზაზე (ა)
23. მულტიპარადიგმული დაპროგრამების საფუძვლები Java ენის ბაზაზე (ა)
24. სისტემების ობიექტზე ორიენტირებული ანალიზი და დაპროექტება
25. დაპროგრამება კომპიუტერული ქსელებისათვის (ა)
26. განაწილებული და პარალელური გამოთვლები (ა)
27. განაწილებული მონაცემთა ბაზების მართვის სისტემა Oracle (ა)
28. აპლიკაციების დაპროგრამება და მონაცემთა მენეჯმენტი
29. კომპიუტერული ქსელების ადმინისტრირება
30. მობილური აპლიკაციების დაპროგრამება Android-თვის (ა)
31. მობილური აპლიკაციების დაპროგრამება iOS-თვის (Swift-ის ენაზე) (ა)
32. საწარმოო პრაქტიკა (ინფორმატიკაში)
33. დიდი მონაცემების შენახვისა და დამუშავების სისტემები
34. ორგანიზაციული კონტენტ-მენეჯმენტის სისტემის დეველოპმენტი (ა)
35. ვებ-დაპროგრამების PHP/MySQL ტექნოლოგია (ა)
36. პროგრამული პროდუქტების დეველოპმენტი
37. პროგრამული სისტემების ინფორმაციული უსაფრთხოება
38. პროგრამული სისტემების დაპროექტებისა და ანალიზის ტექნოლოგიები (CASE, Agile) (ა)
39. ავტომატიზებული მართვის ამოცანების მოდელები და პროგრამული რეალიზება (ა)

4. საბაკალავრო პროექტის თემატიკის ძირითადი მიმართულებები

საბაკალავრო პროექტი „პროგრამული ინჟინერიის“ კონცენტრაციის შესაბამისად სრულდება *გამოყენებითი პროგრამული სისტემების* აგების პროექტის სამი მიმართულებით:

- სამაგიდო აპლიკაცია (Desktop App);
- ვებ-აპლიკაცია (Web App);
- მობილური აპლიკაცია (Mobile App).

პროგრამული აპლიკაციის ტიპი და სირთულე განისაზღვრება პროექტის ხელმძღვანელის მიერ სტუდენტის სასტარტო მონაცემების საფუძველზე, მისი სურვილის გათვალისწინებით (იხ. დანართი-1).

5. საათების განაწილება (სტუდენტის დატვირთვა)

საბაკალავრო პროექტი 10 კრედიტიანია (1 კრ = 25 სთ). პროექტზე ჯგუფისთვის გამოყოფილია 60 სთ და დამოუკიდებელი მუშაობისთვის 186 სთ. აგრეთვე შუასემესტრული (2 სთ) და დასკვნითი გამოცდა (2 სთ).

6. საბაკალავრო პროექტის შესრულების ეტაპები და შინაარსი (სილაბუსის მიხედვით)

ზოგადად, „ინფორმატიკის“ საბაკალავრო პროექტის სილაბუსი მოიცავს შემდეგ ეტაპებს და შინაარსს [1]:

1. საბაკალავრო პროექტის თემატიკის შერჩევა: სისტემა (პროგრამული, აპარატურული, პროგრამულ–აპარატურული), კონკრეტული პროდუქტი (პროგრამა, მოწყობილობა). არჩეული თემატიკის მიხედვით საბაკალავრო პროექტის თავისებურებების და სპეციფიკის გაცნობა. საბაკალავრო პროექტის მიზნის, გაფორმების წესის, შეფასების სისტემისა და კრიტერიუმების გაცნობა.

2. საპროექტო სისტემის/პროდუქტის ობიექტის გაცნობა და შესწავლა. ობიექტის კვლევა და კრიტიკული გააზრება (აღწერა)

3. არსებული ანალოგიური სისტემების/პროდუქტების მოძიება. არსებული ანალოგიური სისტემების/პროდუქტების ფუნქციური და/ან ტექნიკური თვალსაზრისით კვლევა. არსებული ანალოგიური სისტემების/პროდუქტების კრიტიკული ანალიზი;

4. სისტემის/პროდუქტის პროექტირების/დეველოპმენტის მიზნების განსაზღვრა. სისტემის/პროდუქტის შექმნის ეტაპების და სტადიების ფორმირება.

ტექნიკური დავალება, ესკიზური პროექტი, ტექნიკური პროექტი, სამუშაო პროექტი

5. სისტემის/პროდუქტის სისტემური საკითხების და პროექტირებისათვის/დეველოპ-მენტისათვის ტექნიკური წინადადების ჩამოყალიბება: სისტემის/პროდუქტის პროექტირების ტექნიკური დავალების შედგენა. სისტემის/პროდუქტის ესკიზური პროექტი და მისი ტექნიკურ–ეკონომიკური დასაბუთება;

6. პროგრამული სისტემის/პროდუქტის ტექნოლოგიური პლატფორმის კვლევა და შერჩევა: შერჩეული პლატფორმის პროექტის მიზნებთან თავსებადობის დადგენა ან აპარატურული

სისტემის/პროდუქტის კომპონენტების კვლევა, სისტემის კომპონენტების შერჩევა კონკრეტული საჭიროებიდან გამომდინარე. სისტემასთან კომპონენტების თავსებადობის დადგენა;

7. სისტემის/პროდუქტის ფუნქციონირების ალგორითმების შემუშავება: სისტემის/ პროდუქტის ფუქნციონირების თანმიმდევრობის დადგენა. ალგორითმის შედგენა, ალგორითმის სისწორის შემოწმება;

8. სისტემის/პროდუქტის პროექტირება/დეველოპმენტი-1: შერჩეული მეთოდების, ხერხების და ინსტრუმემტული საშუალებების გამოყენებით დაპროექტების/დეველოპმენტის პროცესის განხორციელება;

9. სისტემის/პროდუქტის პროექტირება/დეველოპმენტი-2: შერჩეული მეთოდების, ხერხების და ინსტრუმემტული საშუალებების გამოყენებით დაპროექტების/დეველოპმენტის პროცესის განხორციელება;

10. სისტემის/პროდუქტის პროექტირება/დეველოპმენტი-3; შერჩეული მეთოდების, ხერხების და ინსტრუმემტული საშუალებების გამოყენებით დაპროექტების/დეველოპმენტის პროცესის განხორციელება;

11. სისტემის/პროდუქტის სამომხმარებლო ინტერფეისის პროექტირება/დამუშავება: ინტერფეისის ამოცანების დადგენა, ინტერფეისის ფუნქციონირების ალგორითმის შემუშავება;

12. დაპროექტებული სისტემის/პროდუქტის ინფორმაციული უსაფრთხოების შეფასება. ინფორმაციული უსაფრთხოების უზურნველყოფის ხერხების შერჩევა, ინფორმაციული უსაფრთხოების სისტემის შემუშავება. ინფორმაციული უსაფრთხოების სისტემის შეფასება

13. დაპროექტებული სისტემის/პროდუქტის ეფექტიანობის შეფასება: შესაბამისი მეთოდების შერჩევა სისტემის ეფეფქტიანობის შესაფასებლად. ეფექტიანობის შეფასების ჩატარება;

14. პროექტის ტექსტური და გრაფიკული ნაწილის გაფორმება: ტექსტური და გრაფიკული ნაწილების გაფორმება არსებული მოთხოვნების გატალისწინებით;

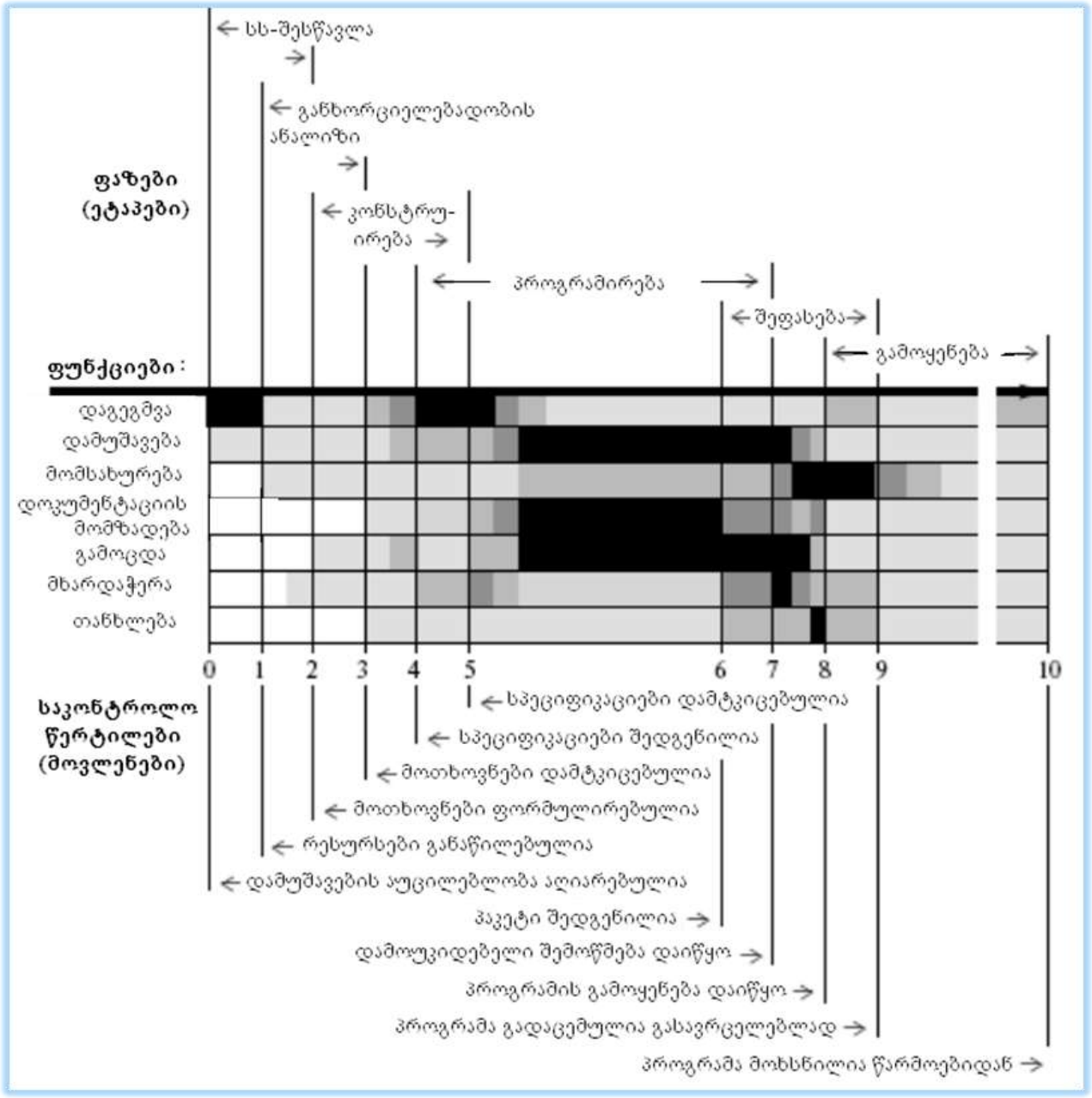
15. პროექტის პრეზენტაციის მომზადება: პრეზემტაციის გრაფიკული ნაწილის და ტექსტების მომზადება.

7. საბაკალავრო პროექტის შესრულების ეტაპები და ამოცანები (პროგრამული სისტემის სასიცოცხლო ციკლის მიხედვით)

გამოყენებითი პროგრამული სისტემების განვითარების (Aapplied Software Development) საბაკალავრო პროექტი, პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის (Software Lifecycle) შესაბამისად, მოიცავს განსაზღვრულ ეტაპებს და ამოცანებს.

მასში აისახება საკვლევი ობიექტის საწარმოო (ბიზნეს) ფუნქციები, რომელთა ანალიზს, მოდელირებას და პროექტირებას ასრულებენ ბიზნეს-ანალიტიკოსები და პროგრამული სისტემის არქიტექტორები, პროექტის მენეჯერის ხელმძღვანელობით.

პროგრამული სისტემების სასიცოცხლო ციკლის ერთ-ერთი ფართოდ გამოყენებული მოდელია განტერის „ფაზები და ფუნქციები“ (ნახ.1). მასში ასახულია ორგანიზაციული და ტექნიკური საწარმოო ფუნქციები. მასზე განთავსებულია საკონტროლო წერტილები, სადაც შედეგების ანალიზის საფუძველზე ხდება მომდევნო პროცედურების შესრულება, მათ შორის იგი ითვალისწინებს იტერაციებსაც (წინა ეტაპებზე დაბრუნება გარკვეული ხარვეზების გასასწორებლად ან ფუნქციონალის გასაფართოვებლად) [2,5].



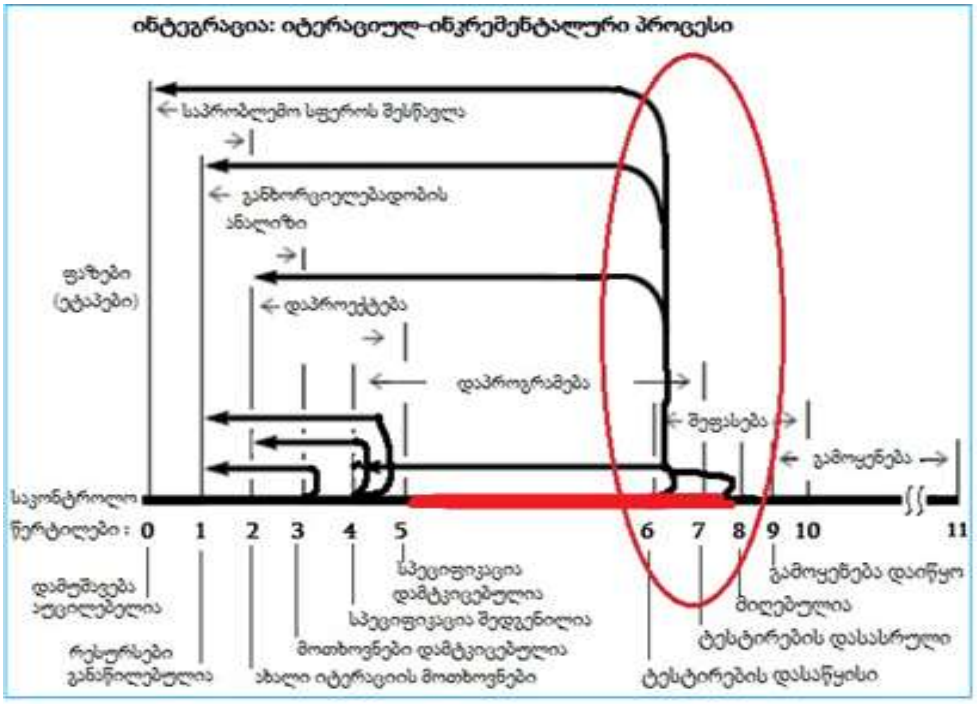
ნახ.1. განტერის მოდელი „ფაზები-ფუნქციები“

განტერის მოდელს აქვს ორი განზომილება:

- ფაზური, რომელიც ასახავს შესრულების ეტაპებს და თანმხლებ მოვლენებს;
- ფუნქციური, სადაც ჩანს, თუ რომელი საწარმოო ფუნქციები სრულდება პროექტის განვითარების პროცესში და როგორია მათი ინტენსივობა თითოეულ ეტაპზე.

ფუნქციების გამოყენების ინტენსივობა ფაზების მიხედვით მოცემულია მუქი ფერით.

მე-2 ნახაზზე ნაჩვენებია სასიცოცხლო ციკლის საკონტროლო წერტილებში (მაგალითად, 3,4,6) *იტერაციული ციკლების* შესაძლებლობები. იტერაციულობა გარდაუვალია რთული პროგრამული სისტემების დასაპროექტებლად. ამიტომაც მიზანშეწონილია ასეთი პროცესების დაგეგმვა და შესრულება, რაც ხელს უწყობს პროგრამული პროდუქტის საბოლოო ვერსიის ხარისხის სრულყოფას (სისტემის ვალიდაციის თვალსაზრისით) და დამკვეთის ინტერესების მაქსიმალურად გათვალისწინებას.



ნახ.2. იტერაციულობა განტერის მოდელში

მომდევნო ეტაპებზე განვიხილავთ საბაკალავრო პროექტის შესრულების დროს აუცილებელი ამოცანების გადაწყვეტის საკითხებს. კერძოდ, კონკრეტული საკვლევი ობიექტის (საბაკალავრო პროექტის თემა) მხარდამჭერი კომპიუტერული სისტემის ბიზნეს-მოთხოვნილებათა და რეალიზაციის ინფრასტრუქტურის განსაზღვრის, ობიექტ-ორიენტირებული ანალიზის, დაპროექტების, პროგრამული დეველოპმენტის, ტესტირების, დანერგვისა და ექსპლუატაციის პროცესში თანხლების ამოცანების შესრულებას [3,8].

განიხილება გამოყენებითი პროგრამული სისატემების აგების ორი მეთოდოლოგია: UML – დიდი პროექტებისათვის და Agile – სწრაფად გადასაწყვეტი პროექტებისათვის, ან მათი კომპრომისული ვარიანტი [4].

7.1. UML მეთოდოლოგია და მისი ინსტრუმენტული საშუალებები

UML (Unified Modeling Language) არის განაწილებული მართვის საინფორმაციო სისტემების დაპროექტების მეთოდოლოგიური საფუძველი (ამერიკული ტექნოლოგია).

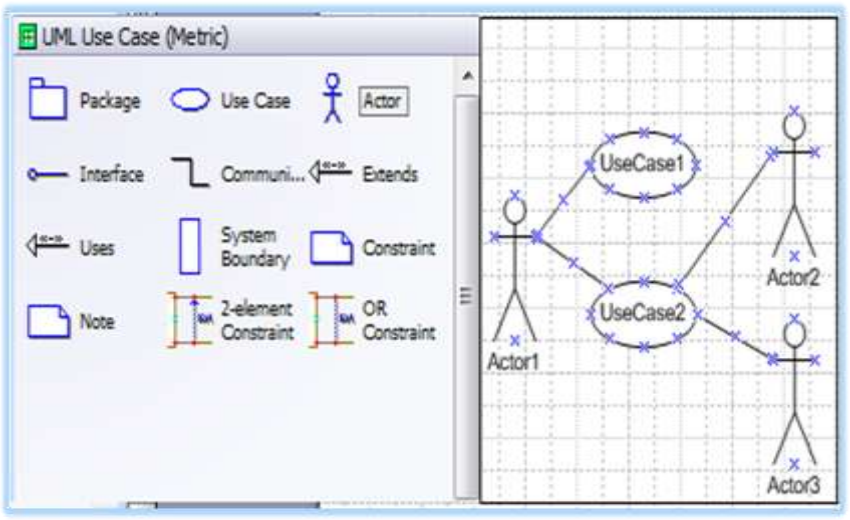
UML/1-ის ოთხეტაპიანი კლასიკური მოდელი შედგება 4 წყვილი დიაგრამით. მათგან 4 სტატიკური და 4 - დინამიკური (ნახ.3). შეიძლება UML/2 ვერსიის გამოყენებაც, რომელშიც მეტი ფუნქციონალობა და მოდელია [5,15].

უნიფიცირებული მოდელირების ენის ეტაპები და ამოცანები ზუსტად შეესაბამება 1-ელ ნახაზზე განხილულ პროგრამული პროდუქტების სასიცოცხლო ციკლის ეტაპებს.

ნახ.3

7.1.1. გამოყენებით შემთხვევათა (Use Case) დიაგრამა

საკურსო პროექტი სრულდება კონკრეტული კვლევის ობიექტის (დამკვეთის) ფუნქციური დეპარტამენტების თანამშრომლებისთვის. საჭიროა განისაზღვროს როლების (Actors) და მათი ფუნქციების (UseCases) დიაგრამა, UML ტექნოლოგიით ესაა გამოყენებით შემთხვევათა (პრეცედენტების) UseCase დიაგრამა (ნახ.4). აქ UseCase1 ეკუთვნის მხოლოდ პირველ როლს, ხოლო UseCase2-ის შესასრულებლად ორივე როლი მონაწილეობს.



ნახ.4. UseCase დიაგრამის აგების ინტერფეისი (Visual Studio 2015)

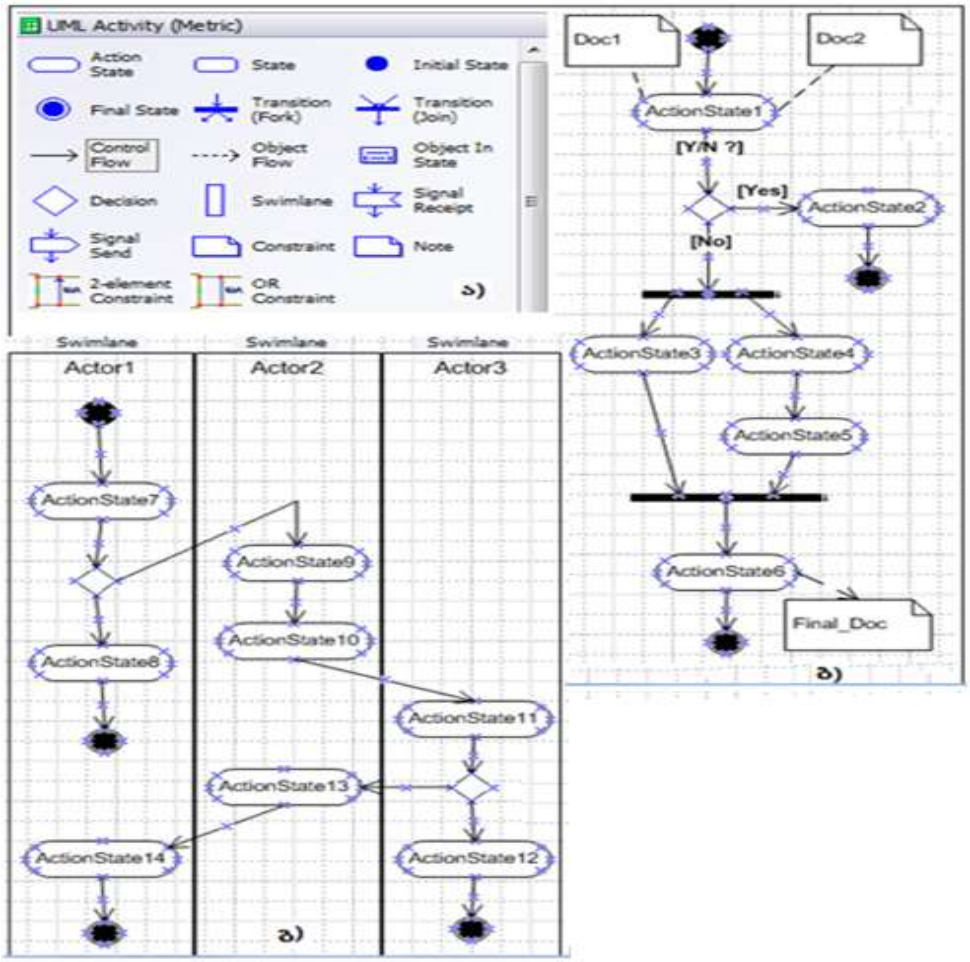
7.1.2. აქტიურობათა Activity დიაგრამა

კვლევის ობიექტის ბიზნესპროცესებისა და ბიზნესწესების გაცნობის, ანალიზის და სტრუქტურული ფორმალიზაციის საფუძველზე აიგება აქტიურობის (ქმედებათა) დიაგრამა. იგი კონკრეტული როლის (როლების) კონკრეტული ფუნქციაა, რომელიც შედეგება იერარქიულად სივრცესა და დროში დალაგებული მიმდევრობით ან პარალელურად შესასრულებელი სუბქმედებებისგან. აქვს ერთი დასაწყისი და რამდენიმე შესაძლო დასასრული, ბიზნესწესებით განსაზღვრული განშტოების ან შეერთების პროცედურები, საწყისი, შუალედური ან საშედეგო დოკუმენტაცია და ა.შ. საილუსტრაციო მაგალითი მოცემულია მე-5 ნახაზზე.

ქმედებათა დიაგრამა მიეკუთვნება პროცესების აღწერის დინამიკურ მოდელს, იგი ასახავს საკვლევი ობიექტის ქცევას. ასეთი დინამიკური მოდელების პროცესების გამოსაკვლევად შეიძლება გამოიყენებულ იქნას ავტომატიზებული მართვის (მაგალითად, გრაფო-ანალიზური) მოდელები და სხვა ეკონომიკურ-მათემატიკური მეთოდები).

საბოლოოდ, ამ ორი სახის (UseCase, Activity) დიაგრამათა ერთობლიობის ანალიზის საფუძველზე კეთდება დასკვნები საავტომატიზაციო ობიექტის მართვის სისტემის ფუნქციური და არაფუნქციური მოთხოვნილებების განსაზღვრის შესახებ.

ამავდროულად დგება მომავალი პროგრამული სისტემის შექმნის *ტექნიკური დავალება*, შესაბამის ეკონომიკურ გაანგარიშებებთან ერთად, რომელიც დამკვეთ ორგანიზაციის ხელმძღვანელობასთან კონსულტაციების შემდეგ, ორმხრივად მტკიცდება. ამის შემდეგ იწყება ობიექტორიენტირებული ანალიზის ეტაპი, რომელზეც აიგება სისტემის მომხმარებელთა ინტერაქტიული სქემები: მიმდევრობითი და თანამოქმედების დიაგრამათა სახით.



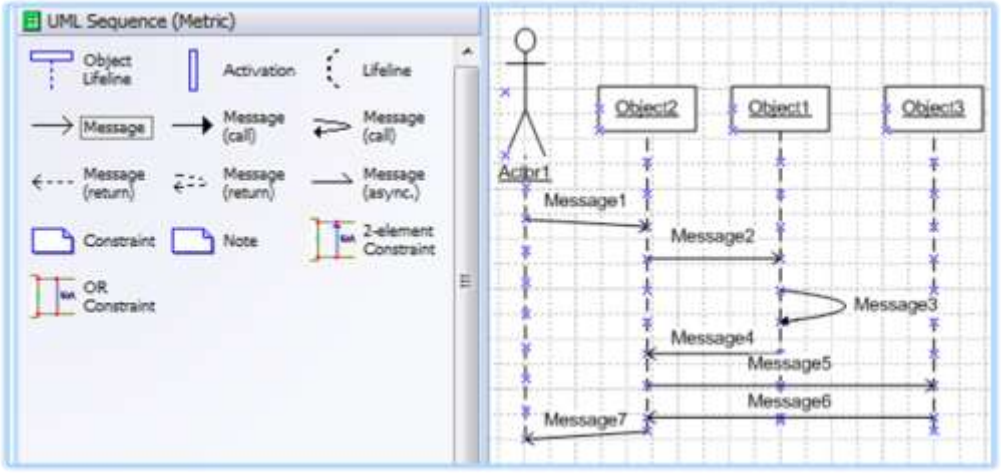
ნახ.5. Activity დიაგრამის:

ა) ინსტრუმენტების პანელი; ბ) ქმედებათა სქემა (მარტივი);

გ) ქმედებათა სქემა (მართვის სფეროების ან როლების ბილიკებით)

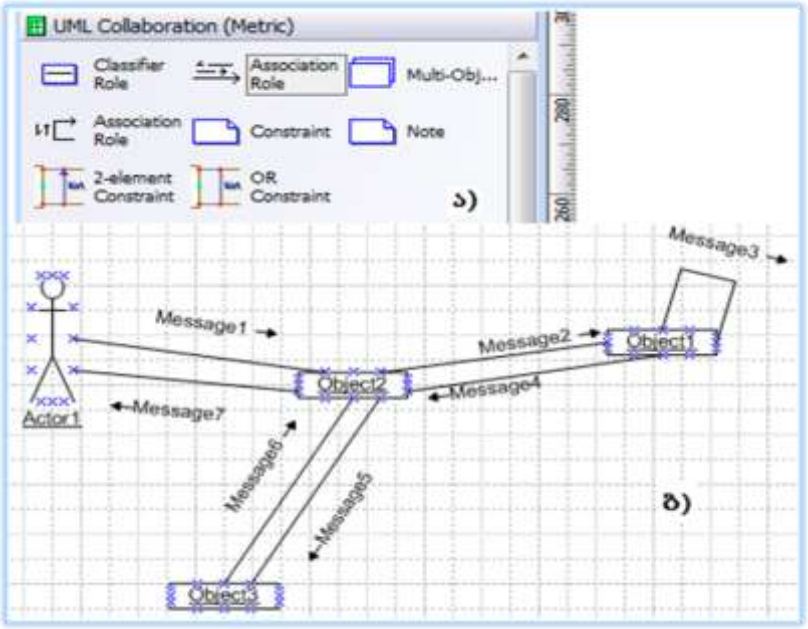
7.1.3. მიმდევრობითობის (Sequence) და თანამოქმედების (Collaboration) დიაგრამები

მიმდევრობითობის დიაგრამა აღწერს საპრობლემო სფეროს რომელიმე თანამშრომლის (როლის) კონკრეტული ამოცანის *კომპიუტერზე შესრულების სცენარს.* აქ ხდება როლის სისტემასთან ურთიერთქმედების ბიზნესპროცესის ქმედებათა და მათი მაინიცირებელ, სინქრონულ ან ასინქრონულ შეტყობინებათა დროში მიმდევრობით განლაგება (ნახ.6).



ნახ.6.equence დიაგრამისაგების ინტერფეისი

მე-7 ნახაზზე ნაჩვენებია თანამოქმედების დიაგრამის აგების ინსტრუმენტების პანელი (ა) და თვით დიაგრამა (ბ), რომელიც შესაბამისი მიმდევრობითობის დიაგრამის ტრანსფორმაციით იქნა მიღებული.



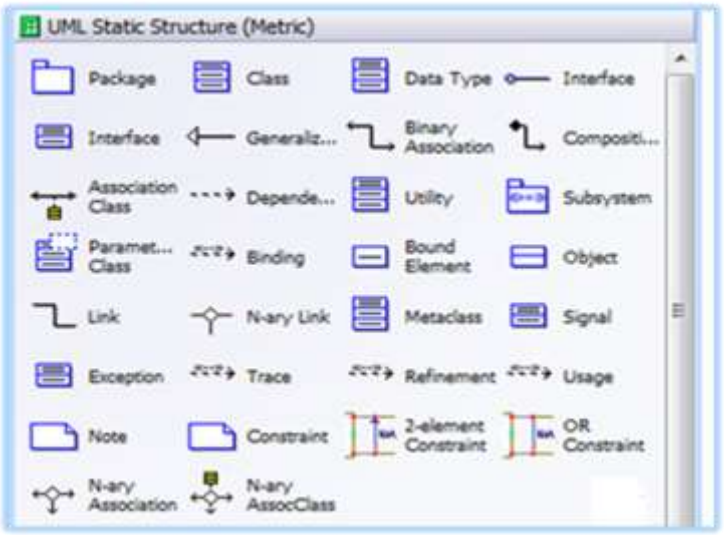
ნახ.7. Collaboration დიაგრამის აგების ინტერფეისი (ა) და სქემის მაგალითი (ბ)

აქ შეტყობინებათა და მონაცემთა გაცვლის მიმდევრობა არაა დროში დალაგებული, სამაგიეროდ ჩანს კლასის ობიექტებს შორის კავშირებისა და ინფორმაციული ნაკადების გაცვლის სემანტიკა.

CASE ტექნოლოგიის მრავალ პროდუქტში (მაგალითად, ფირმა Rational Rose) თანამიმდევრობითობის დიაგრამის აგება ხდება ავტომატიზებულ რეჟიმში. ანუ ჯერ აიგება მიმდევრობითობის დიაგრამა და შემდეგ ინსტრუმენტის რომელიმე სწრაფი ღილაკით (მაგალითად, F5) სისტემა თვითონ ააგებს თანამოქმედების დიაგრამას].

7.1.4. კლასი (Class) და კლასთა-ასოციაციის (Class-Assotiation) დიაგრამა

➢ ***კლასი*** არის ერთგვაროვან ობიექტთა ერთობლიობის სტრუქტურა. მაგალითად, ყველა მოქალაქე (ზოგადად Person), სტუდენტები, ლექტორები, ავტომანქანები, ცხოველები და ა.შ.



ტერმინი „კლასიფიკაცია“ სწორედ განსახილველი სფეროს ობიექტების სისტემატურ მოწესრიგებას ემსახურება (*გენერალიზაცია* – იერარქიაში განზოგადება ზევით და *სპეციფიკაცია* – იერარქიაში დეტალიზაცია ქვევით). MsVisio-ში კლასებისა და კლასთაშორის კავშირების მოდელირებისათვის გამოიყენება Static Structure ინსტრუმენტების პანელი (ნახ.8).

ნახ.8

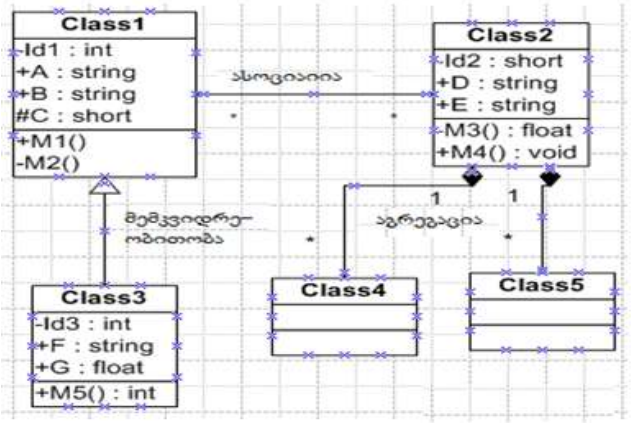
ობიექტორიენტირებული მოდელირების და პროგრამირების გაგებით, კლასი არის „დასახელების“, „კლასის მონაცემებისა“ და „კლასის მეთოდების“ ინკაფსულაცია. მართვის სფეროს შესაბამისი კლასი, ზოგადად ასე უნდა გამოიყურებოდეს (ნახ.9). კლასის ატრიბუტებს შეესაბამება მონაცემთა გარკვეული ტიპი (int, float, string ან სხვ.) და „ხილვადობის“ ანუ მონაცემთა წვდომის მოდიფიკატორები („-“ private, „+“ public, „#“ protection). ასევეა კლასის მეთოდებისთვისაც.

ნახ.9. ინკაფსულაცია

კლასის მეთოდები (ფუნქციები) ის პროგრამული მოდულებია, რომლებიც ამუშავებს კლასის მონაცემებს. მათი ინიციალიზაცია ხდება გარედან შემოსული შეტყობინებით.

➢ **Class-Assotiation დიაგრამა**

კლასთაშორისი კავშირები შეიძლება იყოს: მემკვიდრეობითი, აგრეგატული, რელაციური და ასოციაციური (ნახ.10):

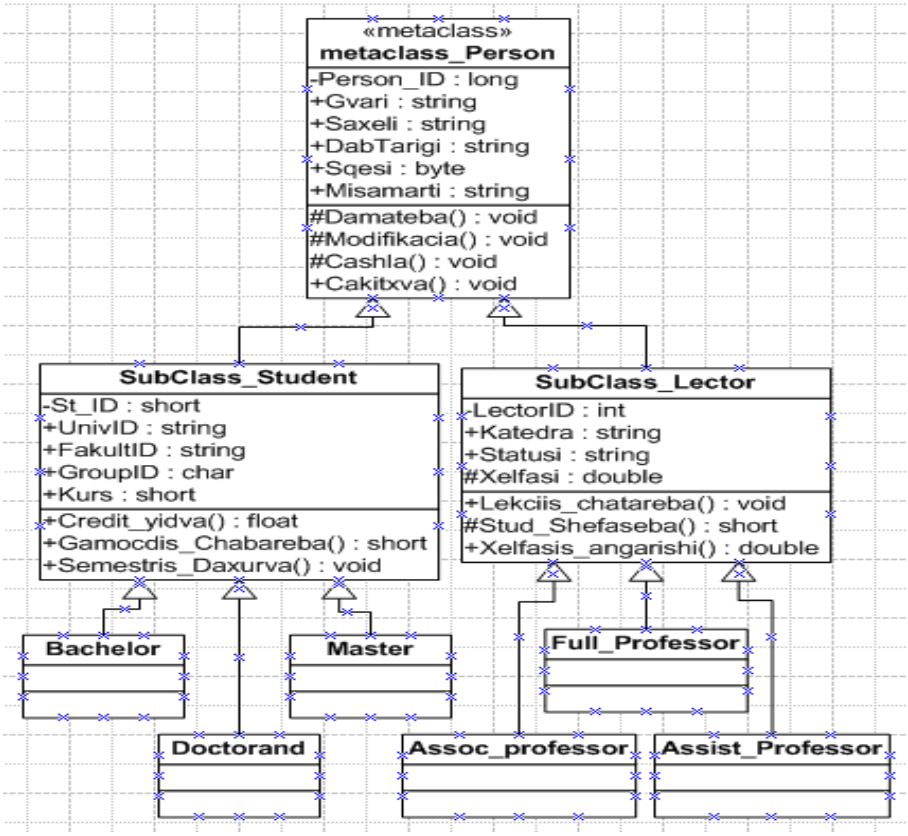


ნახ.10. კლასთაშორისი კავშირების დიაგრამა

- მემკვიდრეობითი (Generalization) ასახავს „გენეტიკურ“, განზოგადოებულ კავშირებს კლასებს შორის. ასეთ დროს ერთი კლასი („შვილი“) მთლიანად იღებს მეორე კლასის („მშობელი“) ყველა ატრიბუტს, მეთოდს და კავშირს;
- აგრეგირებული (Aggregation) ნიშნავს კავშირს „მთელი“–„ნაწილი“. მაგალითად, „ავტომობილი“ – „ძარა, ძრავი, საბურავები და სხვ.“;
- ასოციაციური (Assotiation) ნიშნავს სემენტიკურ კავშირს კლასებს შორის. ის შეიძლება გამოისახოს ერთ- ან ორმიმართუ-ლებიანი ხაზით. ისარი გვიჩვენებს შეტყობინების გადაცემის მიმართულებას. ასოციაციური კავშირის რეალიზება ხდება ერთ კლასში დამატებით მეორე კლასის ატრიბურტის ჩასმით.
- რელაციური (Dependency) ნიშნავს ერთი კლასის დამოკიდებულებას მეორეზე. იგი ერთმიმართულებიანი წყვეტილი ისრით გამოიხატება. მასში დამატებითი დამაკავშირებელი ატრიბუტები არ გამოიყენება.

მე-11 ნახაზზე ილუსტრირებულია კლასთა ასოციაციის დიაგრამა მემკვიდრეობითი კავშირების საფუძველზე. ისარი მიმართულია „შვილიდან“ „დედისკენ“, რაც მათ ცალსახა დამოკიდებულებაზე

მეტყველებს. „შვილს“ ჰყავს ერთი „დედა“, ხოლო „დედას“ შეიძლება ჰყავდეს რამდენიმე „შვილი“, ამიტომაც ეს არაა ცალსახა.



ნახ. 11. Class-Assotiation დიაგრამა მემკვიდრეობითი კავშირებით

მშობელი კლასი ლიტარატურაში ზოგჯერ „მეტაკლასად“ (MetaClass) მოიხსენიება, რომელიც შედგება ქვეკლასებისაგან (SubClasses). შეიძლება იერარქიაში ქვეკლასი იყოს მის ქვევით მდგარი კლასისათვის მეტაკლასი. მაგალითად, SubClass_Student არის ქვეკლასი MetaClass_Person კლასისათვის და, ამავდროულად იგი არის მეტაკლასი სამი ქვეკლასისთვის: Bachelor, Master და Doctorand. იგივე შეიძლება ითქვას კლასებისათვის:

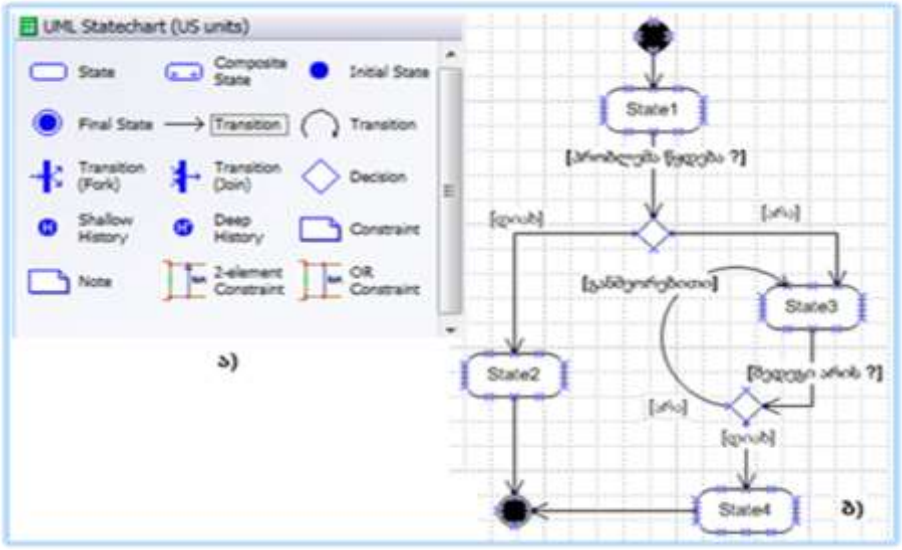
Metaclass_Person <- SubClass_Lector <- {Full_Professor, Assoc_Professor, Assist_Professor},

სადაც როლები ასეა განაწილებული: „მშობელი“-Person, „შვილი“-Lector და „შვილიშვილები“ Full_Professor, Assoc_Professor, Assist_Professor.

- **მდგომარეობათა (Statechart) დიაგრამა**

ყოფაქცევის დიაგრამებიდან არსებობს კიდევ ერთი *კლასების მდგომარეობათა* დიაგრამა - Statechart-D. იგი აღწერს ქმედებებს, ობიექტთა მდგომარეობებს, მდგომარეობათა გადასვლებს და მოვლენებს. მე-12-ა,ბ ნახაზებზე ნაჩვენებია ინსტრუმენტული პანელისა და მდგომარეობათა დიაგრამის ფრაგმენტი ზოგადი მაგალითისთვის.

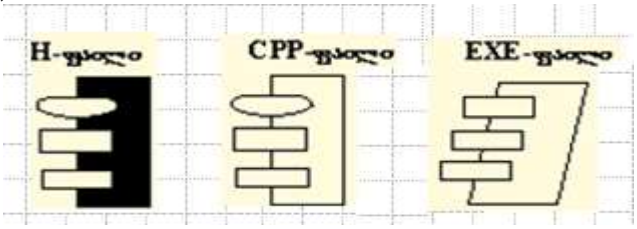
მისი გამოყენება ყველა კლასისათვის არაა საჭირო. აუცილებელია მხოლოდ მაშინ, როდესაც კლასი შეიძლება იმყოფებოდეს რამდენიმე მდგომარეობაში და თითოეულ მათგანში მისი ქცევა უნდა იყოს სხვადასხვანაირი.



ნახ.12. Statechart-ის პანელი (ა) და ზოგადი დიაგრამა (ბ)

- **კომპონენტების (Components) დიაგრამა**

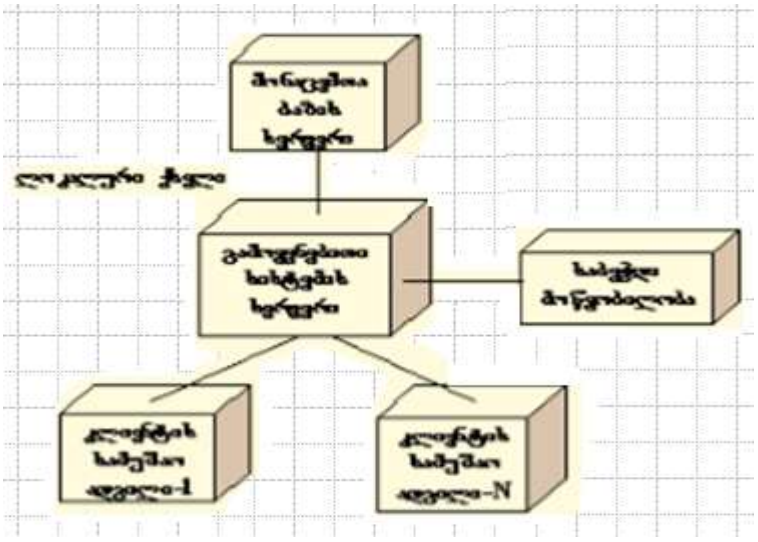
პროგრამული რეალიზაციის (development-ის) ეტაპზე აიგება კომპონენტების დიაგრამა (Component-D), რომელშიც იგულისხმება პროგრამული კოდების (მაგალითად: .cs, .cpp, .h, .dll, .exe და ა.შ.) დამუშავება (ნახ.13).



ნახ.13. კომპონენტების დიაგრამა

- **განთავსების (Deployment) დიაგრამა**

განაწილებული სისტემის რეალიზაციის ბოლო ეტაპზე აიგება განთავსების დიაგრამა (Deployment-D), რომელიც აღწერს კომპონენტების განაწილებას „კლიენტ-სერვერის" ქსელში (ნახ.14).



ნახ.14. განთავსების დიაგრამა

7.1.5. კლასების დიაგრამიდან პროგრამული კოდის გენერაცია

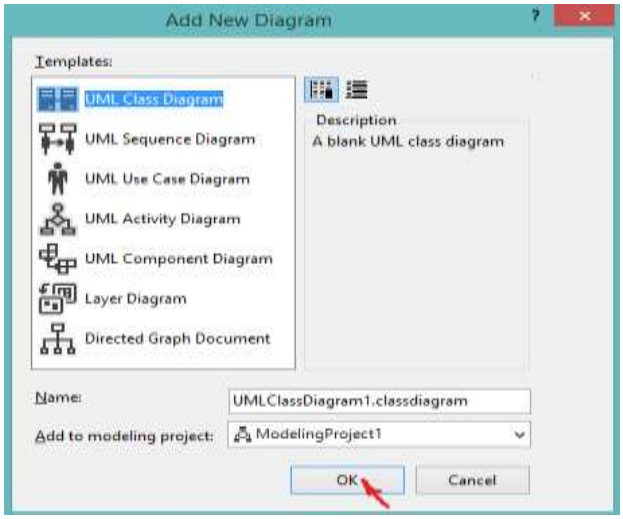
კომპიუტერული სისტემის დაპრექტების ეტაპის შემდეგ საჭიროა პროგრამული კოდის გენერაცია CASE (Computer-aided software engineering) ინსტრუმენტის გამოყენებით.

თანამედროვე CASE-ტექნოლოგიები, რომლებიც სისტემების დაპროგრამების ავტომატიზაციაზეა ორიენტირებული, მაგალითად, Rational Rose, Visual Paradigm, Enterprise Architect და მრავალი სხვ., ახორციელებს რევერსული დაპროგრამების კონცეფციას. ანუ კლასების დიაგრამიდან შესაძლებელია პროგრამული კოდის გენერაცია და პირიქითაც, კოდიდან ავტომატურად აიგება გრაფიკული დიაგრამა. მაგალითად, Visual Studio .NET Framework 2015-ის პლატფორმაზე განვიხილოთ ეს ამოცანა.

შევქმნათ პროექტი მოდელირების მიზნით. VS 2015-ის მთავარ მენიუში ავირჩიოთ

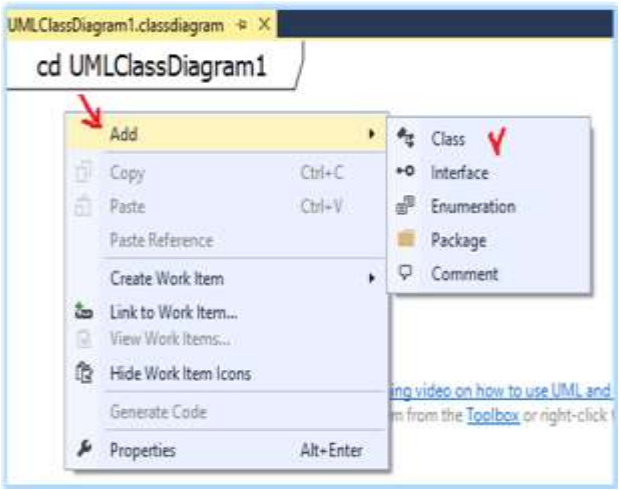
File / NewProject და პროექტის ტიპი: Modeling Project

პროექტში ახალი UML-დიაგრამის დასამატებლად ავირჩიოთ მენიუს პუნქტი Architecture, რომელიც სპეციალურად გამოიყენება პროგრამის არქიტექტურის წარმოსადგენად UML-მოდელის სახით და მის შიგნით პუნქტი Add new Diagram. მიიღება ფანჯარა (ნახ.15), რომელშიც ვირჩევთ UML Class Diagram.



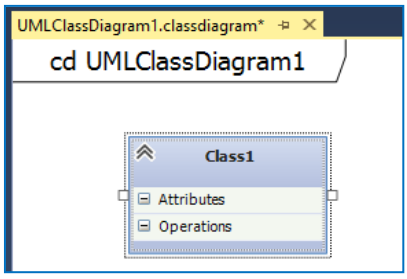
ნახ.15. ახალი UML-დიაგრამის არჩევა

შევქმნათ ახალი კლასთა დიაგრამა, თავიდან ცარიელი. კლასის დასამატებლად დიაგრამაზე ვირჩევთ პუნქტს Add / class (ნახ.16).



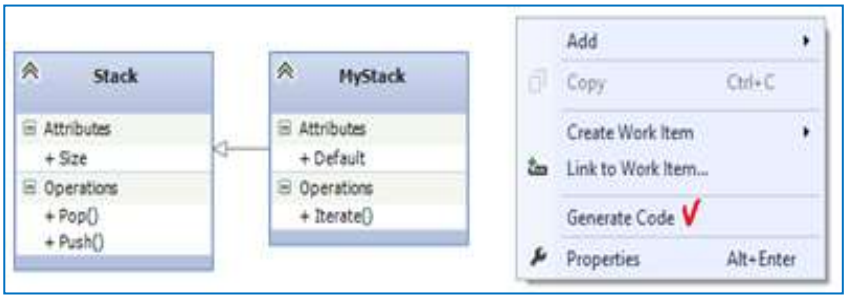
ნახ.16. დიაგრამაზე ახალი კლასის დამატება

ახალი კლასი ემატება ავტომატურად სახელით Class1. იგი შედგება ატრიბუტებისა და ოპერაციებისაგან (ნახ.17).



ნახ.17. კლასი ატრიბუტებითა და ოპერაციებით

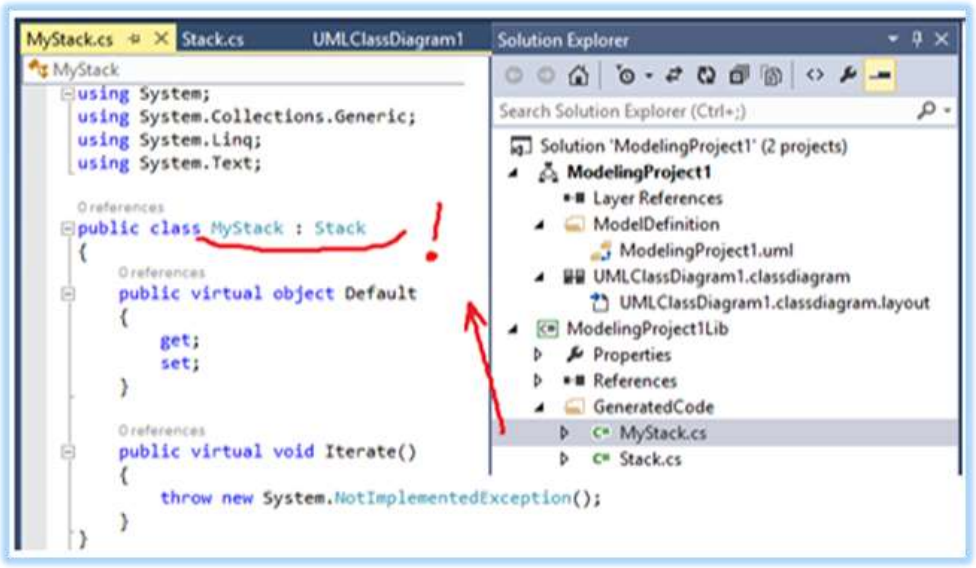
Add / Inheritance - არჩევით დიაგრამაზე დაემატა მემკვიდრეობითობის კავშირი. საბოლოოდ მიიღება მე-18 ნახაზზე ნაჩვენები კლასთა დიაგრამა.



ნახ.18. კლასთა-ასოციაციის დიაგრამა და კოდის გენერირება

კოდის გენერაციისათვის ვირჩევთ შაბლონს - ClassTemplate. სისტემა ავტომატურად ქმნის C# კოდებს თითოეული კლასისთვის, რაც გამოჩნდება Solution Explorer-ში. პროგრამაში ასახულია ასევე კლასთა მემკვიდრეობითობის კავშირიც (ნახ.19).

Public class MyStack : Stack



ნახ.19. კლასიდან გენერირებული Stack.cs კოდი

შენიშვნა: პროგრამული კოდების აგება საბაკალავრო პროექტში შესაძლებელია უშუალოდ რომელიმე პროგრამული ენის საფუძველზე, მაგალითად, C++, C#, Python, JavaScript, PHP და სხვ., იმ მეთოდებისა და ინსტრუმენტული საშუალებების გამოყენებით, რომელსაც სტუდენტი აირჩევს თავისი პროექტის რეალიზაციის შესაბამისად.

7.2. Agile მეთოდოლოგია, მისი მეთოდები და ინსტრუმენტული საშუალებები

დაპროგრამების მოქნილი მეთოდოლოგია (Agile Software Methodology) განიხილავს განსხვავებულ Agile-მეთოდებს, როგორიცაა მაგალითად, Extreme Programming (XP), Scrum, Kanban/Lean, Agile Unified Process (AUP), Dynamic Systems Development Method (DSDM), Disciplined Agile Delivery (DAD), Feature-Driven Development (FDD), Test-Driven Development (TDD), Rapid Application Development (RAD) და Scaled Agile Framework (SAFe) [4,25].

მოქნილი დამუშავების მანიფესტი და პრინციპები მოიცავს მაღალი დონის კონცეფციებს იმის შესახებ თუ როგორ უნდა განხორციელდეს პროგრამული უზრუნველყოფის დამუშავების პროცესი, რათა წარმატებით დასრულდეს პროექტი, შეიქმნას სამუშაო გუნდები, რომლებშიც სასიამოვნო და საინტერესო იქნება მუშაობა. Agile მეთოდოლოგიის *პრინციპებია* [26]*:*

1. კლიენტის დაკმაყოფილება ღირებული პროგრამული უზრუნველყოფის ადრეული და უწყვეტი მიწოდების მეშვეობით;
2. მოთხოვნილებათა ცვლილებების მისალმება სამუშაოს ბოლო ეტაპზეც კი (ეს ზრდის საშედეგო პროდუქტის კონკურენტუნარიანობას);
3. სამუშაო პროგრამული უზრუნველყოფის ხშირი მიწოდება დამკვეთზე (ყოველთვიური, კვირეული ან უფრო ხშირი);
4. დამკვეთის ხშირი, ყოველდღიური კონტაქტი მიმწოდებელთან პროექტის შესრულების მთელ მანძილზე;

5. პროექტზე მუშაობენ მოტივირებული პირები, რომლებიც უზრუნველყოფილია მუშაობის საჭირო პირობებით, მხარდაჭერითა და ნდობით;

6. ინფორმაციის გადაცემის რეკომენდებული მეთოდი – პირადი საუბრები (პირისპირ);

7. მომუშავე პროგრამული უზრუნველყოფა – პროგრესის საუკეთესო საზომია. პროექტების მიზანია პროგრამული სისტემის შექმნა და არა გეგმებისა და დოკუმენტაციის. აპლიკაციის მუშაობისუნარიანობის შეფასებით შეიძლება პროექტის პროგრესის ობიექტური გაზომვა;

8. სპონსორებს, მიმწოდებლებსა და მომხმარებლებს უნდა ჰქონდეთ მხარდაჭერის შესაძლებლობა მუდმივი ტემპის შესანარჩუნებლად გაურკვეველი ვადით;

9. მუდმივი ყურადღება ტექნიკური ოსტატობის (უნარების) სრულყოფას და მოსახერხებელ დიზაინს;

10. სიმარტივე – ხელოვნება ზედმეტი სამუშაოს შესრულების გარეშე. არაა საჭირო რთული უნივერსალური გადაწყვეტების მიღება, თუ ამის ცხადი აუცილებლობა არაა;

11. საუკეთესო ტექნიკური მოთხოვნები, დიზაინი და არქიტექტურა გამოსდის თვითორგანიზებულ გუნდს;

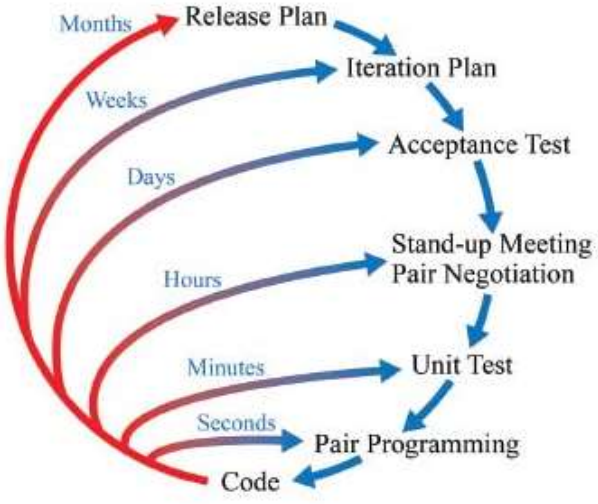
12. მუდმივი ადაპტაცია ცვალებად გარემოებებისადმი. იტერაციული სასიცოცხლო ციკლი ბაზირებულია მართვაზე უკუკავშირით, რომლის მნიშვნელოვანი ელემენტია შედეგების ანალიზი, უკუკავშირის განხორციელება და პროცესის სრულყოფა.

7.2.1. ექსტრემალური პროგრამირება

ექსტრემალური პროგრამირება (XP) არის პროგრამული უზრუნველყოფის დამუშავების მეთოდიკა, რომლის მიზანია პროგრამული აპლიკაციის ხარისხის გაუმჯობესება და დამკვეთ-მომხმარებელთა მოთხოვნების ცვილებებზე რეაგირება. მეთოდოლოგიამ სახელი მიიღო იმ იდეიდან, რომ პროგრამული უზრუნველყოფის ინჟინერიის ტრადიციული პრაქტიკის სასარგებლო ელემენტები სრულდება „ექსტრემალურად სწრაფ" დროში.

ექსტრემალური პროგრამირების საუკეთესო პრაქტიკებიდან ძირითადად შეიძლება განვიხილოთ რამდენიმე. მაგალითად, ესენია უწყვეტი ინტეგრაცია, ტესტირება (მათ შორის, ტესტირებაზე ორიენტირებული დეველოპმენტი და ქცევაზე ორიენტირებული დეველოპმენტი), რეფაქტორინგი, წყვილში მუშაობა და კოლექტიური საკუთრება.

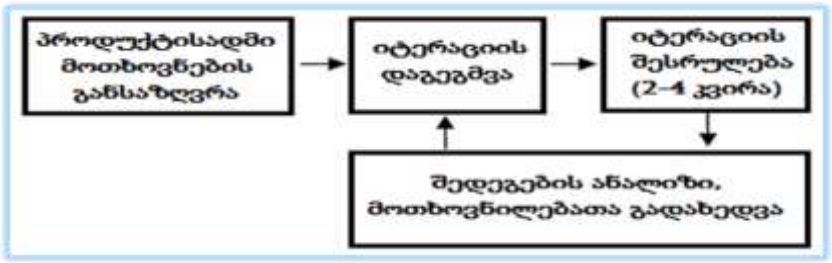
ზოგიერთი გუნდი იყენებს XP-ის სხვა პრაქტიკას, მაგალითად, წყვილთა პროგრამირებას და სისტემის მეტაფორებს. მიზანი არის ის, რომ ყველა დეველოპერს გააცნონ შინაარსობრივად (მაგალიტად, ტექსტურად) სისტემის საერთო ხედვა, რომელიც ემთხვევა ამ სისტემის მომხმარებელთა ხედვას. ამ მიზნით, ექსტრემალური პროგრამირება უპირატესობას ანიჭებს მარტივ დიზაინს, საერთო მეტაფორებს, მომხმარებლებისა და პროგრამისტების თანამშრომლობას, ხშირ სიტყვიერ კომუნიკაციას და უკუკავშირს (ნახ.20).



ნახ.20. დაგეგმვის / უკუკავშირის მარყუჟები

7.2.1. Scrum მეთოდი

Scrum მეთოდი საშუალებას იძლევა პროექტები დამუშავდეს მოქნილად (სწრაფად) მცირე გუნდის მიერ (5-9 კაცი გუნდში) [4,5,27,28]. დამუშავების პროცესი იტერაციულია და დიდ თავისუფლებას აძლევს გუნდს. ამასთანავე, მეთოდი ძალზე მარტივია და შესასწავლად ადვილი, ამიტომაც პრაქტიკაში ადვილად გამოყენებადია (ნახ.21).



ნახ.21. Scrum მეთოდის ბიჯები

თავიდან განისაზღვრება მოთხოვნები მთლიანი პროდუქტისათვის. შემდეგ ამოირჩევა მათგან ყველაზე აქტუალურები და შედგება პირველი (მომდევნო) იტერაციის გეგმა. იტერაციის პერიოდში გეგმა არ იცვლება (ეს ხელს უწყობს დამუშავების პროცესის სტაბილობას), მისი ხანგრძლიობა 2-4 კვირაა. იტერაცია სრულდება პროდუქტის სამუშაო ვერსიის შექმნით, რომელიც შეიძლება გადაეცეს დამკვეთს, მოხდეს მისი დემონსტრირება, თუნდაც მინიმალური ფუნქციური შესაძლებლობებით.

ამის შემდეგ ხდება შედეგების განხილვა და პროდუქტისადმი მოთხოვნილებათა კორექტირება. ეს მოსახერხებელია, რადგან არა მხოლოდ ზუსტდება პროდუქტის ფუნქციები, არამედ დამკვეთს შეუძლია მისი გამოყენებაც. შემდეგ იგეგმება ახალი იტერაცია და ყველაფერი მეორდება.

იტერაციის შიგნით მთლიანად მუშაობს გუნდი. Scrum აქ როლებს არ განსაზღვრავს. მენეჯმენტთან და დამკვეთთან სინქრონიზაცია ხდება იტერაციის დასრულების შემდეგ. იტერაცია შეიძლება შეწყდეს მხოლოდ განსაკუთრებულ შემთხვევებში.

როლები. Scrum მეთოდში არის სამი როლი:

- *პროდუქტის მფლობელი (Product Owner)* – ესაა პროექტის მენეჯერი, რომელიც წარმოადგენს დამკვეთის ინტერესებს. მისი მოვალეობაა პროდუქტის საწყისი მოთხოვნების (Product Backlog) განსაზღვრა, მათი დროულად კორექტირება, პრიორიტეტების, ჩაბარების ვადების დადგენა და სხვ. იგი არ მონაწილეობს უშუალოდ იტერაციის შესრულებაში;
- *Scrum-ოსტატი (Scrum Master)* – უზრუნველყოფს გუნდის მაქსიმალურ მწარმოებლურობასა და პროდუქტიულობას, როგორც Scrum-პროცესის შესასრულებლად, ისე სამეურნეო და ადმინის-ტრაციული ამოცანების გადასაწყვეტად. კერძოდ, მისი ამოცანაა გუნდის დაცვა იტერაციის დროს ყოველგვარი გარე ზემოქმედებიდან;

Scrum-გუნდი (Scrum Team) – ჯგუფია, რომელიც შედგება 5-9 დამოუკიდებელი, ინიციატივიანი პროგრამისტ-დეველოპერებისგან. გუნდის *პირველი ამოცანაა* იტერაციისათვის რეალურად მიღწევადი და პროექტისთვის პრიორიტეტული დავალებების განსაზღვრა (Project Backlog-ის საფუძველზე და პროდუქტის მფლობელისა და Scrum-ოსტატის აქტიური მონაწილეობით). *მეორე ამოცანაა* ამ დავალებების უეჭველი შესრულება დადგენილ ვადებში და მოთხოვნილი ხარისხით. მნიშვნელოვანია, რომ გუნდი თვითონ მონაწილეობს დავალებათა დასმის პროცესში და თვითონ წყვეტს მათ. აქ შეთავსებულია თავისუფლება და პასუხისმგებლობა, დონეზეა მოვალეობათა დისციპლინა [4,5].

7.2.2. Kanban მეთოდი

Kanban – არის პროგრამული უზრუნველყოფის დამუშავების ეკონომიური მეთოდი (Lean method of software development) [4,29]. სიტყვა kanban იაპონურად არის „სასიგნალო ბარათი“ (kan - სიგნალი, ban - ბარათი). მეთოდს აქვს პრინციპები და ძირითადი პრაქტიკები. განვიხილთ ისინი.

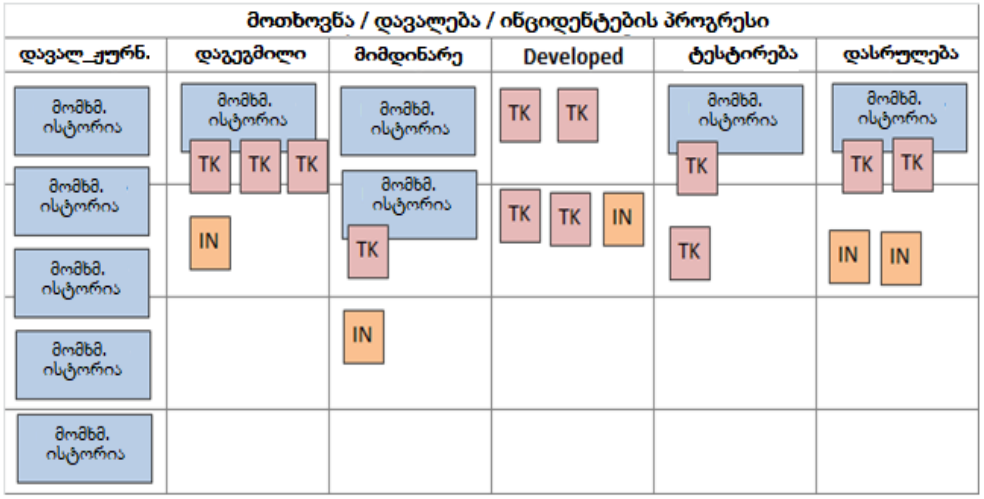
➢ **Kanban-ის პრინციპები:**

- *პ31. დაიწყეთ იმით, რასაც ამჯერად აკეთებთ*: რომელი აქტუალური სამუშაოც სრულდება, ჯერ ის უნდა დასრულდეს და მხოლოდ ამის შემდეგ დაიწყოს ახალი სამუშაო;
- *პ32. დათანხმდით, რომ ევოლუციური ცვლილებები მოსალოდნელია*: შემდგომ განვითარებას აქვს განსაკუთრებული მნიშვნელობა, ოღონდაც აქ სრულყოფა მიღწეულ უნდა იქნას ძირითადად მცირე / ევოლუციური ბიჯების ხარჯზე;
- *პ33. პატივი ეცით პირველარსებულ პროცესებს / როლებს / ვალდებულებებს*: Kanban ადვილად რეალიზებადია, ყველა როლი, პროცესი და ა.შ., რჩება;
- *პ34. წაახალისეთ ლიდერობა ორგანიზაციის ყველა დონეზე*: სრულყოფა შესაძლებელია მხოლოდ იმ შემთხვევაში, თუ ამოქმედებულია ორგანიზაციის ყველა დონე. განსაკუთრებით მნიშვნელოვანია ის, რომ სამუშაოს შემსრულებლები უშუალოდ უკეთებდნენ დემონსტრირებას „ლიდერობის აქტებს“ და ახმოვანებდნენ გაუმჯობესების კონკრეტულ წინადადებებს.

➢ **Kanban-ის ძირითადი პრაქტიკები:**

- *ძ31. სამუშაოს მსვლელობის (ნაკადის) ვიზუალიზაცია:* ღირებულებათა ჯაჭვი პროცესის სხვადასხვა ეტაპებისათვის (მაგალითად, მოთხოვნილების განსაზღვრა, პროგრამირება, დოკუმენტირება, ტესტირება, დანერგვა) კარგადაა ვიზუალიზირებული ყველა მონაწილისათვის. ეს ხორციელდება Kanban-დაფის (Kanban-Board) დახმარებით, რომელზეც სხვადასხვა კვანძები (Stations) აისახება სვეტების სახით (ნახ.22).

ინდივიდუალური მოთხოვნილებები (ამოცანები, ფუნქციები, მომხმარებელთა ისტორიები, მინიმალური საბაზრო მახასიათებლები და ა.შ.) ჩაიწერება სააღრიცხვო ბარათებში („ბილეთები“, მიმაგრებული დაფის უჯრაზე, Tiket - TK).



ნახ.22. Kanban-ის დაფა (იყენებს Software Development Life Cycle-ს)

ისინი დროის და შესრულებული ბიჯის შესაბამისად გადაადგილდება დაფაზე მარცხნიდან მარჯვნივ;

- *ძ32. დაწყებული სამუშაოს მოცულობის შეზღუდვა*: ბილეთების რაოდენობა (Work in Progress - WiP), რომლებიც შეიძლება დამუშავებულ იქნას ერთდროულად ერთ კვანძში, შეზღუდულია.

მაგალითად, თუ კვანძი „პროგრამირება“ ამუშავებს ორ ბილეთს და ამ კვანძის ლიმიტი 2-ია, მაშინ მას არ შეუძლია მე-3 ბილეთის მიღება, თუნდაც მოთხოვნის განსაზღვრება ამის უფლებას აძლევდეს. ეს ქმნის ამოთრევის-სისტემას (Pull-System), სადაც ყოველ კვანძს გადააქვს თავისი სამუშაო წინა კვანძში, და არ გადასცემს დასრულებულ სამუშაოს მომდევნო კვანძს;

- *ძპ3. ნაკადის მართვა (Manage flow)*: Kanban-პროცესის მონაწილეები ზომავენ ტიპურ მაჩვენებლებს, როგორიცაა რიგის სიგრძე, ციკლის დრო, გამტარუნარიანობა, რათა დაადგინონ თუ რამდენად კარგადაა ორგანიზებული სამუშაო პროცესი, სად შეიძლება მისი სრულყოფა და რა შეიძლება დაპირდეს პარტნიორებს, ვისთვისაც მუშაობენ. ეს აადვილებს დაგეგმვას და ამაღლებს საიმედოობას;
- *ძპ4. პროცესისთვის წესები უნდა გაკეთდეს ცხადად:* იმისათვის, რომ პროცესში მონაწილეებმა იცოდნენ თუ რა მოსაზრებებით და კანონებით მუშაობენ, აუცილებელია რაც შეიძლება მეტი ცხადი წესების გაკეთება. მათ შორის:
  - განმარტებულ იქნას ტერმინი „დასრულებულია“, ასევეა განსაზღვრება „მზადაა Scrum-ში“;
  - Kanban-დაფის თითოეული სვეტის მნიშვნელობა;
  - პასუხები შეკითხვებზე: ვინ მოძრაობს, როდის მოძრაობს, როგორ შეირჩეს შემდეგი ბილეთი არსებულებიდან და ა.შ.;
- *ძპ5. უკუკავშირის ციკლების რეალიზაცია*: ფიქსირებულ თარიღებში გუნდები ერთმანეთს უკავშირდება. მაგალითად:
  - რეტროსპექტივები: თანამშრომლობის მიმოხილვა;
  - მომდევნო შეხვედრა: მომავალი დავალებების შეთანხმება / ბლოკირებების მოხსნა / ნაკადის კოორდინაცია;
  - სამუშაო ოპერაციების რეცენზირება: კომპანიის Kanban გუნდები ხვდება და გამოცდილებას უზიარებს ერთმანეთს;
- *ძპ6. მოდელების გამოყენება პროცესის ერთობლივად გაუმჯობესების შესაძლებლობების დასადგენად:* მოდელები ამარტივებს პროცესს. პოპულარული მოდელია, მაგალითად, ღირებულება, ნაკადი, ნარჩენები „ეკონომიური IT“-დან (Lean IT).

7.2.3. Scrum და Kanban მეთოდების შედარება

Scrum-ში ზომავენ სპრინტის დროს შესრულებული დავალებების საერთო წონას. პროექტის ყველა ამოცანის მთლიანი წონის გაყოფით სპრინტის მწარმოებლურობაზე, ღებულობენ პროექტის შესრულების სავარაუდო ვადას. აქედან გამომდინარე, Scrum-ის გუნდის ამოცანაა მწარმოებლურობის გაზრდა და ამით პროექტის შესრულების ვადის შემცირება [4,27].

Kanban – „ბალანსური მიდგომაა“. მისი ამოცანაა გუნდში სხვადასხვა სპეციალისტების (დიზაინერები, დეველოპერები და სხვ.) სამუშაო დატვირთვის დაბალანსება. გუნდი ერთიანია და მასში არაა როლები. ბიზნეს-პროცესი იყოფა არა „სპრინტებად“, არამედ კონკრეტული ამოცანების შესრულების სტადიებად, მაგალითად, „დაგეგმილია“, „მუშავდება“, „ტესტირდება“, „დასრულებულია“ და ა.შ.

ეფექტიანობის მთავარი მაჩვენებელია *ამოცანის გავლის საშუალო დრო* კანბანის დაფაზე დასაწყისიდან დასასრულამდე. თუ ამოცანა სწრაფად გავიდა „ფინალში“, ე.ი. გუნდი მუშაობდა ნაყოფიერად და ჰარმონიულად. თუ გაჭიანურდა ამოცანის შესრულება, მაშინ საჭიროა გარკვევა, თუ რომელ ეტაპზე და რა მიზეზით მოხდა შეფერხება, ვისი საქმიანობის პროცესი მოითხოვს ოპტიმიზაციას [4,29].

7.3. პროგრამული კოდების ტესტირება

განვიხილავთ პროგრამული კოდის ტესტირებას Visual Studio.NET ინტეგრირებულ გარემოში. Unit testing და Coded UI არის Microsoft-ის ტესტირების ინსტრუმენტები, რომლებიც სრულდება Visual Studio.NET-გარემოში [5].

Unit testing – ანუ *მოდულური ტესტირება* დაპროგრამების პროცესია, რომლის საშუალებითაც მოწმდება საწყისი კოდის ცალკეული მოდულების კორექტულობა. ასეთი ტესტირების იდეა მდგომარეობს იმაში, რომ ყოველი არატრივიალური ფუნქციის ან მეთოდისთვის დაიწეროს ტესტი. ეს უზრუნველყოფს კოდის სწრაფად შემოწმებას, ხომ არ მიიყვანა კოდის ბოლო ცვლილებამ პროგრამა შეცდომების გაჩენამდე პროგრამის უკვე ტესტირებულ ნაწილში.

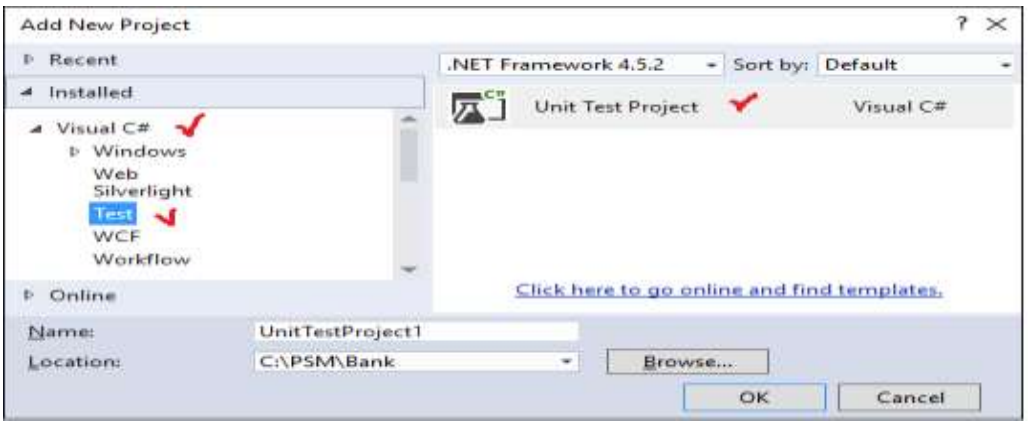
Coded UI ტესტი კი ავტომატურად იწერს, შესრულებაზე უშვებს და ამოწმებს ტესტ-ქეისებს. ასეთი ტესტების წერა შესაძლებელია C# ან Visual Basic-ზე Visual Studio გარემოში.

ამჯერად Unit testing ტექნოლოგია განვიხილოთ ვირტუალური ობიექტის, მაგალითად, *ფინანსური ბანკის* მაგალითზე. საჭიროა დასატესტი პროგრამის შექმნა (ლისტინგი_1), შემდეგ თვით ტესტ-პროგრამის (ლისტინგი_2) შექმნა (Visual Studio .NET გარემოში).

```
//-- ლისტინგი_13.1 --- BankAccount.cs --------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BankAccountNS
{
 public class BankAccount
  {
    private string m_customerName;
    private double m_balance;
    private bool m_frozen = false;
   private BankAccount()
    {
    }

   public BankAccount(string customerName, double balance)
    {
      m_customerName = customerName;
      m_balance = balance;
    }
   public string CustomerName
    {
      get { return m_customerName; }
    }
   public double Balance
    {
      get { return m_balance; }
    }
   public void Debit(double amount)
    {
      if (m_frozen)
       {
         throw new Exception("Account frozen");
       }
      if (amount > m_balance)
       {
          throw new ArgumentOutOfRangeException("amount");
        }
       if (amount < 0)
```

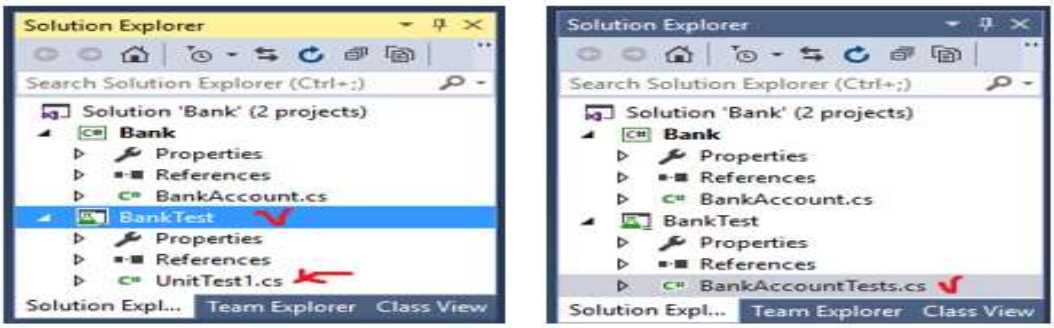
```
        {
          throw new ArgumentOutOfRangeException("amount");
        }
        m_balance += amount; // განზრახ არასწორი კოდი
        // m_balance -= amount;   // გასწორებული
      }
    public void Credit(double amount)
      {
        if (m_frozen)
          {
           throw new Exception("Account frozen");
          }
        if (amount < 0)
         {
          throw new ArgumentOutOfRangeException("amount");
         }
        m_balance += amount;
      }
    private void FreezeAccount()
      {
         m_frozen = true;
      }
    private void UnfreezeAccount()
      {
         m_frozen = false;
      }
    public static void Main()
      {
       BankAccount ba = new BankAccount("Mr.Bryan Walton",
                                        11.99);
       ba.Credit(5.77); ba.Debit(11.22);
       Console.WriteLine("Current balance is ${0}",
                         ba.Balance);
      }
  }
}
```

ტესტ-ფაილის პროექტის შექმნა (Add new Project):



ნახ.23. Unit Test პროექტის შექმნა

მივიღებთ 24-ე ნახაზზე ნაჩვენებ Solution Explorer-ს BankTest პროექტით. მარჯვენა სურათზე შეცვლილია UnitTest კლასის სახელი BankAccountTests-ით.

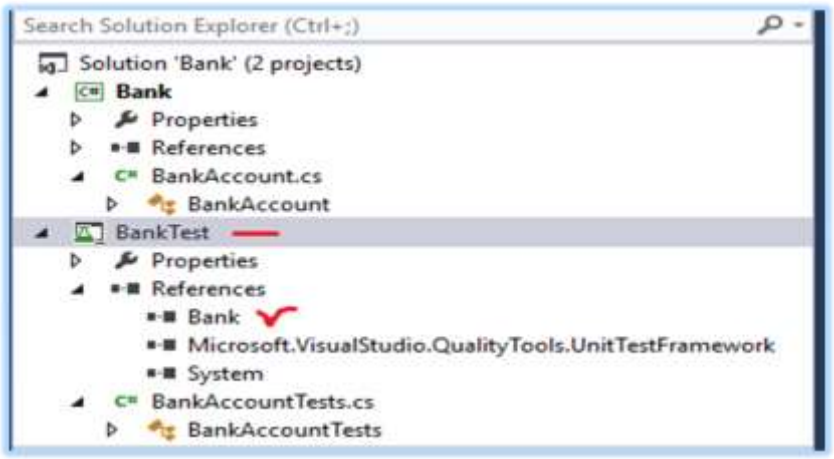


ნახ.24

BankTests პროექტში დავამატოთ reference **Bank** solution-იდან. ამისათვის **BankTests**-ზე მაუსის მარჯვენა ღილაკით ავირჩიოთ **Add Reference** და მივიღებთ 25-ე ნახაზზე ნაჩვენებ ფანჯარას. აქ Solution სტრიქონში ვირჩევთ Projects და Bank-ის ჩეკბოქსს მოვნიშნავთ.

ნახ.25

შედეგი Solution Explorer-ში ნაჩვენებია 26-ე ნახაზზე.



ნახ.26

ჩვამატოთ BankAccountTests პროგრამაში სახელსივრცე Bank-ის პროექტიდან:

```
using BankAccountNS;
```

ამგვარად, BankAccountTests.cs ფაილს ექნება 2_ლისტინგზე ნაჩვენები სახე.

```
//-- ლისტინგი_2 -----  BankAccountTests.cs -----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
    [TestClass]
    public class BankAccountTests
```

```
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

ახლა შევქმნათ პირველი ტესტ-მეთოდი. ამ პროცედურაში, დაიწერება უნიტ-ტესტის მეთოდები BankAccount class-ის Debit მეთოდის ქცევის ვერიფიკაციისათვის. ეს მეთოდები ზემოთაა ჩამოთვლილი.

დასატესტი მეთოდების ანალიზის გზით გაირკვა, რომ საჭიროა მინიმიუმ სამი ქცევის შემოწმება:

1) მეთოდი ქმნის ArgumentOutOfRangeException – გამონაკლისს, თუ კრედიტის ჯამი გადააჭარბებს ბალანსს;

2) იგი ქმნის ArgumentOutOfRangeException – გამონაკლისს მაშინაც, როცა კრედიტის ზომა უარყოფითია;

3) თუ 1 და 2 პუნქტები წარმატებით დასრულდა, მაშინ მეთოდი ითვლის ჯამს ბალანსის ანგარიშიდან.

პირველ ტესტში შევამოწმოთ, რომ კრედიტის დასაშვები მნიშვნელობისთვის (როცა დადებითი მნიშვნელობისაა და ბალანსის ანგარიშზე ნაკლებია) ანგარიშიდან მოიხსნება საჭირო თანხა.

- დავამატოთ BankAccountTests კლასს შემდეგი მეთოდი:

```
// unit test code -----
[TestMethod]
public void Debit_WithValidAmount_UpdatesBalance()
{
  // arrange --------
   double beginningBalance = 11.99;
   double debitAmount = 4.55;
   double expected = 7.44;
   BankAccount account = new BankAccount("Mr. Dito", beginningBalance);
   // act ---------
   account.Debit(debitAmount);
   // assert -------
   double actual = account.Balance;
   Assert.AreEqual(expected, actual, 0.001, "Account not debited correctly");
}
```

მეთოდი საკმაოდ მარტივია. ჩვენ ვქმნით ახალ BankAccount ობიექტს საწყისი ბალანსით და შემდეგ ვაკლებთ სწორ ოდენობას. ჩვენ ვიყენებთ Microsoft-ის unit-ტესტის ფრეიმვორკს მართვადი კოდის AreEqual მეთოდისთვის, რათა მოხდეს საბოლოო ბალანსის ვერიფიკაცია – „არის ის, რასაც ჩვენ ველით“ ?

ტესტ-მეთოდის მოთხოვნები ასეთია:

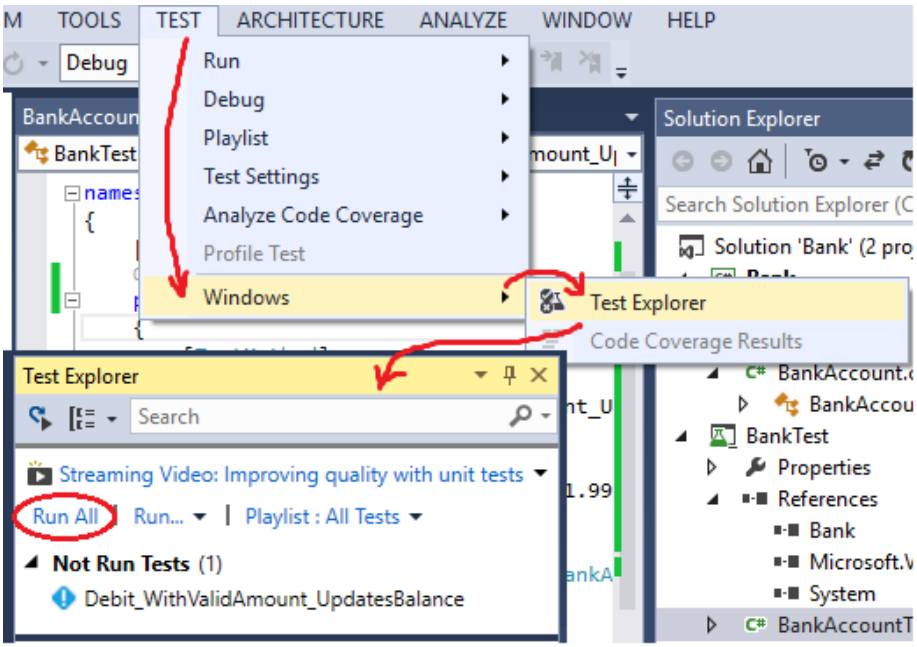
1) მეთოდი მონიშნული უნდა იყოს [TestMethod] ატრიბუტით;
2) მეთოდმა უნდა დააბრუნოს void;
3) მეთოდს არ შეიძლება ჰქონდეს პარამეტრები.

- ტესტის აგება და ამუშავება (მთლიანი ტესტის კოდი იხ. ლისტინგი_3):

```
//-- ლისტინგი_3 ----- BankAccountyTests.cs ----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
  [TestClass]
  public class BankAccountTests
  {
     [TestMethod]
     public void Debit_WithValidAmount_UpdatesBalance()
     {
       // arrange
       double beginningBalance = 11.99;
       double debitAmount = 4.55;
       double expected = 7.44;
       BankAccount account = new BankAccount("Mr. Dito",
                    beginningBalance);
       // act
       account.Debit(debitAmount);

       // assert
       double actual = account.Balance;
       Assert.AreEqual(expected, actual, 0.001, "Account
              not debited correctly");
     }
  }
}
```

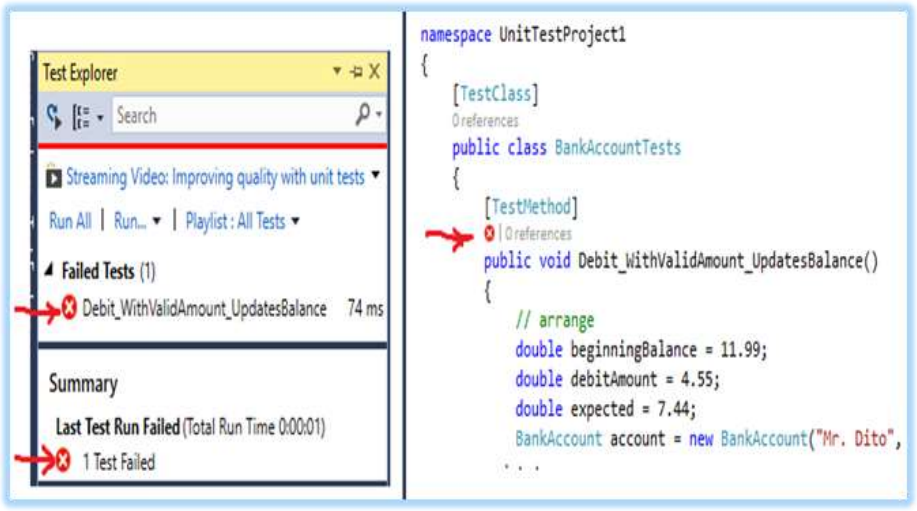
- BUILD მენიუდან ვირჩევთ Build Solution;

- TEST მენიუდან ვირჩევთ Windows და Test Explorer პუნქტებს. იხსნება Test Explorer ფანჯარა (ნახ.27).



ნახ.27

აქ ვირჩევთ Run All - ს და ვღებულობთ შედეგს (ნახ.28).



ნახ.28. არასწორი შედეგი

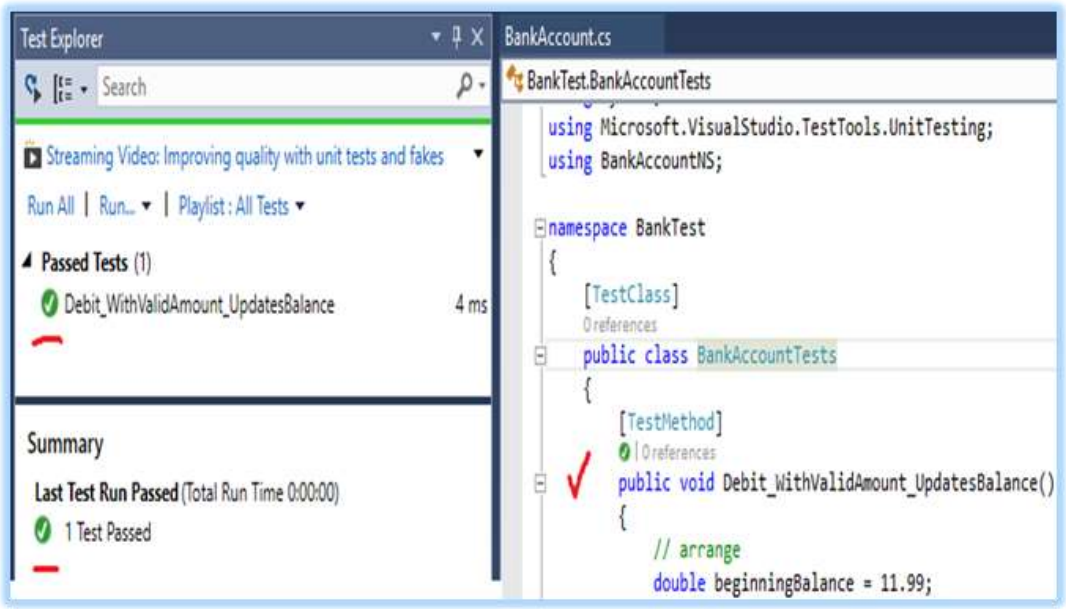
ნახაზზე მითითებული „x“-სიმბოლოები წითელ წრეშია მოთავსებული, ე.ი. ტესტირებამ აღმოაჩინა შეცდომები და კოდი ვერ შესრულდა წარმატებით.

თუ მეთოდი წარმატებით ჩაივლიდა, მაშინ მივიღებდით მწვანე ფერის x-სიმბოლოებს.

შემდეგი ეტაპი კოდის გასწორება და ხელახალი ტესტირებაა. დასატესტ პროგრამაში შევცვალოთ სტრიქონში „ + „ ნიშანი „ – „ -ით.

```
// m_balance += amount; // intentionally incorrect code
m_balance -= amount; // intentionally correct code
```

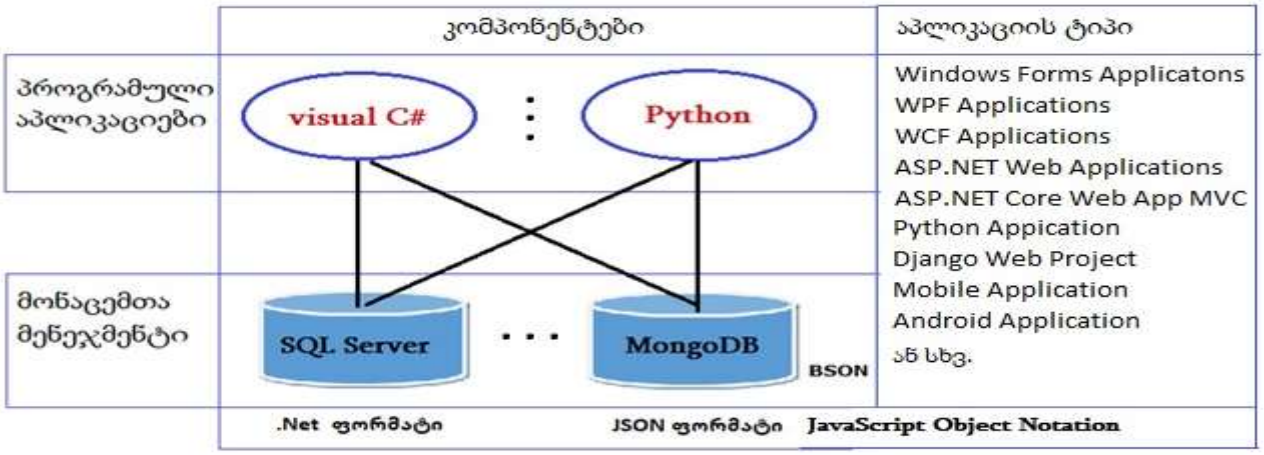
ტესტის თავიდან ამუშავებით ვღებულობთ წარმატებულ შედეგს, ანუ მიიღება მწვანე ფერის სიმბოლოები (ნახ.29).



ნახ.29. სწორი შედეგი

7.4. მონაცემთა ბაზის აგება პროგრამული აპლიკაციისთვის

საბაკალავრო პროექტის პროგრამული სისტემის (Desktop-, Web- ან Mobile-აპლიკაციის) მონაცემთა ბაზის დაპროექტება და რეალიზაცია ხორციელდება Ms SQL Server, MySQL, Oracle ან სხვა პაკეტების საფუძველზე [2,14], რომელსაც ფლობს სტუდენტი და რომელიც მისაღებია საკვლევი ობიექტის აპლიკაციისათვის (ნახ.30).

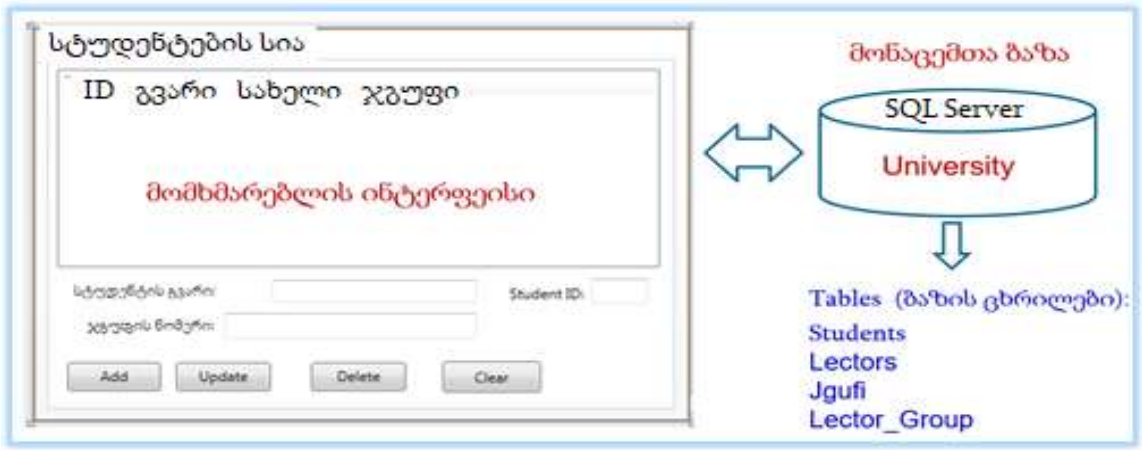


ნახ.30. აპლიკაციის ტიპები და კომპონენტები

მონაცემთა ბაზა შეიძლება იყოს ლოკალური ან განაწილებული. რელაციური ბაზების თეორია საერთოა ზემონახსენები პაკეტებისთვის. ამიტომ საპრობლემო სფეროს კონცეპტუალური მოდელი (ER-Model) ერთნაირია, განსხვავება იქნება რეალიზაციის დონეზე. შესაძლებელია NoSQL ტიპის ბაზის გამოყენებაც (მაგალითად, MongoDB), თუ პროექტი ეხება დიდ მონაცემთა (Big Data) დამუშავების სისტემებს [9,10,13].

პროგრამული სისტემის სასიცოცხლო ციკლის ეტაპების მიხედვით, ეს მასალა მოიცავს ოო-ანალიზს, ოო-დაპროექტებას და პროგრამულ რეალიზაციას (დეველოპმენტ-ტესტირებას).

სტუდენტმა ამ პარაგრაფში უნდა წარმოადგინოს აგრეთვე თავისი ობიექტისთვის აგებული მონაცემთა ბაზის მიერთების საკითხები პროგრამულ აპლიკაციასთან. შესაძლებელი უნდა იყოს პროგრამული სისტემის დამკვეთი ორგანიზაციის მომხმარებელთა ინტერფეისებში (ეკრანებზე) ინფორმაციული ბაზის მონაცემთა გამოტანა, დამატება, მოდიფიკაცია და წაშლა (მონაცემთა ავტომატიზებული მენეჯმენტის განხორციელება). 31-ე ნახაზზე ნაჩვენებია ასეთი დიალოგური სისტემის სტრუქტურული სქემის ფრაგმენტი „უნივერსიტეტი“-ს მაგალითზე.



ნახ.31. პროგრამული აპლიკაციის ზოგადი სქემა

საჭიროა ღილაკების ფუნქციების დაწერა და მონაცემთა ბაზასთან (SQL Server -თან) მიერთება (connection) მონაცემების დასამატებლად (Add), შესასწორებლად (Update), წასაშლელად (Delete) და შესატანი ტექსტბოქსების გასასუფთავებლად (Clear). საილუსტრაციო C# კოდის ფრაგმენტები (მაგალითად, WPF აპლიკაციისათვის: C#, XAML) მოცემულია 4-7 ლისტინგებში [2].

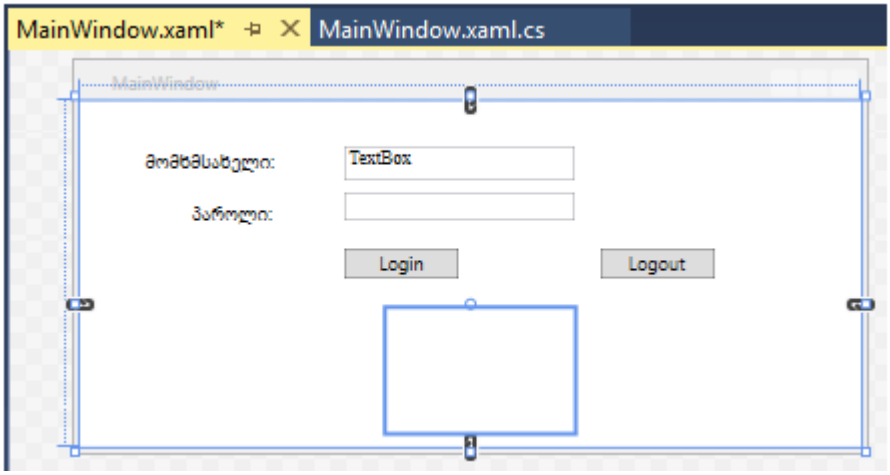
```
// ---- ლისტინგი_4 ---- Add() მეთოდი ----------
    private void Window_Loaded(object sender, RoutedEventArgs e)
     {
         ShowData();
     }
      // ---   ShowData() მეთოდის ტექსტი ------
     public void ShowData()
     {
       SqlConnection con = new SqlConnection(@"Data Source=Z-PC\MSSQLSERVER1;Initial
                        Catalog=University;Integrated Security=True");
       con.Open();
       SqlCommand comm = new SqlCommand("Select St_ID, Name, Gr_Nom from Student", con);
       DataTable dt = new DataTable();
       SqlDataAdapter da = new SqlDataAdapter(comm);
       da.Fill(dt);
       listView1.DataContext = dt.DefaultView;
      }

// ---- ლისტინგი_5 ---- Add() მეთოდი ----------
private void btnAdd_Click(object sender, RoutedEventArgs e)
    {
       string Name = textBox1.Text;
       string Gr_Nom = textBox2.Text;
       string St_ID = textBox3.Text;
       SqlConnection con = new SqlConnection(@"Data Source=Z-PC\MSSQLSERVER1;Initial
                                              Catalog=University;Integrated Security=True");
       con.Open();
       SqlCommand comm = new SqlCommand("insert into Student(Name,Gr_Nom, St_ID)
                                               values(@Name, @Gr_Nom, @St_ID)", con);
       comm.Parameters.AddWithValue("@Name", textBox1.Text);
       comm.Parameters.AddWithValue("@Gr_Nom", textBox2.Text);
       comm.Parameters.AddWithValue("@St_ID", textBox3.Text);
       comm.ExecuteNonQuery();
       con.Close();
       ShowData();
    }

// ---- ლისტინგი_6 ---- Update() მეთოდი ----------
private void btnUpdate_Click(object sender, RoutedEventArgs e)
   {
     if (listView1.SelectedItems.Count > 0)
```

```
        {
            DataRowView drv = (DataRowView)listView1.SelectedItem;
            string id = drv.Row[0].ToString();
            SqlConnection con = new SqlConnection(@"Data Source=Z-PC\MSSQLSERVER1;Initial
                        Catalog=University;Integrated Security=True");
            con.Open();
            SqlCommand comm = new SqlCommand("update Student set Name=@Name,Gr_Nom=@Gr_Nom
                        where St_ID=@St_ID", con);
            comm.Parameters.AddWithValue("@St_ID", id);
            comm.Parameters.AddWithValue("@Name", textBox1.Text);
            comm.Parameters.AddWithValue("@Gr_Nom", textBox2.Text);
            comm.ExecuteNonQuery();
            con.Close();
            ShowData();
        }
    }

// ---- ლისტინგი_7 ---- Delete() მეთოდი ----------
private void btnDelete_Click(object sender, RoutedEventArgs e)
    {
        if (listView1.SelectedItems.Count > 0)
        {
            DataRowView drv = (DataRowView)listView1.SelectedItem;
            string id = drv.Row[0].ToString();
            SqlConnection con = new SqlConnection(@"Data Source=Z-PC\MSSQLSERVER1;Initial
                    Catalog=University;Integrated Security=True");
            con.Open();
            SqlCommand comm = new SqlCommand("delete from Student where St_ID=@St_ID", con);
            comm.Parameters.AddWithValue("@St_ID", id);
            comm.ExecuteNonQuery();
            ShowData();
        }
    }

// ---- ლისტინგი_8 ---- Clear() მეთოდი ----------
    private void btnClear_Click(object sender, RoutedEventArgs e)
    {
        textBox1.Text = "";
        textBox2.Text = "";
        textBox3.Text = "";
    }
```

7.5. პროგრამული აპლიკაციის უსაფრთხოება

საბაკალავრო პროექტი სასურველია შეიცავდეს პროგრამული სისტემის უსაფრთხოების საკითხის გადაწყვეტას. კონკრეტული ობიექტიდან და პროექტის მიზნებიდან გამომდინარე შესაძლებელია განხილულ იქნას სისტემის, პროგრამების და მონაცემების დაცვის ალტერნატიული ვარიანტები. ამას გადაწყვეტს თვით პროექტის ავტორი ხელმძღვანელის რეკომენდაციით [7].

მაგალითად, აქ წარმოდგენილი გვაქვს პროგრამული სისტემის დაცვა არასანქცირებული მიმართვისგან. ანუ მრავალმომხმარებლურ პროგრამულ სისტემაში შესვლა ხორციელდება, ჩვეულებისამებრ, UserName და Password - მნიშვნელობების შეტანით (ნახ.32).



ნახ.32. ავტორიზაციის დიზაინის ფორმა

9-10 ლისტინგებში მოცემულია შესაბამისი სისტემის XAML და C# პროგრამების ტექსტის ფრაგმენტები.

```
<!-- ლისტინგი_9 ---- XAML პროგრამა დიზაინისთვის -->
<Window x:Class="test2Log.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Label Content="მომხმ_სახელი:" HorizontalAlignment="Left" Margin="38,42,0,0"
               VerticalAlignment="Top" Width="98"/>
        <Label Content="პაროლი:" HorizontalAlignment="Left" Margin="69,77,0,0"
               VerticalAlignment="Top" Width="67"/>
        <TextBox x:Name="UserName" HorizontalAlignment="Left" Height="23"
               Margin="175,46,0,0" TextWrapping="Wrap" Text="TextBox"
               VerticalAlignment="Top" Width="152" FontFamily="Times New Roman"/>
        <PasswordBox x:Name="PasswordBox" HorizontalAlignment="Left" Margin="175,77,0,0"
               VerticalAlignment="Top" Width="152"/>
        <Button x:Name="LoginBTN" Content="Login" HorizontalAlignment="Left"
                Margin="175,123,0,0" VerticalAlignment="Top" Width="75"/>
        <Button x:Name="LogoutBTN" Content="Logout" HorizontalAlignment="Left"
                Margin="345,123,0,0" VerticalAlignment="Top" Width="75"
                Visibility="Collapsed"/>
        <Image x:Name="mainImage" HorizontalAlignment="Left" Height="146"
               Margin="203,163,0,0" VerticalAlignment="Top" Width="124"
               Visibility="Collapsed"/>
    </Grid>
</Window>
```

```
// ---  C# კოდი ---- ავტორიზაციის ლოგიკის რეალიზაცია ---
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace test2Log
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void LoginBTN_Click(object sender, RoutedEventArgs e)
        {
          if (!UserName.Text.Equals("") && !PasswordBox.Password.Equals(""))
           {
             if (UserName.Text.Equals("gsurg") && PasswordBox.Password.Equals("123"))
               {
                  mainImage.Visibility = Visibility.Visible;

                  LoginBTN.Visibility = Visibility.Collapsed;
                  LogoutBTN.Visibility = Visibility.Visible;
               }
              else
                   MessageBox.Show("მომხმ_სახელი ან პაროლი არასწორია !");
            }
            else
                MessageBox.Show("ინფორმაცია არაა !");
        }

        private void LogoutBTN_Click(object sender, RoutedEventArgs e)
        {
            mainImage.Visibility = Visibility.Collapsed;
            LoginBTN.Visibility = Visibility.Visible;
            LogoutBTN.Visibility = Visibility.Collapsed;
        }
    }
}
```

მონაცემთა მენეჯმენტის თვალსაზრისით, საბაკალავრო პროექტში სასურველია მონაცემთა ბაზის დაცვის პრიორიტეტული ამოცანისათვის დაიწეროს შესაბამისი ტრიგერ (trigger) პროცედურები, განისაზღვროს მისი მუსაობის ლოგიკა.

პროგრამული სისტემის უსაფრთხოების საკითხები მრავალფეროვანია, ამიტომ პროექტის ავტორი თვითონ განსაზღვრავს შესაბამისი ამოცანების დასმას და მათ გადაწყვეტას. მეცნიერული კვლევის თვალსაზრისით სასურველია მაგალითად, პროგრამული სისტემის საიმედოობის ამოცანების გადაწყვეტა.

7.6. მომხმარებლის ინტერფეისი (UI/UX) პროგრამულ აპლიკაციაში

*მომხმარებლის ინტერფეისი / მომხმარებლის გამოცდილება (**User Interface - UI) / User Experience - UX)*** –მნიშვნელოვანი ცნებებია ინფორმაციული სისტემების დაპროექტებისა და აგების სფეროში [3,24].

UI – მომხმარებლის ინტერფეისი არის კომპიუტერული სისტემის დაპროექტებისა და პროგრამირების პროცესი, რომლის დროსაც დეველოპერების მიერ მზადდება დესკტოპ-, ვებ-ან მობილური აპლიკაციის სამუშაო ინტერფეისი დამკვეთი-მომხმარებლისთვის.

UX – მომხმარებლის გამოცდილება არის მომხმარებლის ქცევის მანიპულირების პროცესი ანუ როგორ იმუშავებს იგი მომხმარებლის ინტერფეისთან.

33-ე ნახაზზე ნაჩვენებია, სიმბოლურად, UI / UX-ის ურთიერთმიმართების სურათი, „აისბერგის“ პრინციპის სახით. მომხმარებელი ხედავს მის წყლისზედა ნაწილს ანუ UI-ინტერფეისის დიზაინს, წყალქვეშა ნაწილი UX კი მისთვის უხილავია.

ნახ.33. UI / UX შედარება

UI და UX ტერმინები ხშირად გამოიყენება ვებ- და მობილური აპლიკაციების დიზაინის პროცესში. ჩვეულებრივ, UI/UX ტერმინების დონეზე არაა მარტივი მათ შორის ძირეული განსხვავებების დაფიქსირება. შევეხოთ მოკლედ ამ საკთხს უფრო დეტალურად.

UX მჭიდროდაა დაკავშირებული UI დიზაინთან, მარტივად რომ ვთქვათ, UX დიზაინერები წყვეტენ როგორ იმუშაოს „User Inteface”-მა, ხოლო UI დიზაინერები კი - როგორ გამოიყურებოდეს „User Interface”.

UI დიზაინსა და UX დიზაინს აქვს განსხვავებული დავალებები, მაგრამ ისინი მუშაობს ერთმანეთის წარმატებისთვის. კარგ UI დიზაინად ვერ ჩაითვლება ისეთი, რომელიც დამაბნეველია კლიენტისათვის. ასევე შესაძლებელია საუკეთესოდ იყოს შექმნილი UX მხარე, თუმცა UI დიზაინმა, მისი არაესთეტიურობის გამო, დააბრკოლოს გამოყენება. კარგ UI/UX-ად ითვლება ის ვერსია, რომელშიც ორივე მხარე ჰარმონიულადაა მოგვარებულია.

პროგრამული აპლიკაციის ხარისხის თვალსაზრისით UI/UX ტანდემის ორიგინალური გადაწყვეტა ძალზე მნიშვნელოვანია, მითუმეტეს ისეთი კორპორაციული პროგრამული აპლიკაციებისთვის, რომლებიც დიდ ინფორმაციას ინახავს და ამუშავებს. თუ პროგრამული კოდი და მონაცემთა ბაზები კარგად არ იქნება დაპროექტებული, შესამჩნევად გაიზრდება პროგრამის მუშაობის ხანგრძლიობა და კლიენტისათვის პასუხის მიწოდების დრო.

7.7. პროგრამული აპლიკაციის ხარისხის შეფასება

პროგრამული უზრუნველყოფის ხარისხი მოცემული დავალების ფარგლებში დადგენილი მოთხოვნილებების სრულყოფილად შესაბამისობაა რეალიზებულ პროგრამულ პროდუქტთან. პროგრამული უზრუნველყოფის ხარისხის საერთაშორისო სტანდარტებია - ISO/IEC 25000:2014, IEEE Std 610.12-1990 [3,5].

პროგრამული უზრუნველყოფის ხარისხის მახასიათებლები არაფუნქციონალური მოთხოვნების ნაწილია, რომელიც ძირითადად მოიცავს შემდეგ კრიტერიუმებს:

გასაგები – პროგრამული უზრუნველყოფის დანიშნულება გასაგები უნდა იყოს როგორც სისტემის მუშაობიდან, ისე მისი დოკუმენტაციიდან;

სრული – პროგრამული უზრუნველყოფის ყველა აუცილებელი კომპონენტი უნდა იყოს სრულად წარმოდგენილი და რეალიზებული;

ლაკონიური – ზედმეტი და დუბლირებული ინფორმაცია უნდა იყოს შეზღუდული. კოდის განმეორებადი ნაწილებისათვის დაცული უნდა იყოს პოლიმორფიზმის პრინციპი;

პორტირების შესაძლებლობა – პროგრამული უზრუნველყოფა ადვილად უნდა ადაპტირდეს ახალ გარემოში (მაგალითად, არქიტექტურის, პლატფორმის, ვერსიისა და ა.შ. შეცვლისას).

შეთანხმებული – პროგრამული უზრუნველყოფის ნებისმიერ კომპონენტში (პროგრამული კოდი, ტექნიკური დავალება, ტექნიკური პროექტი, დოკუმენტაცია და ა.შ.) გამოყენებულ უნდა იქნას ერთიანი, შეთანხმებული ტერმინოლოგია და აღნიშვნები.

არსებობს კოდის შემოწმებისა და ანალიზის შემდეგი მიდგომები:

1) ***Maintainability Index (მხარდაჭერის ინდექსი)*** – კოდის ხარისხის კომპლექსური მაჩვენებელია. იგი განისაზღვრება ფორმულით:

$$MI = MAX(0,(171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LoC)) * 100 / 171),$$

სადაც,

- HV – Halstead Volume, გამოთვლითი სირთულე. მეტრიკის სიდიდე პირდაპირპრო-პორციულად იზრდება გამოყენებული ოპერატორების სიმრავლის შესაბამისად;
- CC – Cyclomatic Complexity. კოდის სტრუქტურული სირთულე ანუ კოდში სხვადასხვა განშტოებების რაოდენობა. რაც უფრო მაღალია მაჩვენებელი, მითი უფრო მეტი ტესტირებაა გასაწერი;
- LoC – კოდის სტრიქონების რაოდენობა.

ამ მეტრიკის ფარგლებში კოდის სირთულის მაჩვენებელი ვარირებს 0-დან 100-მდე. რაც უფრო მაღალია მნიშვნელობა, მით უფრო მოქნილია კოდის მხარდაჭერა.

Visual Studio პაკეტში თვალსაჩინოებისათვის შემუშავებულია ფერებად რანჟირებული დაყოფა – მწვანე ფერი შეესაბამება 20-100 დიაპაზონის მნიშვნელობას, ყვითელი ფერი შეესაბამება 10-20 დიაპაზონის მნიშვნელობას, ხოლო 10-ზე ნაკლების დიაპაზონის მნიშვნელობის ფერია წითელი.

2) ***Depth of Inheritance – მემკვიდრეობითობის სიღრმე.*** თითოეული კლასისათვის აჩვენებს მემკვიდრეობის ჯაჭვის იერარქიას. მაგალითად, პირველი კლასის მემკვიდრეა მეორე კლასი, ხოლო მესამე კლასი მეორე კლასის მემკვიდრეა. შესაბამისად, პირველი კლასის მაჩვენებელია 1, მეორესი 2, მესამესი 3.

3) ***Class Coupling – კლასების ურთიერთდამოკიდებულებისა და დაკავშირების მაჩვენებელი.***

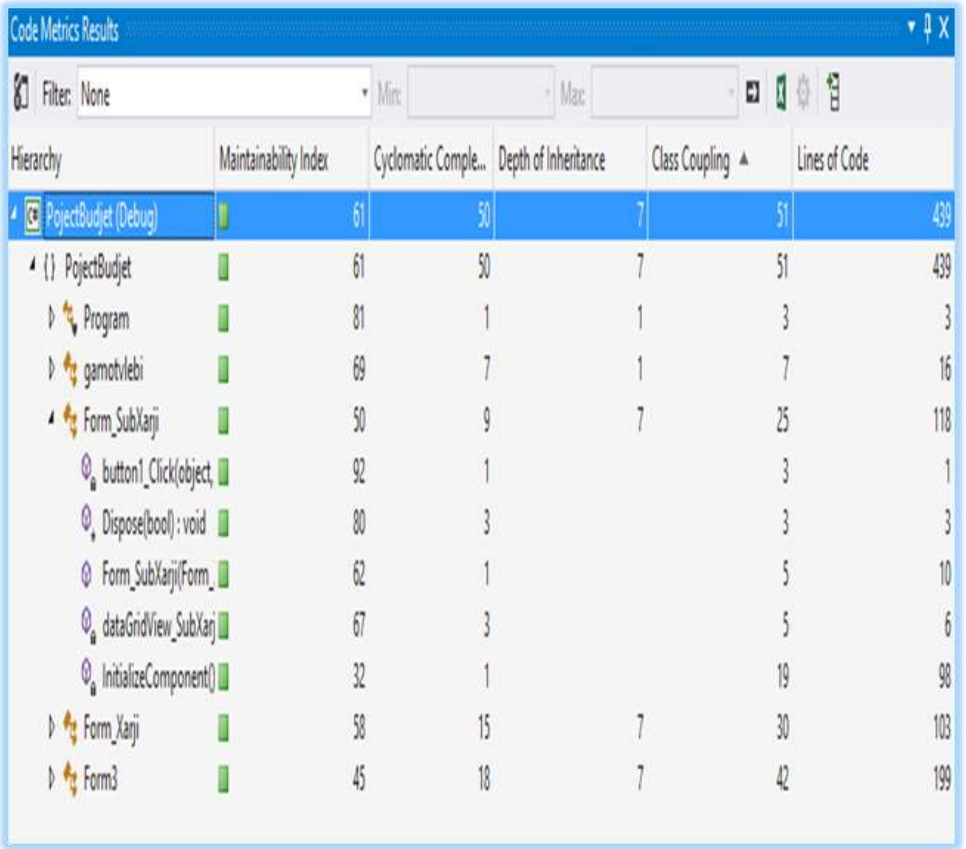
კარგი პრაქტიკაა კლასებს შორის დამოკიდებულება არ იყოს „ჩახლართული“ და არ იყოს გამოყენებული დიდი რაოდენობის ქვეკავშირი (გამოთვლაში მონაწილეობს – პარამეტრული კლასები, ლოკალური ცვლადები, მეთოდით დაბრუნებული ტიპები, საბაზო კლასები, ატრიბუტები და სხვ.)

4) ***Lines of Code – კოდის სტრიქონების რაოდენობა*** (არ გაითვალისწინება კოდში ცარიელი სტრიქონები და კომენტარები).

Visual Studio პაკეტში პროგრამული კოდის მეტრიკის შედეგების მიღება პროექტისთვის ხდება შემდეგი ბრძანებით:

ძირითად მენიუში ღილაკით Analyze – Calculate Code Metrics - for Solution. ასევე, შესაძლებელია ნაწილში Solution Explorer, solution- Calculate Code Metrics გამოძახება მაუსის მარჯვენა ღილაკით.

აღნიშნული ბრძანება აჩვენებს როგორც ზოგად, ისე კლასების დონეზე ჩაშლილ შედეგს (Code Metrics Results) პარამეტრებით - Maintainability Index, Cyclomatic Complexity, Depth of Inheritance, Class Coupling, Lines of Code (ნახ.34).



ნახ.34

კოდში კომპილიატორის შეცდომების ან გაფრთხილების ფანჯარა Error List იხსნება ძირითადი მენიუს ღილაკით View-Error List, ასევე ძირითად მენიუში ღილაკით Analyze – Run code Analysis and suppress active issues.

კომპილატორის გაფრთხილება (Compiler warnings) – პროგრამულ კოდში საეჭვო ადგილების არსებობაა, რომელიც პროგრამული ენის თვალსაზრისით არ არის შეცდომა და არ იწვევს პროგრამული კოდის კომპილირების პროცესის შეწყვეტას, თუმცა არის პროგრამული შეცდომა.

კომპილატორის გაფრთხილება შესაძლებელია მნიშვნელოვანი იყოს საინფორმაციო სისტემების რისკების მართვის პროცესისათვის, რაც იმის მანიშნებელია, რომ კოდს შესაძლოა ჰქონდეს მრავალი სისუსტე, ღია ადგილები ან გადატვირთული გამოუყენებელი ელემენტები. ასეთი ტიპის შევდომებმა შესაძლოა გამოიწვიოს კოდის შესრულების შენელებაც.

ნაწილში (Warnings) ველი suppress (გაბათილებადია) მიანიშნებს, კოდის რომელი ელემენტია გამოცხადებული, თუმცა გამოუყენებელი.

7.8. DevOps მეთოდოლოგია და ინსტრუმენტული საშუალებები

Agile და DevOps მეთოდოლოგიები ასრულებს ურთიერთდამატებით როლებს. მაგალითად, პროგრამული სისტემების ავტომატიზებული კონსტრუირება და ტესტირება, უწყვეტი ინტეგრაცია და უწყვეტი მიწოდება [3,4]. Agile შეიძლება ჩაითვალოს, როგორც დამკვეთებსა და დეველოპერებს შორის საკომუნიკაციო ხარვეზების გადაჭრის მექანიზმი, ხოლო DevOps კი ორიენტირებულია დეველოპერებსა და IT ოპერაციებს / ინფრასტრუქტურებს შორის არსებული ხარვეზების აღმოფხვრაზე.

ნახ.35. DevOps მეთოდოლოგია

DevOps მეთოდოლოგია აერთიანებს პროგრამული უზრუნველყოფის დეველოპმენტის (Dev) და ინფორმაციული ტექნოლოგიების ოპერაციებს (Ops) (ნახ.35).

მისი მიზანია პროგრამული სისტემების შექმნის ციკლის დროის შემცირება და მაღალი ხარისხის პროგრამული უზრუნველყოფის უწყვეტი მიწოდება [17]. DevOps ასევე, ყურადღებას ამახვილებს აგებული პროგრამული უზრუნველყოფის განთავსების (deployment) ამოცანებზე.

DevOps არის გუნდური მუშაობის კონცეფცია, ამიტომაც არ არესებობს ერთი კონკრეტული ინსტრუმენტი მის განსახორციელებლად. პირიქით, არსებობს რამდენიმე ინსტრუმენტის ერთობლიობა („DevOps ინსტრუმენტების ჯაჭვი“), რომლებიც კარგად ასახავს პროგრამული სისტემების დეველოპმენტის და დამკვეთებზე მიწოდების ასპექტებს [17]:

1) Coding – კოდის დეველოპმენტი და ანალიზი, საწყისი კოდის მენეჯმენტის (Source Code Management) ინსტრუმენტი, კოდების შერწყმა [18];

2) Building – უწყვეტი ინტეგრაციის (Continuous Integration) ინსტრუმენტი, კონსტრუირების სტატუსი [19];

3) Testing – უწყვეტი ტესტირების (Continuous Testoing) ინსტრუმენტი, რომელიც უზრუნველყოფს სწრაფ და დროულ უკუკავშირს ბიზნეს რისკების მიხედვით [20];

4) Packaging – არტეფაქტების რეპოზიტორია (მონაცეთა საცავი - data warehouse), აპლიკაციის წინასწარი განთავსება. საცავში ინახება პროგრამული პაკეტები და მათი მეტამონაცემები, ცვლილებათა ჟურნალი სისტემის მენეჯერისათვის [21];

5) Releasing – ცვლილებათა მენეჯმენტი, რელიზის (ვერსიის) გამოცემის დამტკიცება, აპლიკაციის ვერსიის ავტომატიზაცია Application-release automation (ARA) [22];

6) Configuring – ინფრასტრუქტურის კონფიგურაცია და მენეჯმენტი, ინფრასტრუქტურა, როგორც კოდის ინსტრუმენტი (Infrastructure as code – IaC) [23];

7) მონიტორინგი – აპლიკაციების მწარმოებლურობის (სწრაფქმედების) მონიტორინგი, მუშაობის გამოცდილება საბოლოო მომხმარებელთან (Application Performance Management – APM). APM ცდილობს კომპლექსური პროგრამების შესრულების პრობლემების გამოვლენას [3] (იხ. პარაგრაფი 2.3 და 3.2.1 – პროფაილერი), სადაც პროგრამული სისტემის მწარმოებლურობის ოპტიმიზაციის საკითხი დეტალურადაა განხილული – React Hook-ის საფუძველზე.

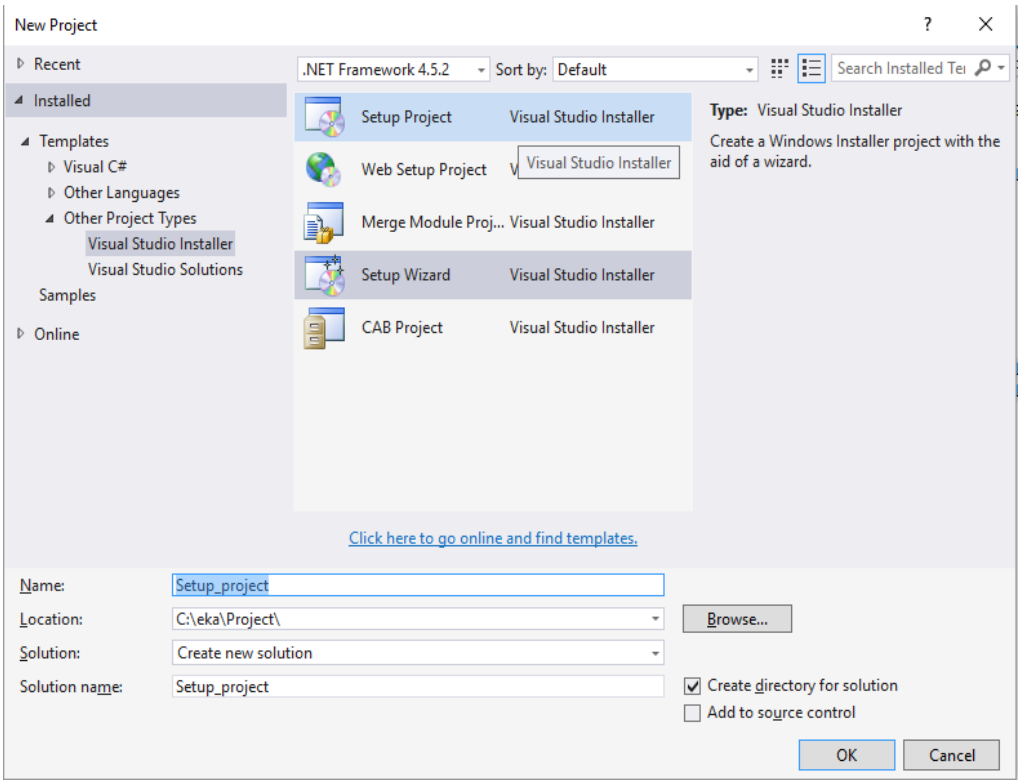
8) და სხვ.

7.9. პროგრამული აპლიკაციის დისტრიბუციული ფაილის (საინსტალაციო პაკეტის) შექმნა

საბაკალავრო პროექტის პროგრამული ნაწილის დეველოპმენტისა და წარმატებული ტეატირების შემდეგ სასურველია გაკეთდეს მისი დისტრიბუციული პაკეტი (საინსტალაციო .exe ფაილი), რომელიც გადაეცემა დამკვეთს ან გამზადებულია გასაყიდი პროდუქტის სახით [5].

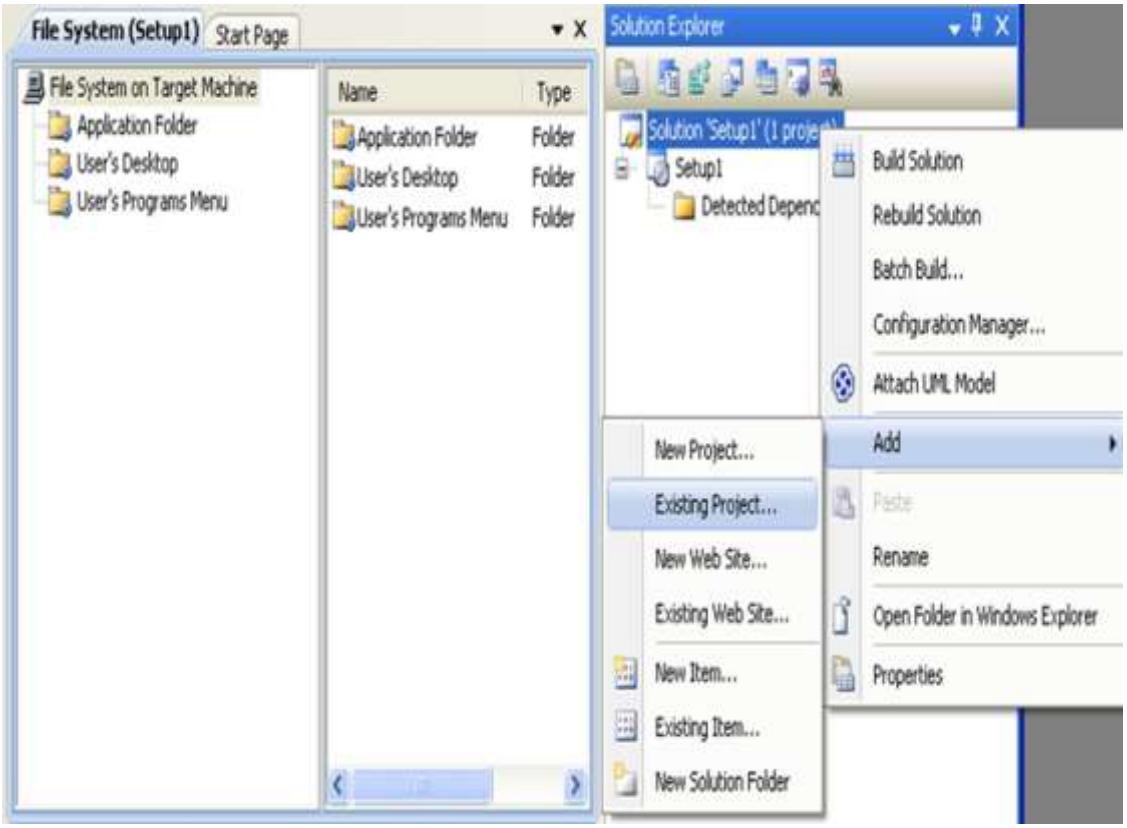
Visual Studio სისტემაში ახალი პროექტის შექმნის მენიუში Other Project Type-Visual Studio Installer პროგრამული პაკეტის გამშვები (exe) ფაილის შექმნისათვის (ნახ.36) არის ორი არჩევანი Setup Project და Setup Wizard. მიუხედავად ამისა, მათი ფუნქციონირება მცირედით განსხვავდება.

Setup Wizard საშუალებას იძლევა ბიჯური რეჟიმით შეიქმნას Web ან Windows ფორმის საინსტალაციო გარემო და ინტეგრირდეს სისტემისათვის თანამდევი ფაილებით. თუმცა გამშვები ფაილის მომზადების ბირთვის ავტომატიზებულ მექანიზმებს Setup Wizard საშუალება არ შეიცავს.

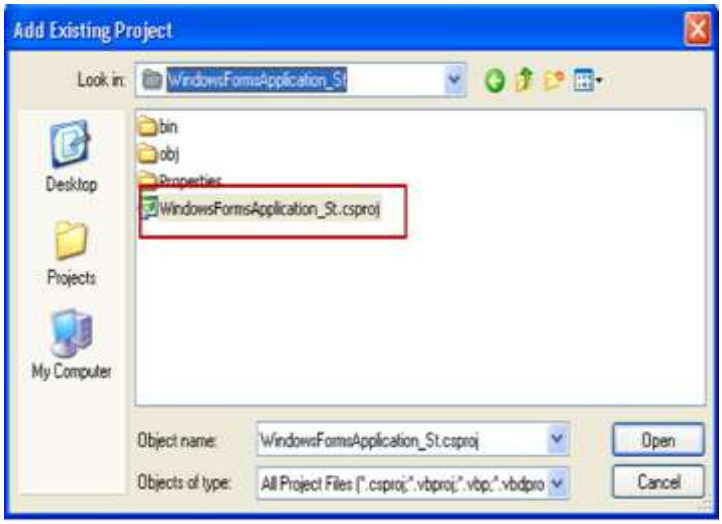


ნახ.36

Setup Project ან Setup Wizard საშუალებებით პროექტის შექმნის შემდეგ, პროექტში უნდა ინტეგრირდეს რეალიზებული პროგრამული პაკეტი ფუნქციით Add – Existing Project (მაუსის მარჯვენა ღილაკი Solution Explorer ფანჯარაში. მიეთითება ფაილი გაფართოებით - .csproj (ნახ.37, 38).

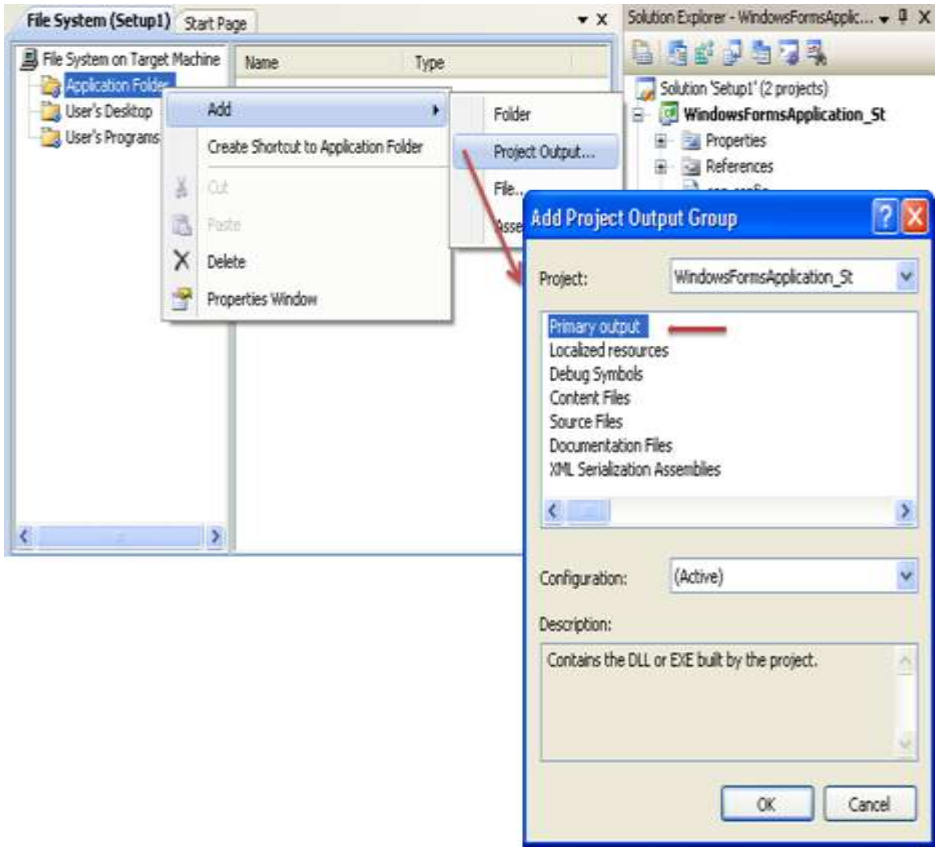


ნახ.37



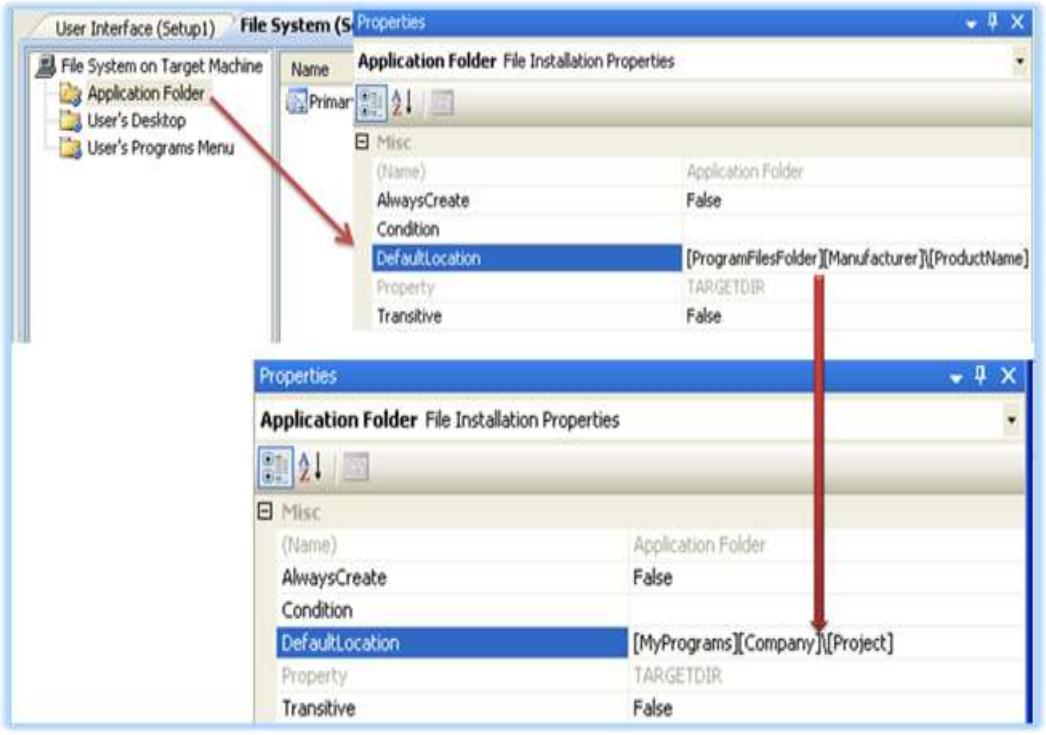
ნახ.38

პროექტის ნაწილში File System საქაღალდე Application Folder განკუთვნილია საინსტალაციო პაკეტისადთვის. ამ საქაღალდეზე ფუნქციით Add (მაუსის მარჯვენა ღილაკი) იხსნება პროექტის შედეგების დასაპროექტებელი ფანჯარა, სადაც რეკომენდებულია ბაზური შედეგის - Primer output მითითება (ნახ.39).



ნახ.39

Application Folder საქაღალდეს თვისებების ფანჯარაში შესაძლებელია დასაინსტალირებელი პაკეტის სახელის მითითებაც (ნახ.40).



ნახ.40

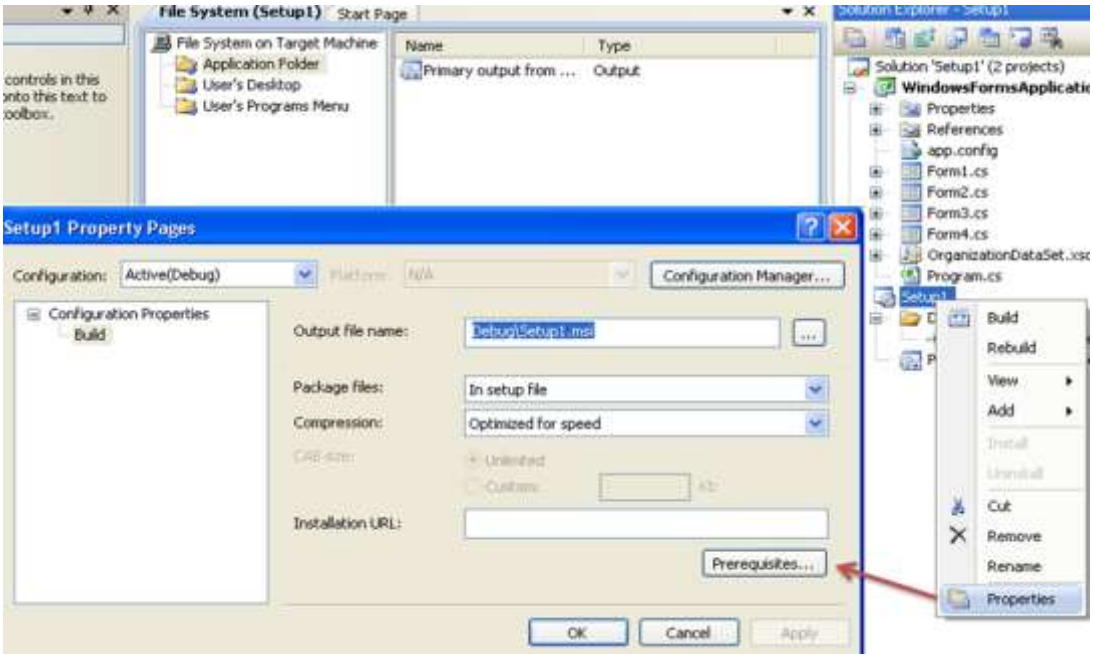
სისტემის ინსტალირების დროს, ხშირად საჭიროა დამხმარე პროგრამების, კომპონენტების ან ბიბლიოთეკების დაყენებაც (მაგ., Windows Installer, .NET Framework, SQL Server Express და სხვ.).

პროექტის Solution Expoler-ის დიალოგურ ფანჯარაში რეალიზებული პროგრამული პაკეტის დამატების შემდეგ ჩნდება ორი პროექტი:

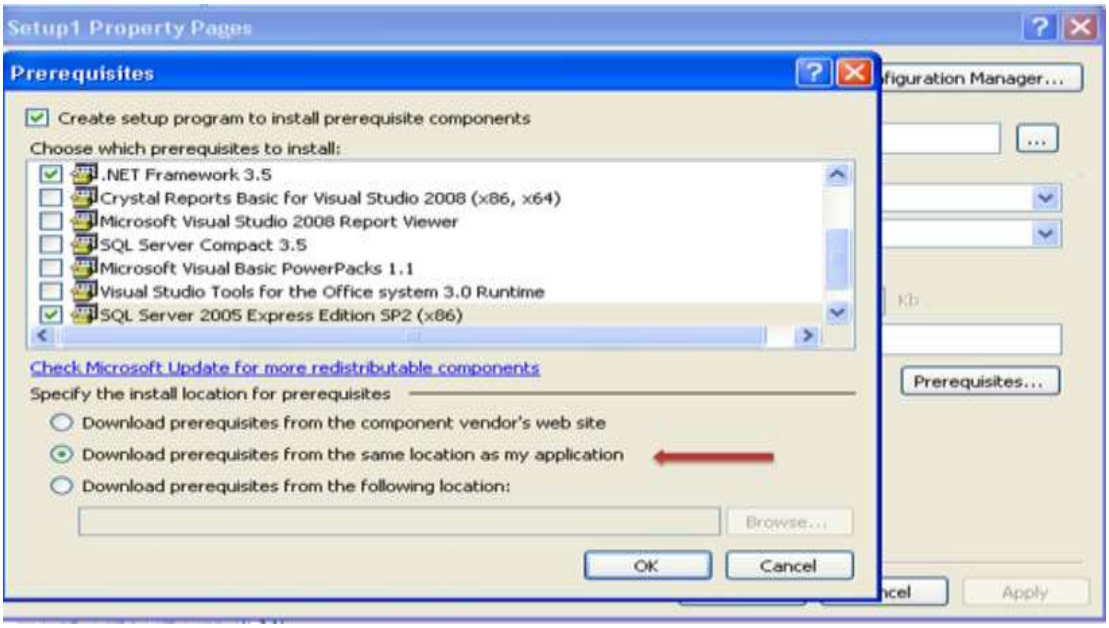
1) მიმდინარე ანუ Setup პროექტი და

2) მიერთებული ანუ რაც უნდა დაინსტალირდეს.

Setup პროექტზე მაუსის მარჯვენა ღილაკის მეშვეობით ფუნქციაზე Property ღილაკით ხდება თვისებების დიალოგური ფორმის გამოძახება, სადაც ღილაკით Prerequisites იხსნება წინაპირობების მისათითებელი დიალოგური ფანჯარა (ნახ.41 და 42).



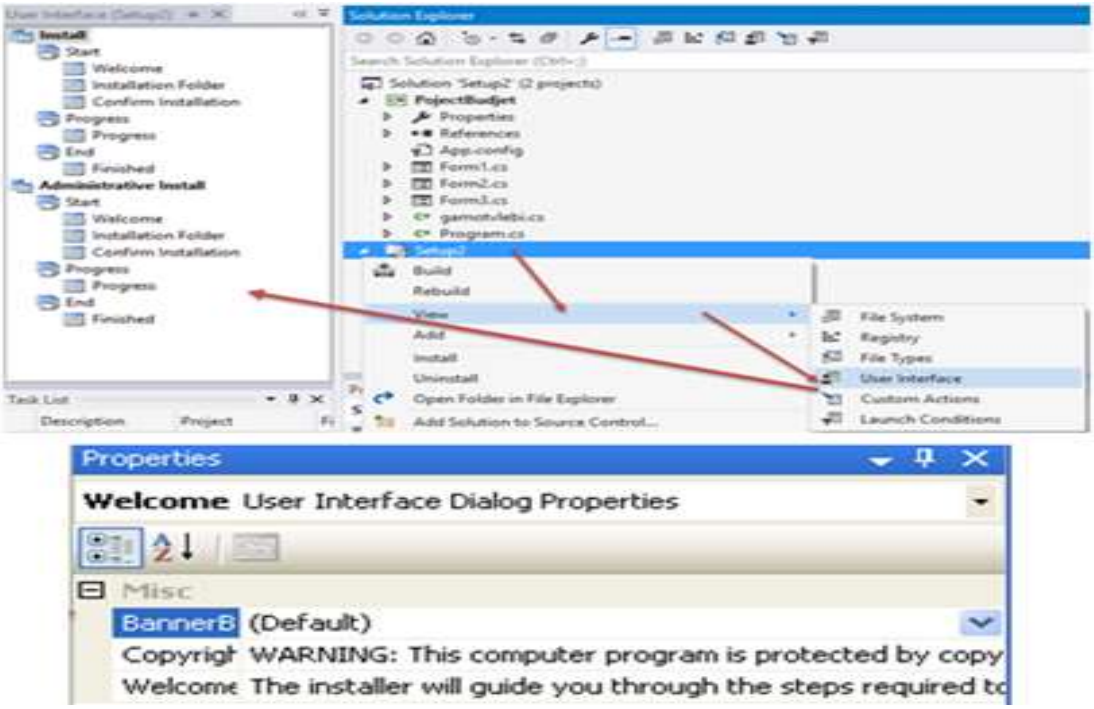
ნახ.41



ნახ.42

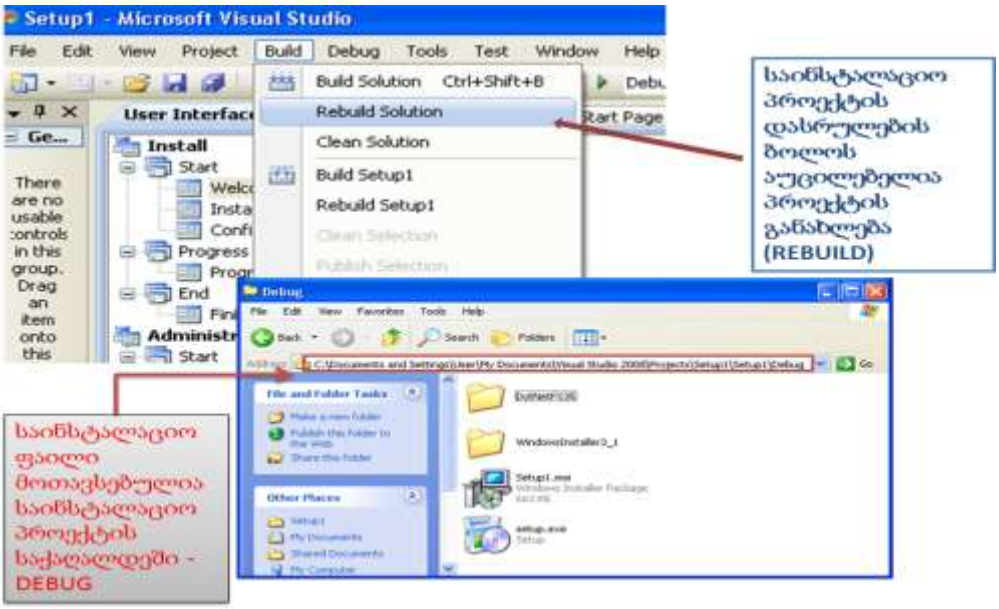
პროექტის ნაწილში File System მოთავსებულია ასევე ორი დამატებითი საქაღალდე User's Desktop (რა განთავსდეს მომხმარებლის სამუშაო მაგიდის საქაღლდეში) და User's Programs Menu (რა განთავსდეს პროგრამული ჩამონათვალის საქაღალდეში).

შესაძლებელია ინსტალაციის პროგრამის დიზაინის ცვლილება (ძირითადად, სურათებით/იკონებით გაფორმება) Setup პროექტზე თაგუნას მარჯვენა ღილაკის მეშვეობით View-User Interface ფუნქციის გამოძახება (ნახ.43).



ნახ.43

საბოლოოდ, აუცილებელია Setup პროექტის განახლება ფუნქციით Rebuild (ნახ.44). Setup.exe გამშვები ფაილი მოთავსებულია Setup პროექტის საქაღალდის ქვესაქაღალდეში - Debug.



ნახ.44

რეკომენდებული ლიტერატურა:

1. „საბაკალავრო პროექტი ინფორმატიკაში“. სილაბუსი - ინფორმატიკისა და მართვის სისტემების ფაკ.-ის საბაკალავრო პროგრამა „ინფორმატიკა“. სტუ. 2020. /ლ.იმნაიშვილი, გ. სურგულაძე, მ.თევდორაძე/.
2. ჩოგოვაძე გ., ფრანგიშვილი ა., სურგულაძე გ. მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. მონოგრ., ISBN 978-9941-20-790-7. სტუ, „ტექნიკ.უნივ.“, თბ., 2017. -1001 გვ., https://gtu.ge/book/monacemta_menejmenti.pdf
3. ჩოგოვაძე გ., სურგულაძე გ., გულიტაშვილი მ., დოლიძე ს. პროგრამული აპლიკაციების ხარისხის მართვა: ტესტირება და ოპტიმიზაცია. მონოგრაფია. ISBN 978-9941-8-0629-2. სტუ. „ITკონსალტინგ ცენტრი". თბ., 2020. -365 გვ. https://gtu.ge/book/Surgu_SoftwareQuality.pdf
4. სურგულაძე გ. კომპიუტერული პროგრამირების მეთოდები და მეთოდოლოგიები (SP, OOP, VP, Agile, UML). ISBN 978-9941-1900-1. სტუ. „IT-კონსალტინგ სამეცნიერო ცენტრი“. თბ., 2019. -200 გვ. https://gtu.ge/book/Surg_ProgMethod_2019.pdf
5. სურგულაძე გ., თურქია ე.. პროგრამული სისტემების მენეჯმენტის საფუძვლები. სახელმძღვანელო. სტუ, თბ., 2016. -350 გვ. http://www.gtu.ge/katedrebi/kat94/pdf/NET_C_40.pdf
6. სურგულაძე გ., გულუა დ., კახელი ბ. პროგრამული აპლიკაციების აგება ვირტუალიზაციის პირობებში. მონოგრ. ISBN 978-9941-8-0627-8. სტუ. „ITკონსალტინგ ცენტრი“. თბილისი. 2019, -159 გვ. https://gtu.ge/book/SurGuluaKakheli_Virtualization.pdf
7. სურგულაძე გ., ურუშაძე ბ. საინფორმაციო სისტემების მენეჯმენტის საერთაშორისო გამოცდილება (BSI, ITIL, COBIT). ISBN 978-9941-20-458-6. სტუ, „ტექნიკ.უნივ.“. თბ., 2014. -320 გვ. ბიბლ.ინდ. 004/23. https://gtu.ge/book/gia_sueguladze/sainfo_sistemebi_BSI_ITIL_COBIT.pdf
8. Sommerville I. Software engineering (Ch.24: Software Quality Management). 10th edition. ISBN 978-0-13-394303-0, published by Pearson Education. 2016. USA. 811 p. Internet resource: http://dinus .ac.id/repository/docs/ajar/Sommerville-Software-Engineering-10ed.pdf (15.05.2020)
9. Fowler A. NoSQL For Dummies®. Published by: John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com Copyright © 2015, New Jersey. -450 p. http://index-of.es/ Miscellanous/LIVRES/Wiley.NoSQL.Mar.2015.ISBN.1118905741.pdf (25.04.2020)
10. Tom White, O'Reilly Media; 4 edition (April 11, 2015), Hadoop: The Definitive Guide 4th Edition, https://www.amazon.com/gp/ product/1491901632/ (15.01.2020)
11. სურგულაძე გ. ქრისტესიაშვილი ხ., სურგულაძე გ. საწარმოო რესურსების მენეჯმენტის ბიზნესპროცესების მოდელირება და კვლევა. მონოგრ., ISBN 978-9941-20-557-6. სტუ, „ტექნიკ. უნივ.“. თბ., 2015. -216 გვ. ბიბლ.ინდ. 004.5/2
12. სურგულაძე გ., ბულია ი. კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. მონოგრ., ISBN 978-9941-20-165-1. სტუ, „ტექნიკ.უნივ.“. თბ., 2012. -324 გვ. ბიბლ.ინდ. 681.327/18
13. სურგულაძე გ., კივილაძე გ. (2017). შესავალი NoSQL მონაცემთა ბაზებში. ISBN 978-9941-0-9642-6. სტუ. „ტექნიკ.უნივ.“. თბ., – 152 გვ., https://gtu.ge/book/NoSQL_Surgul.pdf
14. პეტრიაშვილი ლ., სურგულაძე გ. მონაცემთა მენეჯმენტის თანამედროვე ტექნოლოგიები (Oracle, MySQL, MongoDB, Hadoop). ISBN 978-9941-27-176-2. სტუ. „ITC-ცენტრი”, თბ., 2017. 202 გვ. ბიბლ.ინდ. 004.42(02). https://gtu.ge/book/PetriSurgu_DataManagmTechn.pdf
15. გოგიჩაიშვილი გ., ბოლხი გ., სურგულაძე გ., პეტრიაშვილი ლ. მართვის ავტომატიზებული სისტემების ობიექტ-ორიენტირებული დაპროექტების და მოდელირების ინსტრუმენტები (MsVisio, WinPepsy, PetNet, CPN). სტუ. „ტექნიკური უნივერსიტეტი”. თბ. 2013. -232 გვ. https://gtu.ge/book/ims/GogichaiSurgul.pdf
16. სურგულაძე გ., კაკაშვილი გ., მარტიაშვილი გ. მობილური აპლიკაციების დეველოპმენტის საფუძვლები (Java, Android). ISBN 978-9941-8-2488-3. სტუ. „IT-კონსალტინგ ცენტრი", თბ., 2020. -178 გვ. https://gtu.ge/book/SurgMobAppAndr.pdf
17. DevOps. Internet resoutce: https://en.wikipedia.org/wiki /DevOps

18. Source Code Control System. Internet resource: https://de. wikipedia.org/wiki/Source_Code _Control_System

19. Fowler M. Continuous Integration. Internet resource: 2006. https://martinfowler.com/ articles/continuousIntegration.html

20. The Relationship between Risk and Continuous Testing. 2014. https://www.stickyminds.com/ interview/relationship-between-risk-and-continuous-testing-interview-wayne-ariola

21. Rahm E. Data Warehouses. Einführung. S.2, 2015. Vorlesungsskript, Universität Leipzig, Germany

22. Application Release Automation. Internet resource: https://en.wikipedia.org/wiki/Application-release_automation

23. Infrastructure as code. Internet resource: https://en.wiki¬pedia.org/wiki/Infrastructure_as_code

24. UI vs UX. Internet resource: https://uxplanet.org/what-is-ui-vs-ux-design-and-the-difference-d9113f6612de

25. Борута Я. Методология Agile. Матерь драконов или всех гибких методологий. w-Blog. 2017. https://worksection. com/blog/¬agile.html

26. Beck K. et al. Manifesto for Agile Software Development. 2001. Internet resource; https://agilemanifesto.org/

27. Resources Scrum. Internet resource: 2019. https://www. scrum.org/resources

28. Rumpe B. Agile Modellierung mit UML. Berlin, „Springer“. 2-te Auflage. 2012

29. Lean Software Development - in What is Agile Kanban Methodology ? https://www.inflectra.com/methodologies/kan¬ban.aspx

დანართი_1

პროგრამა „ინფორმატიკის“ **პროგრამული ინჟინერიის** კონცენტრაციის
საბაკალავრო პროექტების განახლებადი თემები (2021 წლიდან)

N	ხელმძღვანელი	თემა
1	სურგულაძე გია	1. სასტუმროს მენეჯმენტის პროსესების სრულყოფის პროგრამული აპლკიაცია 2. სამკურნალო-სარეაბილიტაციო ცენტრის მართვის ავტომატიზებული სისტემის აგება 3. კლიენტ-სერვერული აპლიკაციის დეველოპმენტი პროგრამირების ჰიბრიდული ტექნოლოგიით 4. განაწილებული სისტემის ვებ-დეველოპმენტი სერვის-ორიენტირე-ბული არქიტექტურის ბაზაზე 5. შავი ზღვის ეკომონიტორინგის ავტომატიზებული სისტემის აგება
2	სუხიაშვილი თეიმურაზ	1. ინტერნეტ-მაღაზიის პროექტირება და რეალიზაცია 2. ავიაკომპანიის Web-საითის აგება 3. საქმისწარმოების მართვის კომპიუტერული სისტემის დამუშავება ადმინისტრაციული ორგანიზაციისათვის 4. საკრედიტო მომსახურების მართვის კომპიუტერული სისტემების დამუშავება 5. სასამართლო პროცესების ავტომატიზებული სისტემა
3	მეფარიშვილი ბადრი	1. დიდი ადრონული კოლაიდერის ექსპერიმენტული მონაცემების შენახვის, დამუშავებისა და ანალიზის სისტემა. 2. სადაზღვევო კომპანიაში ავტომანქანების ექსტერიერის დაზიანების

		ქსპერტიზა კონვოლუციური ნეირონული ქსელების გამოყენებით 3. ონლაინ მაღაზიისთვის პროდუქციის კატალოგის განთავსებისა და გაყიდვის პლატფორმა საწყობში არსებული მარაგების გათვალის-წინებით 4. რესტორნების ქსელში მიტანის სერვისის აპლიკაციის შემუშავება
4	ჩაჩანიძე გურამ	1. მცირე ბიზნესის სტრატეგიული ამოცანების ოპტიმალური გადაწყვე-ტილების მიღების საინფორმაციო სისტემის აგება 2. მცირე ბიზნესის მარკეტინგული გადაწყვეტილების ფუნქციონალურ-ტექნოლოგიური სტრუქტურის ფორმირება და პროგრამული რეალიზაცია 3. მცირე ბიზნესის მარკეტინგის ფუნქციონალურ-ტექნოლოგიური სტრუქტურის ფრეიმული მოდელირება. 4. მცირე ბიზნესის ხელშემწყობი კომპიუტერული ინტერაქტიული მარკეტინგის ფორმალიზებული მოდელი.
5	ღვინეფაძე გელა	1. სასწავლო პროცესის მართვის ავტომატიზებული სისტემა საჯარო სკოლებში 2. ინტერდისციპლინური სწავლების ავტომატიზებული მართვა საჯარო სკოლებში 3. გადაწყვეტილებების მიღების მეთოდების ავტომატიზება 4. ცოდნის შეფასების კომპიუტერული სისტემის აგება 5. სასამართლო პროცესების ფორმალიზაცია და ავტომატიზაცია
6	ყაჭიაშვილი ქართლოს	1. ეკოლოგიური მონაცემების სტატისტიკური დამუშავება 2. ნიადაგების დაბინძურების კლასიფიკაცია კლასტერ ანალიზის მეთოდებით 3. დაკვირვების მონაცემებით საპროგნოზო მნიშვნელობების გამოთვლა 4. შესასწავლ პროცესზე არაშემთხვევითი ფაქტორების გავლენის არსებითობის დადგენა 5. პროცესებს შორის შემთხვევითი და არაშემთხვევითი კავშირების დადგენა და ანალიზი
7	ნარეშელაშვილი გულბაათ	1. ქსელური მოდელის გამოყენება მშენებლობის დაგეგმვის პროცესის მართვაში. 2. დინამიური პროგრამირების გამოყენება ინვესტიციების განაწილებაში. 3. მაღაზიის მუშაობის ოპტიმიზაცია რიგების თეორიის გამოყენებით. 4. მარაგების მართვის მოდელის აგება და კვლევა ლოგისტიკის პროცესებისთვის 5. მართვის პროცესის იმიტაციოური მოდელის აგება რიგების თეორიის საფუძველზე
8	ამილახვარი ნუგზარ	1. უსაფრთხო სარეზერვო კოპირება ავტომატიზებულ სისტემებში 2. ფაილების გაცვლა წვდომის კონტროლით და უსაფრთხოების შეფასებით 3. მონაცემთა გაჟონვის მოძიება და მისი შეფასება 4. SQL ინექციების პრევენცია ავტომატიზებულ სისტემებში 5. მონაცემთა განაწილებული ბაზის აგება და მისი უსაფრთხო ფუნქციონირება
9	ცინცაძე ალიკო	1. ძველი ქართულის ასოითი სტატისტიკური და მომენტური მახასიათებლების დადგენის სისტემა კონკრეტული ტექსტისთვის: 2. ძველი ქართულის კრიპტოგრაფიული ალგორითმების სქემების,

		წესებისა და მეთოდების ალბათური ანალიზი; 3. კორელაციური ანალიზის გამოყენებითი სისტემა ძვ. ქართულისთვის ასო-ნიშნებისა და რიცხვ-ნიშნების ჭრილში კონკრეტული ტექსტის მაგალითზე
10	პეტრიაშვილი ლილი	1. პროგრამული უზრუნველყოფის სისტემის დაპროექტება ურბანული ტრანსპორტის აღრიცხვისა და პარკირებისთვის 2. სამომხმარებლო ინტერფეისის დაპროექტება ჭკვიანი ტრანსპორტის ინტეგრირებულად მართვისთვის 3. სასაწყობე პროცესების ოპტიმალურად მართვის ავტომატიზებული სისტემის შემუშავება 4. სატრანსპორტო ნაკადების მართვის და მონიტორინგის IOT სისტემა 5. სააღრიცხვო-ანალიტიკური პროცესების ავტომატიზაცია ფირმაში
11	თურქია ეკა	1. ელექტრონული მაღაზიის სისტემის დამუშავების დაგეგმვა Agile დაპროექტების მიდგომით (backlog, epic, User Stories) 2. მოქნილი და კასკადური დაპროექტების მიდგომების შედარებითი ანალიზი (ეფექტიანობა, რისკები) 3. ტესტირების სცენარების დაპროექტება case ტექნოლოგიის ინსტრუმენტების ბაზაზე
12	ქართველიშვილი იოსებ	1. საქმიანი პროცესების მართვისა და უსაფრთხოების უზრუნველყოფის პროგრამული სისტემის დამუშავება. 2. კორპორაციულ ქსელში გაყიდვებისა და მარაგების მართვის პროგრამული სისტემის აგება ვირტუალური კერძო ქსელის (VPN) გამოყენებით. 3. საცალო ვაჭრობაში ბიზნეს-პროცესების აღრიცხვის უსაფრთხო პროგრამული სისტემის შემუშავება უსადენო ქსელების გამოყენებით. 4. კომპიუტერული ქსელების ინფორმაციულ რესურსებზე მონიტორინგისა და კონფიდენციალური წვდომის პროგრამული სისტემის შემუშავება.
13	თოფურია ნინო	1. ბიზნეს-პროცესების ავტომატიზაცია Office-365 საქფუძველზე 2. კორპორატიული მონაცემების ბიზნეს ანალიტიკა Power BI-ის ბაზაზე. 3. ბიზნეს გადაწყვეტილებების იმპლემენტაცია Microsoft Power Platform-ის ბაზაზე.
14	აბულაძე ინგა	1. სწავლა/სწავლების პროცესის იმიტაციური მოდელის აგება ობიექტ-ორიენტირებული მიდგომით საფუძველზე პანდემიის პერიოდში 2. უნივერსიტეტში შემოსვლის წესის დაცვის იმიტაციური მოდელის აგება პანდემიის პერიოდში 3. ონლაინ სწავლის პროცესში სტუდენტთა მიერ შესრულებული დავალებების შემოწმება პედაგოგის მიერ მოდელირების სისტემის ოო-პროგრამირებით
15	ჯანელიძე გულნარა	1. აპლიკაციის დამუშავება ტურისტული ბიზნესისთვის 2. გაყიდვების მონაცემთა მრავალგანზომილებიანი ანალიზი 3. მანქანური სწავლების მეთოდები გაყიდვების ბიზნეს-პროცესში
16	კაშიბაძე მარინა	1. ონლაინ შეკვეთების ავტომატიზებული სისტემა 2. პაციენტთა დიაგნოსტიკის ცენტრის კლიენტებთან ურთიერთობის მონაცემთა ბაზის დპროექტება და რეალიზაცია 3. ელექტრონული არჩევნების მართვის ავტომატიზებული სისტემის აგება
17	ჩორხაული ნინო	1. წიგნების ონლაინ მაღაზია web ტექნოლოგიების გამოყენებით. 2. web ტექნოლოგიების გამოყენებით მხატვრის სახლის საიტის აწყობა 3. საქართველოს მთის კურორტების საიტი web ტექნოლოგიების

		გამოყენებით
18	კეკენაძე ალექსანდრე	1. ვებ-ტექნოლოგიების როლი მსოფლიო პანდემიის დროს. 2. ვებ-ტექნოლოგიების დანერგვა და გამოყენება განათლების სისტემაში. 3. ვებ-ტექნოლოგიების როლი ბიზნესის განვითარებაში.
19	ბიტარაშვილი მარინე	1. ელექტრონული ონლაინ მაღაზია. 2. ჯანდაცვის ელ. სისტემა - ონლაინ კონსულტაცია და დანიშნულების გაცემა. 3. ტექნიკის ელ. მაღაზია.
20	ოდიშარია კორნელი	1. საწარმოს ინფორმაციული უსაფრთხოება, როგორც მისი ეკონომიკური უსაფრთხოების უზრუნველყოფის ერთ-ერთი უმთავრესი გარანტი 2. საინფორმაციო სისტემებში მონაცემთა დაცვის თანამედროვე ტექნოლოგიები 3. კორპორატიული საინფორმაციო სისტემების დაცვის უზრუნველყოფის თანამედროვე პრობლემები და მათი გადაწყვეტის მეთოდები და საშუალებები; 4. ბრძოლა კიბერთაღლითობების წინააღმდეგ, პრობლემები და პერსპექტივები
21	ოხანაშვილი მაია	1. კლინიკაში პაციენტთა აღრიცხვისა და მომსახურების პროცესების ავტომატიზაცია 2. ფაკულტეტის საქმიანი პროცესების მართვის ავტომატიზებული სისტემის აგება 3. სილამაზის სალონის მენეჯერის ავტომატიზებული სისტემა
22	პოჩოვიანი სიმონ	1. მორწყვის რეჟიმების მართვის ავტომატიზებული სისტემის განვითარების მოდელები. 2. მორწყვის რეჟიმების მართვის ავტომატიზებული სისტემის საინფორმაციო მონაცემთა ბაზა.
23	კაკაშვილი გიორგი	1. სამოგზაურო ტურის დაგეგმვის აპლიკაცია ჰიბრიდული მობილური პროგრამირების გამოყენებით 2. გეოლოკაციური აპლიკაციის შექმნის სწორი გზა Android მოწყობილობებისთვის.

შენიშვნა:

1. საბაკალავრო პროექტი ინდივიდუალურია, სტუდენტი აკეთებს დამოუკიდებლად ხელმძღვანელის დახმარებით. (ჯგუფური პროექტის შესრულება დაუშვებელია).
2. ძირითადი ტექსტი 30 გვ., დანართი (არასავალდებულო 40 გვ.).
3. ერთი სტუდენტის მიერ დაკავებული თემა არ შეიძლება აირჩიოს სხვა სტუდენტმა.
4. ხელმძღვანელთან შეთანხმებით სტუდენტს შეიძლება მიეცეს მოდიფიცირებული (ან ახალი დასახელების) თემა,

საყურადღებო:

საბაკალავრო პროექტის შესრულებისას „პროგრამული ინჟინერიის“ კონცენტრაციის მიმართულებით იხელმძღვანელეთ პროგრამული სისტემების სასიცოცხლო ციკლის (Software Lifecycle) მეთოდოლოგიით. ნაშრომის ორიგინალური მხარე (პრობლემის გადაწყვეტის საკუთარი ხედვა) შეიძლება ეხებოდეს მის ყველა ეტაპს (ჰორიზონტალური კონცეფცია) ან ერთ რომელიმე ეტაპს უფრო ღრმად (ვერტიკალური კონცეფცია).

ეს საკითხი სტუდენტისა და მისი ხელმძღვანელის შეთანხმებით წყდება და დამოკიდებულია სტუდენტის მომზადების დონესა და პროექტის სირთულეზე.

სადემონსტრაციო პროგრამული ნაწილი (დემოვერსია) უნდა ეხებოდეს: Desktop-, Web- ან Mobile Development მიმართულებას.

დანართი_2

საპრეზენტაციო ფაილის და საკურსო პროექტის დოკუმენტაციის მომზადება

საბოლოო შედეგების პრეზენტაციის მიზნით კომპიუტერისა და პროექტორის გამოსაყენებლად შეირჩევა აპლიკაციის პროექტის საილუსტრაციო მასალა: მიზანი, ამოცანები, გადაწყვეტა, შედეგები და რეკომენდაციები. საპრეზენტაციო სლაიდები მომზადდება Ms_PowerPoint ინსტრუმენტით.

პროექტის დოკუმენტაცია მზადდება MsWord ფაილის სახით, ნაბეჭდი A4 ფორმატით Sylfaen ფონტით, 12p და 1.15 ინტერვალით, არეები 2 სმ, ძირითადი ტექსტი 30 გვერდი (საჭიროების შემთხვევაში დანართი 40 გვ.).

გადაეცა წარმოებას 25.03.2021. ხელმოწერილია დასაბეჭდად 30..03.2021. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 2,5. ტირაჟი 50 ეგზ.

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრი“, თბილისი, მ.კოსტავას 77

ISBN 978-9941-8-2927-7