გოჩა ჩოგოვაძე, გია სურგულაძე,
მიხეილ გულიტაშვილი,
სანდრო დოლიძე

პროგრამული აპლიკაციების ხარისხის მართვა: ტესტირება და ოპტიმიზაცია

„სტუ-ს IT კონსალტინგის ცენტრი”

გოჩა ჩოგოვაძე, გია სურგულაძე,
მიხეილ გულიტაშვილი, სანდრო დოლიძე

პროგრამული აპლიკაციების ხარისხის მართვა: ტესტირება და ოპტიმიზაცია

დამტკიცებულია:
სტუ-ს „IT კონსალტინგის
სამეცნიერო ცენტრის “სარე-
დაქციო კოლეგიის მიერ

თბილისი- 2020

უაკ 003.5

განიხილება ინფორმაციული სისტემების პროგრამული აპლიკაციების ხარისხის მენეჯმენტის საკითხები, კერძოდ მათი შექმნის ბიზნეს-პროცესების ობიექტორიენტირებული ანალიზი, დაპროექტება, დეველოპმენტი, ტესტირება, ოპტიმიზაცია და დანერგვა. წარმოდგენილია პროგრამული პროდუქტის ხარისხის შეფასებისა და მართვის მეტრიკების და არსებული ინსტრუმენტების მიმოხილვა საერთაშორისო სტანდარტების საფუძველზე. ხარისხის სრულყოფის მიზნით გამახვილებულია ყურადღება მოდულირი (Unit) ტესტირების გამოყენებაზე სატესტო პროგრამული აპლიკაციების შექმნისას მათი სასიცოცხლო ციკლის მენეჯმენტის ეტაპებზე. განხილულია ტესტირების, ვერიფიკაციის და ვალიდაციის საკითხები, ავტომატური ტესტირების გამოყენების პრაქტიკული ამოცანა Coded UI ტესტ-სცენარი და მისი რეალიზაცია Visual Studio, Microsoft Test Manager და Selenium RC/WebDriver/AutoIT პროგრამული ინსტრუმენტებით, მონოგრაფიაში შეტანილია სტუ-ს UNESCO-ს და მართვის ავტომატიზებული სისტემების კათედრების ერთობლივი სამეცნიერო კვლევის შედეგები. მათი გამოყენება შეუძლიათ ინფორმატიკის სპეციალობის სტუდენტებს და პროგრამული უზრუნველყოფის ხარისხის მენეჯმენტის საკითხებით დაინტერესებულ მკითხველს.

რეცენზენტები:

- პროფ, ეკატერინე თურქია
- პროფ, ნუგზარ ამილახვარი
- ასოც. პროფ. კორნელი ოდიშარია

რედკოლეგია:

ა. ფრანგიშვილი (თავმჯდომარე), მ. ახობაძე, გ. გოგიჩაიშვილი, ზ. ბოსიკაშვილი, ე. თურქია, რ. კაკუბავა, ნ. ლომინაძე, ჰ. მელაძე, თ. ობგაძე, გ. სურგულაძე (რედაქტორი), გ. ჩაჩანიძე, ა. ცინცაძე, ზ. წვერაიძე

ISBN 978-9941-8-0629-2

Gocha Chogovadze, Gia Surguladze,
Mikheil Gulitashvili, Sandro Dolidze

SOFTWARE QUALITY MANAGEMENT: TESTING AND OPTIMIZATION

Issues related to the quality management of information systems software applications, in particular the objective analysis, design, development, testing, optimization and implementation of business processes for their creation, are considered. Provides an overview of software product quality assessment and management metrics and existing tools based on international standards. In order to improve quality, the focus is on the use of unit testing in the development of test software applications during their lifecycle management stages. Issues of testing, verification and validation are discussed. Practical task of using automated testing Coded UI test script and its implementation with VisStudio, Ms Test Manager and Selenium RC/WebDriver/AutoIT software tools, as well as software optimization and reliability issues. The collective monograph contains joint scientific results from the UNESCO- and Automated Control Systems Departments of GTU. They are intended for use by computer science students (undergraduate, master's and doctoral students) as well as readers interested in software quality management.

ISBN 978-9941-8-0629-2

ავტორთა შესახებ:

გოჩა ჩოგოვაძე - საქართველოს მეცნიერებათა ეროვნული აკადემიის აკადემიკოსი. ტექნიკის მეცნიერებათა დოქტორი, სტუ-ს „იუნესკოს“ კათედრის გამგე. UNESCO-ს (პარიზი) გენერალური მდივნის მრჩეველი განათლებისა და კულტურის სფეროში. „მართვის ავტომატიზებული სისტემების“ კათედრის დამაარსებელი, სტუ-ს რექტორი, მრავალი წიგნისა და სამეცნიერო შრომის ავტორი, მათ შორის: „ინფორნაცია“, „ბიოსფერია“, „გლობალანსი“ და სხვ. საქართველოს სრულუფლებიანი ელჩი საფრანგეთსა და ესპანეთში. ევროპის მრავალი უნივერსიტეტის საპატიო დოქტორი.

გია სურგულაძე – სტუ-ს „მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერიის) დეპარტამენტის უფროსი, პროფესორი, ტექნიკის მეცნიერებათა დოქტორი, გაეროსთან არსებული „ინფორმატიზაციის საერთაშორისო აკადემიის (IIA)” ნამდვილი წევრი, სტუ-ს „IT-კონსალტინგის სამეცნიერო ცენტრის“ ხელმძღვანელი, გერმანიის DAAD-ის გრანტის მრავალგზის მფლობელი, ბერლინის ჰუმბოლდტის, ნიურნბერგ-ერლანგენის და სხვა უნივერსიტეტების მიწვეული პროფესორი. 399 სამეცნიერო ნაშრომის ავტორი, მათ შორის 80 წიგნი და 60 ელ-სახელმძღვანელო გამოყენებითი პროგრამული ინჟინერიის სფეროში.

მიხეილ გულიტაშვილი – აკად.დოქტორი (სტუ) „ინფორმატიკის“ სფეროში, Web-აპლიკაციების ტესტირება, ვალიდაცია და ვერიფიკაცია. საქ.ფინანსთა სამინისტროს IT-სპეციალისტი. არის 2-მონოგრაფიის, 20-სამეცნიერო შრომის ავტორი, 3 საერთაშორისო კონფერენციის მონაწილე, მომხსენებელი.

სანდრო დოლიძე – სტუს „პროგრამული ინჟინერიის“ დეპარტამენტის დოქტორანტი, მუშაობის გამოცდილებით Web- და Desctop-სისტემების დეველოპინგის სფეროში: შვეიცარია – Google SE Intern; კანადა Flexdealer (Toptal), Senior Full Stack Developer; თბილისი: „Alta Software“ და „Vidal“ – დეველოპერი.

სარჩევი

შესავალი

კომპიუტერული სისტემების ინდუსტრიის განვითარების ტემპები 21-ე საუკუნეში მნიშვნელოვნად გაიზარდა, როგორც ტექნიკური და ტექნოლოგიური ინოვაციების საფუძველზე, ასევე პროგრამული ინჟინერიის ახალი თაობის, მომხმარებელთა ფართო მასებზე ორიენტირებული ქსელური სოფტების ბაზაზე. ამან გამოიწვია დიდი სოციალური ეფექტი, რაც პირველ რიგში გამოიხატა არა მხოლოდ სხვადასხვა სფეროს კორპორაციათა სრული კომპიუტერიზაციით, არამედ მოსახლეობის ფართო სპექტრის (მიუხედაცად ასაკის, პროფესიის, სქესის, სარწმუნოებისა და პოლიტიკური ორიენტაციისა) ჩართვით ასეთ გლობალურ საინფორმაციო სისტემაში. თითქმის ყველა ოჯახი ფლობს თანამედროვე კომპიუტერულ და მობილურ საშუალებებს.

ამიტომაც მთელ მსოფლიოში განსაკუთრებული ყურადღება ექცევა *ინფორმაციული საზოგადოების* ფორმირების პრობლემებს. ტარდება საერთაშორისო სიმპოზიუმები და კონფერენციები ამ სფეროში. მაღლდება საზოგადოების ცოდნის დონე და იცვლება მისი მენტალიტეტი მრავალფეროვანი ინფორმაციული წყაროების ადვილად წვდომის საფუძველზე. UNESCO-ს განათლების სამომავლო კონცეფციაც სწორედ ამას ითვალისწინებს, რომ მოხდეს უმაღლესი, საშუალო და პროფესიული განათლების სისტემის ინტენსიფიკაცია ახალი ინფორმაციული ტექნოლოგიების ბაზაზე.

წინამდებარე მონოგრაფიული სახელმძღვანელო ეხება პროგრამული აპლიკაციების ხარისხის მართვის პრობლემებს, განსაკუთრებით ტესტირების, ვალიდაციის, ვერიფიკაციისა და ოპტიმიზაციის საკითხებს. კორპორაციათა ბიზნეს-პროცესების საიმედო პროგრამული მოდულების (IT-სერვისების სახით) დამუშავების პროცესში ტესტირება და შეფასების განსაზღვრა მეტად მნიშვნელოვანია. აქ წყდება საკითხი დამუშავებული სერვისების საბოლოო გამოყენების ან მათი შემდეგი, უფრო სრულყოფილი ვერსიის შექმნის შესახებ, რაც კარგად აისახება პროგრამის სასიცოცხლო ციკლის იტერაციულ-ინკრემენტალურ და სპირალურ მოდელებზე.

წიგნის *პირველი თავი* ეხება პროგრამული სისტემების ხარისხის მართვის საბაზო საკითხების განხილვას. კერძოდ, განხილულია პროგრამული აპლიკაციების ხარისხის მოდელები და მათი მართვის მეთოდები. განსაკუთრებით მახვილდება ყურადღება პროგრამების ტესტირების თანამედროვე ტექნოლოგიებზე, შეფასების მეტრიკებსა და კრიტერიუმებზე. მოცემულია ტესტირების ძირითადი პრინციპები, ტიპები და მათი კლასიფიკაცია.

მეორე თავში განხილულია პროგრამული აპლიკაციების ხარისხის მართვის და უსაფრთხოების კონცეფციები. მოცემულია მომხმარებლის ინტერფეისის (UI) და მისი გამოცდილების (UX) ცნებების მნიშვნელობა ინფოსისტემების დაპროექტებისათვის.

მესამე თავში ფართოდაა წარმოდგენილი პროგრამული აპლიკაციების ოპტიმიზაციის პრინციპები და მეთოდები, როგორც არსებული წყაროების ანალიზის საფუძველზე, ასევე ახალი, ავტორთა მიერ შემუშავებული მეთოდები. ოპტიმიზაციის ორიგინალური მემორიზაციის მეთოდი შემუშავებულია რეაქტიული პროგრამირების პრინციპების გამოყენებით ReactJS Hook-ის ბაზაზე.

მეოთხე თავი ეხება Web-აპლიკაციების ავტომატური მატესტირებელი სისტემას Selenium, განხილულია მისი ტესტ-სკრიპტების და ტესტ-სცენარების შექმნის ტექნოლოგია. აღწერილია ტესტირების ძირითადი ქმედებების, ვალიდაციის და ვერიფიკაციის ბრძანებები. გამოკვლეულია AJAX ტექნოლოგიით შექმნილი აპლიკაციების ტესტირებისას წარმოქმნილი პრობლემები ავტომატური ტესტის შესრულების თვალსაზრისით.

მეხუთე თავში გადმოცემულია Selenium Grid ტექნოლოგია სერვისების პარალელური ავტომატური ტესტირებისთვის. მოცემულია Java-ს მატესტირებელი „ფრეიმვორკები“ – JUnit და TestNG, ავტომატური სცენარები პრაქტიკული მაგალითები.

მეექვსე თავი ეხება პროგრამული სისტემების და სერვერული ბაზების დაპროექტებას ASP.NET MVC და ობიექტ-როლური მოდელირების ტექნოლოგიებით, შემდეგ მათ ტესტირებას.

მეშვიდე თავში განხილულია პროგრამული აპლიკაციების გამართვის, გამონაკლისი შემთხვევების გამორიცხვის, unit-ტესტირების და კოდის ხარისხის შეფასების კონკრეტული ამოცანები.

I თავი
პროგრამული სისტემების ხარისხის მართვა

პროგრამული უზრუნველყოფის (აპლიკაციის) ხარისხი არის კომპიუტერული პროგრამული პროდუქტის მახასიათებლებისა და მათი მნიშვნელობების ერთობლიობა, რომელიც ეხება მისი გამოყენების შესაძლებლობას დადგენილი ან სავარაუდო მოთხოვნილებების დასაკმაყოფილებლად (ფაქტობრივი / გეგმიური). აღნიშნული პრობლემის კვლევა მე-20 საუკუნის 60-ანი წლებიდან დაიწყო და დღესაც ძალზე აქტუალურია პროგრამული ინჟინერიის სფეროში.

როგორც ამ სფეროში ცნობოლი შოტლანდიელი მეცნიერი იან სომერვილი (Ian Sommerville) აღნიშნავს, პროგრამული უზრუნველყოფის ხარისხის მენეჯმენტი მოიცავს სამ ძირითად საკითხს [1]:

1) ორგანიზაციულ დონეზე ხარისხის მენეჯმენტის მიზანია ისეთი ორგანიზაციული პროცესებისა და სტანდარტების სამუშაო ჩარჩოს ჩამოყალიბება, რომელიც შედეგად უზრუნველყოფს მაღალი ხარისხის პროგრამულ სისტემას. ეს გულისხმობს იმას, რომ ხარისხის მენეჯმენტის გუნდმა აიღოს პასუხისმგებლობა პროგრამული უზრუნველყოფის შემუშავების პროცესებსა და იმ სტანდარტებზე, რომლებიც ეხება პროგრამულ აპლიკაციას და მის დოკუმენტაციას, სისტემის მოთხოვნების, დიზაინისა და კოდის ჩათვლით;

2) პროექტის დონეზე ხარისხის მენეჯმენტი მოიცავს ხარისხის შეფასების პროცესების გამოყენებას, აღნიშნული

დაგეგმილი პროცესების დაცვის შემოწმებას და პროექტის შედეგების შესაბამისობის უზრუნველყოფას, ზემოხსენებულ პროექტზე გავრცელებულ სტანდარტებთან;

3. ხარისხის მენეჯმენტი პროექტის დონეზე მოიცავს ასევე პროექტის ხარისხის გეგმის შემუშავებას. აღნიშნული ხარისხის გეგმა უნდა განსაზღვრავდეს ხსენებული პროექტის ხარისხთან დაკავშირებულ მიზნებს და უნდა ადგენდეს, თუ რა პროცესები და სტანდარტები იქნება გამოყენებული.

სომერვილი ამახვილებს ყურადღებას პროგრამული უზრუნველყოფის 15 მახასიათებელზე, კერძოდ, ესენია: *უსაფრთხოება (Safety), გაგებადობა (Understandability), პორტაბელურობა (Portability), დაცულობა (Security), ტესტირებადობა (Testability), გამოყენებადობა (Usability), საიმედოობა (Reliability), ადაპტირებადობა (Adaptability), ხელახლა გამოყენებადობა (Reusability), გამძლეობა (Resilience), მოდულაროობა (Modularity), ეფექტუროობა (Efficiency), სიმტკიცე (Robustness), სირთულე (Complexity)* და *სწავლებადობა (Learnability)* [1].

ეს მახასიათებლები უკავშირდება პროგრამული უზრუნველყოფის საიმედოობას, გამოყენებადობას, ეფექტუროობასა და შენარჩუნებადობას. რა თქმა უნდა, საიმედოობა არის პროგრამული სისტემის ხარისხის ერთ-ერთი უმნიშვნელოვანესი მახასიათებელი, მაგრამ მომხმარებლისათვის ასევე მთავარია სისტემის მწარმოებლურობა (Performance – სწრაფქმედება). იგი არ მიიღებს პროგრამულ პროდუქტს, თუ ის საგრძნობლად ნელია.

ამასთანავე, შეუძლებელია რომელიმე სისტემის ოპტიმიზაცია ყველა ზემოხსენებული მახასიათებლით.

მაგალითად, სიმტკიცის გაუმჯობესებამ, შესაძლოა მწარმოებლურობის დონის შემცირება გამოიწვიოს.

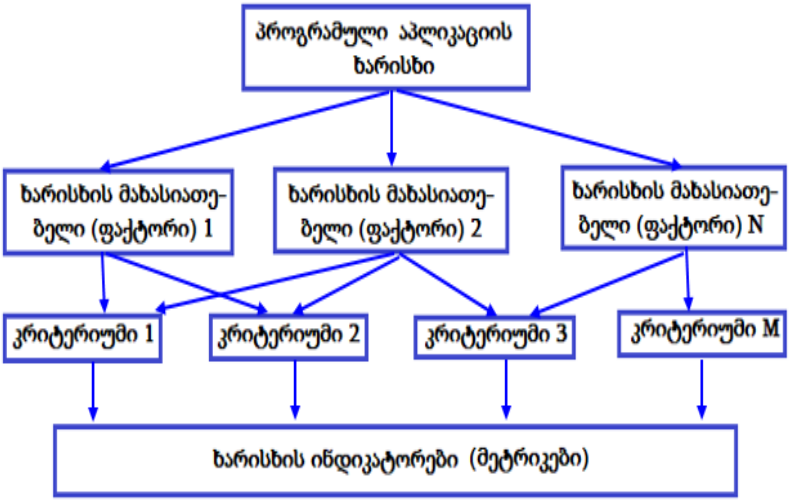
ზემოაღნიშნულიდან გამომდინარე, ხარისხის გეგმა უნდა განსაზღვრავდეს შემუშავების პროცესში არსებული პროგრამული უზრუნველყოფის ყველაზე მნიშვნელოვან ხარისხობრივ მახასიათებლებს. შესაძლოა მიჩნეული იქნეს, რომ გადამწყვეტი მნიშვნელობა აქვს ეფექტურობას და სხვა ფაქტორები უგულებელყოფილი უნდა იქნას ეფექტურობის მისაღწევად.

იმ შემთხვევაში, თუ ხარისხობრივ გეგმაში ზემოხსენებულს მივუთითებთ, შემუშავებაზე მომუშავე ინჟინრებს შეეძლებათ თანამშრომლობა აღნიშნულის მისაღწევად.

გეგმა უნდა მოიცავდეს აგრეთვე *ხარისხის შეფასების* პროცესის განმარტებას. შესაძლოა ეს წარმოადგენდეს იმის შეფასების შეთანხმებულ მეთოდს, ახასიათებს თუ არა პროდუქტს გარკვეული ხარისხობრივი თვისებები, მაგალითად, შენარჩუნებადობა ან სიმტკიცე [1].

1.1. პროგრამული აპლიკაციების ხარისხის მოდელები

პროგრამული აპლიკაციის ხარისხის კონცეფცია უშუალოდ ვერ გამოიყენება პრაქტიკაში, მაგრამ არსებობს მისი შესაბამისი ხარისხობრივი მოდელები, რომლებიც აკონკრეტებს ამ ცნებას და შესაძლებელს ხდის მის შემოტანას ექსპლუატეაციაში. ამგვარად, იქმნება ტერმინებისა და ქვეტერმინების ხე (ქსელი) (ნახ.1.1).

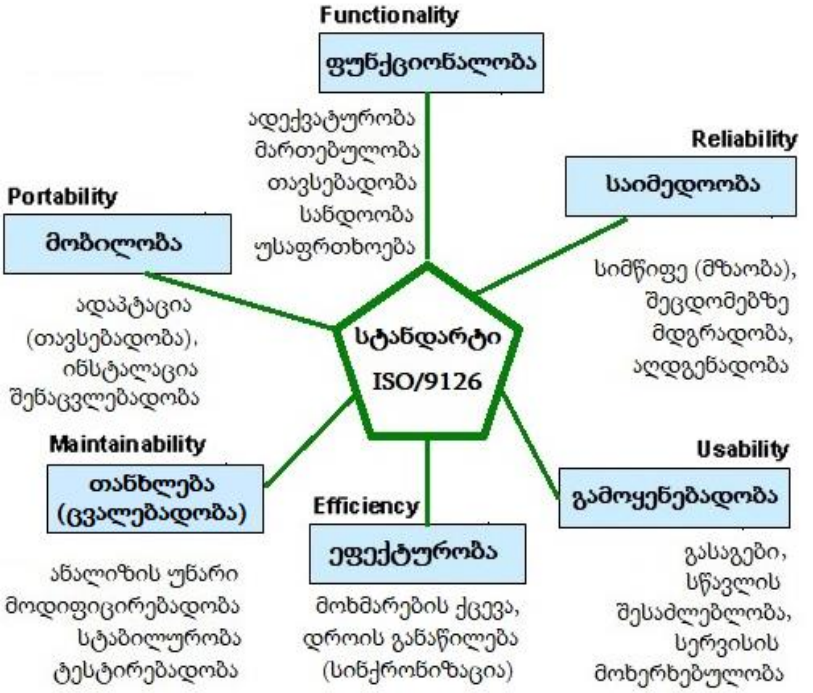


ნახ.1.1. ხარისხის FCM ზოგადი მოდელი
(Factor-Critorium-Metrics)

ISO/IEC 9126 არის პროგრამული უზრუნველყოფის ხარისხის სტანდარტი [4]. იგი ეხება პროგრამული აპლიკაციის, როგორც „პროდუქტის" ხარისხს, აგრეთვე მისი შექმნის პროცესების ხარისხს. ასახავს პროგრამული პროდუქტის ხარისხის ძირითად ფაქტორებს (მახასიათებლებს) და კრიტერიუმებს (ნახ.1.2).

მარტივად რომ ვთქვათ, პროგრამის (ან პროგრამული უზრუნველყოფის) ხარისხი ნიშნავს მასში შეცდომების არარსებობას. ამგავარდ, იგი მიეკუთვნება ფუნქციონალობის კრიტერიუმს.

ხარისხის მახასიათებლები ასახავს განსხვავებულ თვისებებს, რომლებსაც წარმოაჩენს თვით პროგრაული უზრუნველყოფა.



ნახ.1.2. პროგრამული სისტემის ხარისხის მაჩვენებლების საერთაშორისო სტანდარტი

მაღალ დონეზე ეს თვისებებია:

- ფუნქციონალური თვისებები, „პროგრამის ფუნქციების ძირითადი თვისებები“ (რა უნდა გააკეთოს მან და როგორ უნდა იფუნქციონიროს);
- არაფუნქციონალური თვისებები, რომლებიც „ახასიათებს პროგრამული პროდუქტის ქცევას ყოველდღიური გამოყენების დროს“.

არაფუნქციონალური თვისებები, რომლებიც ეხება საიმედოობას, გამოყენებადობას და ეფექტიანობას, კლასიფიცირებულ უნდა იქნას როგორც მოთხოვნები, რომლებსაც პროგრამული პროდუქტი უნდა აკმაყოფილებდეს მისი მუშაობის პროცესში.

თანხლების და მობილობის ხარისხის კრიტერიუმები, პროგრამის (საწყისი ტექსტის) შინაგანი ბუნებით, ხელს უნდა უწყობდეს (ამარტივებდეს) მისი ადაპტაციის პროცესს ახალ გარემოში. ეს კრიტერიუმები ის საფუძველია, რომლებიც უნდა მიეთითოს ყოველ სპეციფიკაციაში თითოეული პროგრამული პროდუქტისათვის, რათა ისინი იქნას გათვალისწინებული პროგრამული უზრუნველყოფის დამუშავებისას.

იმის დასადგენად, რომ პროგრამული პროდუქტი აკმაყოფილებს ხარისხის სხვადასხვა მახასიათებელს, არსებობს პროცესების განსხვავებული მოდელები და მეთოდები.

ზოგიერთი მოდელი შეიძლება მიეკუთვნოს (უფრო ახლოს) პროცესის ხარისხის ცნებას. ეს ნიშნავს, რომ პროდუქტის შექმნის *მაღალი ხარისხის პროცესი* ქმნის *მაღალი ხარისხის პროდუქტს*. ამიტომაც პროცესებს ექცევა განსაკუთრებული ყურადღება.

მაგრამ არსებობს ასევე *პროცედურული მოდელები,* როგორიცაა *Goal-Question-Metric* (GQM) მიდგომა, /მიზანი-კითხვა-მეტრიკა/, რომლებსაც მივყავართ ხარისხის ცალკეულ მოდელებთან.

GQM მოდელი არის ხარისხის კონკრეტული მოდელების შექმნის სისტემატიზებული საშუალება პროგრამული უზრუნველყოფის დამუშავების სფეროში [5]. იგი შეიძლება

წარმოდგენილ იქნას ხისებრი სტრუქტურის სახით. ფესვი - არის მიზანი (Goal), რომელიც ზუსტდება კვანძების (Questions) და ფოთლების (Metric) საშუალებით. ამ გზით მიიღება კითხვები და პროგრამული საზომები. გაზომილი მნიშვნელობები ინტერპრეტირდება გზაზე ფოთლებიდან ფესვისაკენ. შესაფერისი მეტრიკები შეიძლება განისაზღვროს პროგრამული უზრუნველყოფის მეტრიკებში, პასუხის გაცემით შემდეგ კითხვებზე:

- რა მიზანი უნდა იქნას მიღწეული გაზომვის შედეგად ? (Goal);
- რა უნდა გაიზომოს ან რა კითხვებზე უნდა გაეცეს პასუხი ? (Question);
- რომელ მეტრიკას (ან მეტრიკებს) შეუძლია აღწეროს აუცილებელი თვისებები ? (Metric).

ამგვარად, GQM მოდელი აღწერს ხარისხის მოდელის შექმნის პროცედურას, რომლის საშუალებითაც იგი იყოფა ექვს ეტაპად:

1) კომპანიის ბიზნეს-გარემოს და პროექტის დახასიათება, დასმული მისიის დადგენა, გაზომვის მიზნების განსაზღვრა;

2) კითხვების ფორმულირება მიზნების ზუსტად განსაზ-ღვისათვის;

3) გაზომვის მიზნების იდენტიფიცირება მეტრიკების (გაზომვის მახასიათებლების) დადგენა;

4) მონაცემთა შეგროვების მექანიზმების შემუშავება;

5) მონაცემთა შეგროვება, ანალიზი და ინტერპრეტირება;

6) შედეგების შეჯამება და გამოცდილების გამოყენება.

მოკლედ, GQM-პროცესი იწყება ორგანიზაციისა და პროექტის გარემოთა აღწერით. ამ გარემოთა გათვალისწინებით მეორე საფეხურეზე განისაზღვრება ინფორმაციული მოთხოვნილებები მიზნებისა და შესაბამისი კითხვების საფუძველზე.

შემდეგ, მესამე ეტაპზე ხდება გაზომვების დოკუმენტირება, რომელიც ემსახურება ინფორმაციული მოთხოვნების რაოდენობრივ შეფასებას.

მეოთხე და მეხუთე ეტაპებზე ტარდება გაზომვები და მონაცემთა შედეგების ინტერპრეტირება.

ბოლოს, მექვსე ეტაპზე ჯამდება შედეგები. მაგალითად, ამ გზით უზრუნველყოფილია ხარისხის დაგეგმვა და მიღებული ცოდნა.

მიზნები, კითხვები და მეტრიკები შესაბამისი გაზომვებით, დიაგრამებით და სხვა შესაძლებლობებით, ჯამდება GQM-გეგმის სახით. ერთი მიზნისათვის შეიძლება იყოს რამდენიმე კითხვა განსაზღვრული, და ერთი კითხვა - რამდენიმე მახასიათებლისათვის. ანუ, დამოკიდებულება „მიზანსა და კითხვებს“ და „კითხვებსა და მეტრიკებს“ შორის არის 1:N.

მთლიანი GQM მოდელი განისაზღვრება „ზემოდან-ქვემოთ“ („Top-Down“), ანუ იგი მიზანმიმართული მიდგომაა, რომელიც განსაზღვრავს კითხვებს და მეტრიკებს მიზნიდან გამომდინარე. შემდეგ კი, ანალიზი და ინტერპრეტაცია ხდება „ქვემოდან-ზემოთ“ („Bottom-Up“).

ქვემოთ ჩამოთვლილია ხარისხის ზოგიერთი მოდელი:

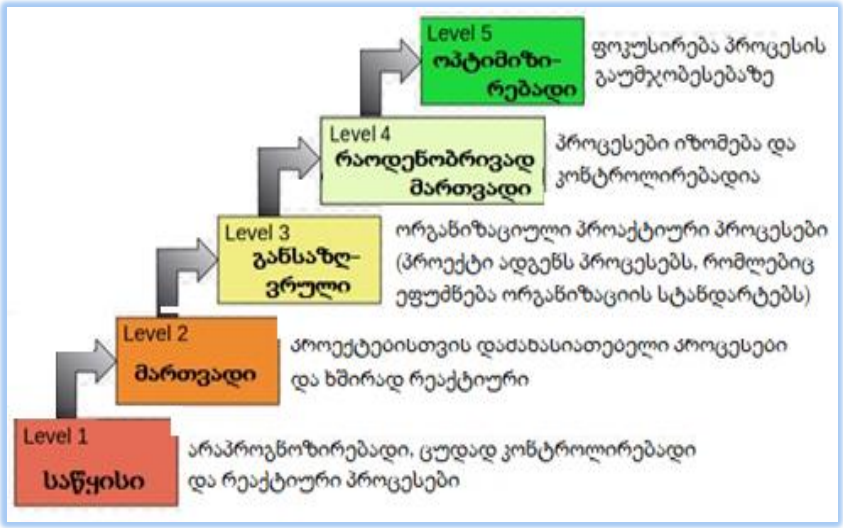
- *ორგანიზაციის ხარისხის მოდელები.* მაგალითად, შესაძლებლობათა სიმწიფის მოდელი (Capability Maturity Model – CMM) ან Automotive SPICE (**S**oftware **P**rocess **I**mprovement and **C**apability d**E**termination – პროგრამული პროცესის გაუმჯობესება და შესაძლებლობების განსაზღვრა). ეს უკანასკნელი შემოტანილია გერმანიის ავტომობილების ინდუსტრიაში [6]:
- *პროცესების მოდელები*, მაგალითად:
  - *შესაძლებლობათა სიმწიფის მოდელის ინტეგრაცია* (Capability Maturity Model Integration (CMMI)) [7]. ესაა ეტალონური მოდელების ოჯახი სხვადასხვა სფეროსათვის გამოსაყენებლად. მაგალითად, პროდუქტების შემუშავებისათვის, პროდუქციის შეძენისა და მომსახურების მიწოდებისთვის.

ინდივიდუალური პროცესის სფეროს შესაძლებლობების დონის გარდა CMMI განსაზღვრავს „სიმწიფის დონეებს“ (maturity levels). სიმწიფის დონე მოიცავს პროცესის მრავალ სფეროს. ისინი უნდა დალაგდეს შესაძლებლობის დონესათან ერთად ისე, რომ შეესაბამებოდეს სიმწიფის დონეს (ნახ.1.3).

სიმწიფის თითოეული დონე განვითარების პლატფორმაა ორგანიზაციის პროცესის გაუმჯობესებაში. CMMI ამრიგად გთავაზობს დახმარებას სრულყოფისთვის, რომელშიც პროცესის სფეროები პრიორიტეტების შესაბამისად განიხილება. სიმწიფის დონეებია:

1) *საწყისი:* მოთხოვნები არაა. ყველა ორგანიზაციას ავტომატურად აქვს სიმწიფის ეს დონე;

2) *მართვადი:* პროექტების მართვა ხდება. მსგავსი პროექტი წარმატებით შეიძლება განმეორდეს;



ნახ.3. შესაძლებლობათა სიმწიფის დონეები

3) *განსაზღვრული:* პროექტები ხორციელდება ადაპტირებული სტანდარტული პროცესის შესაბამისად და არსებობს ორგანიზაციის მასშტაბური უწყვეტი პროცესის გაუმჯობესება;

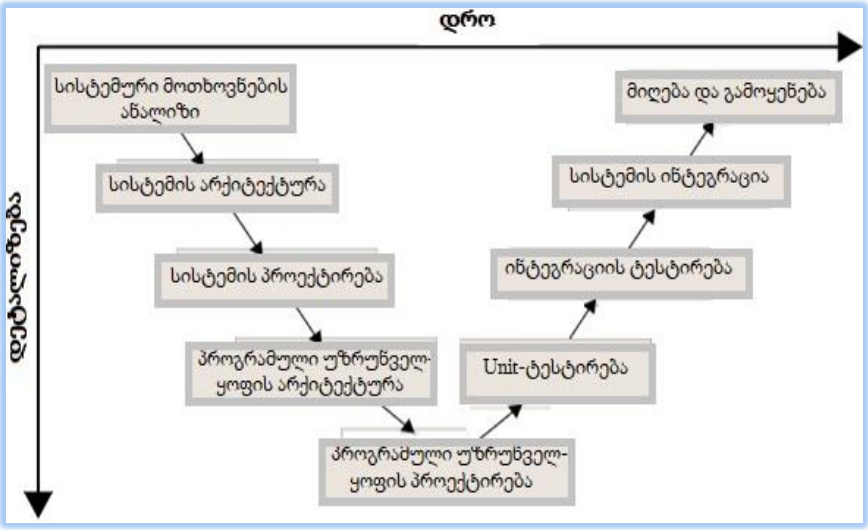
4) *რაოდენობრივად მართვადი:* ხორციელდება სტატისტიკური პროცესის კონტროლი;

5) *ოპტიმიზირებადი*: სამუშაო და მუშაობის მეთოდები გაუმჯობესებულია სტატისტიკური პროცესის კონტროლის დახმარებით.

CMMI მოდელი არის კარგი პრაქტიკის სისტემატური მომზადება ორგანიზაციის გაუმჯობესების ხელშესაწყობად. ამ მოდელის გამოყენება შესაძლებელია საუკეთესო პრაქტიკის შესახებ მიმოხილვის მისაღებად (მაგალითად, პროექტის დაგეგმვაში), ობიექტურად გაანალიზდეს ორგანიზაციის ძლიერი და სუსტი მხარეები, განისაზღვროს გაუმჯობესების ღონისძიებები და დალაგდეს ისინი მათი მნიშვნელობის მიმდევრობით;

- *რაციონალური უნიფიცირებული პროცესი* (Rational Unified Process – RUP) [8]. ესაა IBM კონცერნის Rational Rose ფირმის კომერციული პროდუქტი პროგრამული სისტემების პროექტების მენეჯმენტისათვის UML მეთოდოლოგიის საფუძველზე;
- *V-მოდელი* (V-Modell) [9]. V-Modell არის პროცესის მოდელი, რომელიც „ჩანჩქერის" მოდელის მსგავსად, აყალიბებს პროგრამული უზრუნველყოფის დამუშავების პროცესს ეტაპობრივად. განვითარების ამ ნაბიჯების გარდა, V-Modell ასევე განსაზღვრავს ხარისხის უზრუნველყოფის (ტესტირების) პროცესს (დეველოპმენტის და ტესტირების ფაზების შედარება).

მარცხენა მხარეს იწყება ფუნქციონალური / საგნობრივი სპეციფიკაციებით, რომელიც უფრო დეტალურად აღიწერება მომდევნო ბიჯებზე ტექნიკური სპეციფიკაციებისა და რეალიზების საფუძველზე. პროექტის რეალიზაცია „V"-ს წვერზეა (ნახ.1.4), რომელიც შემდეგ მარჯვენა მხარეს გადის ტესტირებას.



ნახ.1.4. „V“ - მოდელი

ზოგიერთი მეთოდი:

- პროგრამული სისტემის იტერაციული დამუშავება;
- სპირალური მოდელი;
- რეფაქტორინგი;
- ტესტ-ორიენტირებული დამუშავება;
- პროგრამების ტესტირების განსხვავებული სახეები და მეთოდები და სხვ.

შესაძლებელია აგრეთვე ზოგიერთი მოდელის და მეთოდის კომბინირებული გამოყენება.

ბოლო პერიოდში განსაკუთრებით აქტუალური გახდა Agile – მოქნილი პროცესების მოდელები, როგორიცაა, მაგალითად, ექსტრემალური პროგრამირება, Scrum და Kanban [9]. ისინი გამოირჩევა სინერგეტიკული ეფექტებით, მათში სხვადასხვა მეთოდების გამოყენების საფუძველზე.

1.2. პროგრამული უზრუნველყოფის ხარისხის მართვა

პროგრამული უზრუნველყოფის ხარისხის მართვა (Software Quality Management – SQM) არის მენეჯმენტის პროცესი, რომლის მიზანია პროგრამული აპლიკაციების ხარისხის განვითარება და მართვა ისე, რომ პროდუქტი მაქსიმალურად აკმაყოფილებდეს მომხმარებლისთვის საჭირო ხარისხის სტანდარტებს, ასევე მარეგულირებელი ნორმატივებისა და დეველოპერების საჭირო მოთხოვნებს [11].

პროგრამული უზრუნველყოფის *ხარისხის მენეჯერები* ითხოვენ, რომ პროგრამული უზრუნველყოფა შემოწმდეს ბაზარზე გასვლამდე. ისინი ამას აკეთებენ *ხარისხის ციკლური შეფასების პროცესის* გამოყენებით, რათა შეცდომების გამოვლენა და გასწორება მოხდეს პროდუქტის გამოშვებამდე. მათი დანიშნულებაა არა მხოლოდ პროგრამული უზრუნველყოფის კარგ ფორმაში მიწოდება მომხმარებელზე, არამედ, მთელ საწარმოში ხარისხის კულტურის დამკვიდრების პოპულარიზაციის ხელშეწყობა.

პროგრამული უზრუნველყოფის ხარისხის მართვის საქმიანობა ზოგადად იყოფა სამ ძირითად კომპონენტად: ხარისხის უზრუნველყოფა, ხარისხის დაგეგმვა და ხარისხის კონტროლი [12]. ამგვარად:

- *ხარისხის უზრუნველყოფა*: არის ორგანიზაციული ხარისხის სტანდარტებისა და პროცედურების დამკვიდრება;

- *ხარისხის დაგეგმვა*: კონკრეტული პროექტის შესაბამისი ხარისხის სტანდარტების და პროცედურების შერჩევა და შეცვლა;

- *ხარისხის კონტროლი*: ხარისხის სტანდარტებისა და პროცედურების უზრუნველყოფა განვითარების ჯგუფის მიერ.

შენიშვნა: ხარისხის მენეჯმენტი უნდა გამოეყოს პროექტის მენეჯმენტს დამოუკიდებლობის უზრუნველყოის მიზნით [1].

➢ პროგრამული უზრუნველყოფის ხარისხი და პროგრამული უზრუნველყოფის ციკლი

პროგრამული უზრუნველყოფის ხარისხის გაზომვა განსხვავდება მისი წარმოებისგან; ტოლერანტობა არ გამოიყენება და ობიექტური დასკვნები, თუ პროგრამა აკმაყოფილებს სპეციფიკაციებს, რთულია, თუ შეუძლებელი იქნება მისი მიღწევა [11]. ამასთან, პროგრამული უზრუნველყოფის ხარისხი და მიზანთან შესაბამისობის სტატუსი შეიძლება განხორციელდეს სხვადასხვა გზით, რაც დამოკიდებულია ორგანიზაციის და რეალიზებული პროექტის სახეობაზე. ეს მიიღწევა პროგრამის განვითარების მთელი ციკლის მხარდაჭერით, რაც გულისხმობს:

– *მოთხოვნების შეგროვებას და IT პროექტის ფარგლების განსაზღვრას*, ორიენტირებულს გადამოწმებაზე, იქნება თუ არა განსაზღვრული მოთხოვნები ტესტირებადი;

– *გადაწყვეტილების შემუშავებას,* ორიენტირებულს ტესტის პროცესის დაგეგმვაზე, მაგალითად, რა ტიპის ტესტები ჩატარდება და როგორ ჩატარდება ისინი ტესტ-გარემოს და ტესტის მონაცემების კონტექსტში ?;

– *ტესტის შემთხვევებისა და სცენარების საშუალებით მხარდაჭერილი გადაწყვეტის განხორციელებას,* მათ შესრულებას და დეფექტების რეგისტრაციას, მათ შორის დეფექტების მოგვარების კოორდინაციის ჩათვლით;

– *ცვლილებების მენეჯმენტის დანერგვას,* მხარდაჭერით, თუ დაგეგმილმა ცვლილებებმა რამდენად შეიძლება გავლენა იქონიოს შექმნილი გადაწყვეტის ხარისხზე და ტესტის გეგმის საბოლოოდ შეცვლაზე;

– *პროექტის დახურვას,* ტესტების რეალიზაციასთან დაკავშირებით, რომელიც ორიენტირებულია შექმნილი გადაწყვეტის მთლიანი ხარისხის კომპლექსურ შემოწმებაზე.

➢ კავშირები IT მეთოდებთან

პროგრამული უზრუნველყოფის ხარისხის მართვა მკაცრად არის დაკავშირებული პროექტების მენეჯმენტთან, დეველოპმენტთან და IT ოპერაციულ მეთოდებთან, მათ შორის:

- **პროექტების მენეჯმენტის მეთოდები:**

– PRINCE2 (PRojects IN Controlled Environments – პროექტები კონტროლირებად გარემოში) – არის სტრუქტურირებული პროექტის მართვის მეთოდი და პრაქტიკოსი-სპეციალისტების სერტიფიკაციის პროგრამა [13,14]. PRINCE2 პროექტებს ყოფს მართვად და კონტროლირებად ეტაპებად. იგი მიღებულია მსოფლიოს მრავალ ქვეყანაში, მათ შორის დიდ ბრიტანეთში, დასავლეთ ევროპის ქვეყნებსა და ავსტრალიაში;

– PMBOK (Project Management Body of Knowledge – პროექტის მართვის ცოდნის ორგანო) – განსაზღვრავს პროექტის ხარისხის მენეჯმენტის ცოდნის სფეროს და შემდეგ პროცესებს: ხარისხის გეგმა, ხარისხის უზრუნველყოფის შესრულება, ხარისხის კონტროლის შესრულება [15]. PMBOK არის ცოდნის: სტანდარტული ტერმინოლოგიისა და

სახელმძღვანელო პრინციპების ერთობლიობა პროექტის მენეჯმენტისათვის. ცოდნის ერთობლიობა დროთა განმავლობაში ვითარდება და წარმოდგენილია პროექტის მენეჯმენტის ორგანიზაციის მეთოდურ სახელმძღვანელოში, მე-6 გამოცემის წიგნი (2017) არის დოკუმენტი, რომელიც შესრულებულია სამუშაოს ზედამხედველობით. პროექტების მართვის სერტიფიცირებადი ინსტიტუტის მიერ.

- **დეველოპმენტის მეთოდები:**

– RUP (Rational Unified Process) განსაზღვრავს დისციპლინარულ ტესტირებას, რომელიც მოქმედებს ყველა ეტაპზე, დასაწყისიდან – დასრულებამდე, გადასასვლების გათვალისწინებით;

– MSF (Microsoft Solutions Framework) განსაზღვრავს ტესტერის როლს და სტაბილიზაციის ფაზას, რომლებიც ძირითადად ფოკუსირდება გადაწყვეტილების ტესტირებისათვის.

- **Agile (სწრაფი) მეთოდები** ზუსტად ვერ განსაზღვრავს ტესტერის როლს ან პროგრამული უზრუნველყოფის ხარისხის მენეჯმენტთან დაკავშირებულ მექანიზმებს. ეს მეთოდები განსაზღვრავს მხოლოდ ისეთ ტექნიკას, როგორიცაა უწყვეტი ინტეგრაცია და ტესტით-მართვადი დეველოპმენტი.

– CI (Continuous Integration) – უწყვეტი ინტეგრაცია არის ყველა დეველოპერის სამუშაო ასლების გაერთიანების პრაქტიკა ერთ საერთო პროგრამაში (დღეში რამდენჯერმე) [16];

– TDD (Test-Driven Development) – ტესტით-მართვადი დეველოპმენტი არის პროგრამული უზრუნველყოფის შემუშავების პროცესი, რომელიც ეყრდნობა ძალიან მოკლე განვითარების ციკლის განმეორებას: მოთხოვნები გადაიქცევა ძალზე სპეციფიკურ ტესტურ შემთხვევებად [17], შემდეგ კოდი უმჯობესდება ისე, რომ ტესტები გადიოდეს. ეს ეწინააღმდეგება პროგრამული უზრუნველყოფის შემუშავებას, რომლის დროსაც შესაძლებელია ისეთი კოდის დამატება, რომელიც არ შეესაბამება მოთხოვნებს.

- **ოპერაციული მეთოდები:**

– CMMI (Capability Maturity Model Integration - შესაძლებლობათა სიმწიფის მოდელის ინტეგრაცია) განსაზღვრავს პროცესის არეალს PPQA (პროცესისა და პროდუქტის ხარისხის უზრუნველყოფის შესახებ), ეს მოდელი ჩვენ ზემოთ უკვე განვიხილეთ.

– COBIT (Control Objectives for Information and Related Technologies – ინფორმაციული და შესაბამისი ტექნოლოგიების კონტროლის მიზნები) [18]. იგი არის ISACA- ს მიერ შექმნილი ფრეიმვორკი, ინფორმაციული ტექნოლოგიების (IT) მენეჯმენტისა და სახელმწიფო IT-მართვისათვის [19], განსაზღვრავს P08 ხარისხის დამუშავებას [20];

– ITIL (Information Technology Infrastructure Library) – ინფორმაციული ტექნოლოგიების ინფრასტრუქტურის ბიბლიოთეკა არის IT სერვისების მენეჯმენტის (ITSM - Service Management) პრაქტიკის ერთობლიობა, რომელიც ფოკუსირებულია IT სერვისების შესაბამისობაზე ორგანიზაციის ბიზნეს მიზნებთან [18].

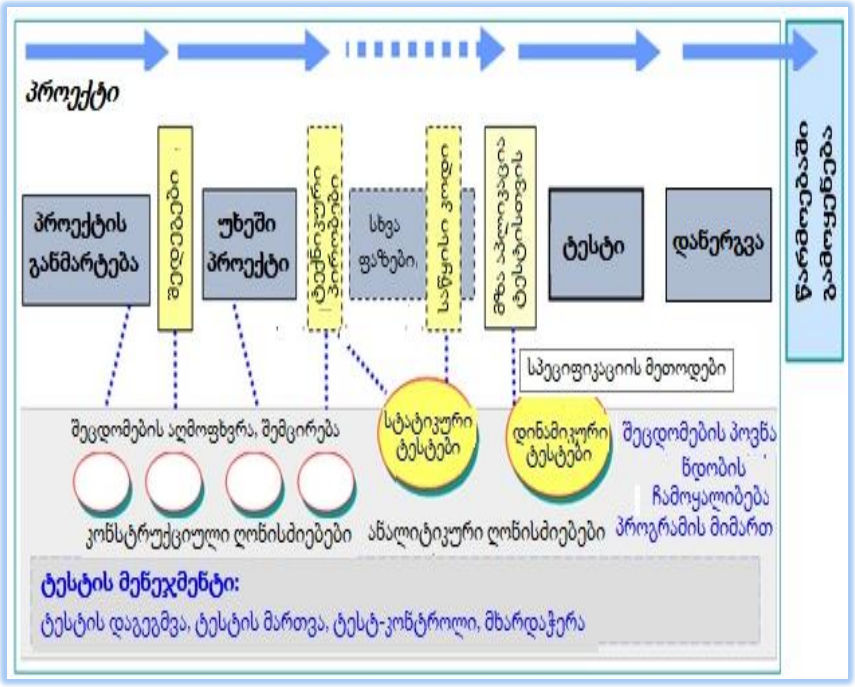
1.3. პროგრამული უზრუნველყოფის ტესტირება

ტესტირება არის პროგრამული უზრუნველყოფის ხარისხის მნიშვნელოვანი ნაწილი პროგრამული უზრუნველყოფის შემუშავების დროს.

შექმნილი ან მოდიფიცირებული / დამატებით შემუშავებული პროგრამული უზრუნველყოფის ხარისხი მოწმდება სხვადასხვა პროცესის გამოყენებით (მაგალითად, საკვანძო სიტყვებზე დაფუძნებული ტესტირება, რისკზე დაფუძნებული ტესტირება, მონაცემებზე დაფუძნებული ტესტირება, ... და ა.შ.), პროცესთა მოდელების, ტესტების ტიპების, ტესტირების დონეების და ა.შ. მიხედვით, მათ რეალურ გამოსაყენებლად გადაცემამდე. ზოგადად შეიძლება ითქვას, რომ ხშირად „ხარისხის შეფასება შეუძლებელია" [2].

ხარისხის უზრუნველყოფის ღონისძიებების შესაბამისად, განასხვავებენ კონსტრუქციულ და ანალიზურ (ანალიტიკურ) ღონისძიებებს (ნახ.1.5) [2].

- *კონსტრუქციული ღონისძიებებია*, მაგალითად, პროექტის სისტემური განმარტება და პროექტის მიზნების განსაზღვრა, აგრეთვე დეტალური მოთხოვნების ანალიზი (დამკვეთის მიერ დამტკიცებული), პროგრამირების დადგენილი ან განსაზღვრული სტანდარტები და ა.შ.
- *ანალიტიკური ღონისძიებები* იყოფა სტატიკურ და დინამიკურ ღონისძიებებად (მისაღებ ზომებად):
    - *სტატიკური ღონისძიებები* (იხ. კოდის სტატიკური ანალიზი, კოდის მიმოხილვა), რომლის დროსაც მოწმდება გენერირებული კოდი, აპლიკაციის რეალურად შესრულების გარეშე.



ნახ.1.5. კონსტრუქციული და ანალიზური ღონისძიებები

ღონისძიებისგან დამოკიდებულებით პროგრამის ანალიზი ხდება სხვადასხვა დროს. მაგალითად, საწყისი კოდის მომზადებისას (იხ. წყვილში პროგრამირება), ან მომხმარებლის ტესტები-სათვის კოდის დამტკიცების წინ;

- *დინამიკური ტესტები,* რომლებშიც გენერირებული აპლიკაცია, ფაქტობრივად, მუშაობს (სრულდება კომპიუტერზე) და მიღებული შედეგები მოწმდება;

ამიტომაც პროგრამული უზრუნველყოფის ხარისხი მისი განვითარების სხადასხვა დროის მომენტში იმყოფება განსხვავებულ მდგომარეობაში და უნდა შეესაბამებოდეს ყველა განსაზღვრულ მოთხოვნა/კრიტერიუმს, როდესაც იგი, ფაქტობრივად, სრულდება შედეგიანად.

1.4. პროგრამების შეფასების მეტრიკების კლასიფიკაცია და კრიტრიუმები

მეტრიკები ემსახურება განვითარებადი პროგრამის (developing program) სხვადასხვა ასპექტს, გამოყენებული პროცესის მოდელს (used process model) და მოთხოვნების შესრულების შეფასებას (requirements execution assessment) [22]. მეტრიკების გამოყენება ვრცელდება პროგრამის განვითარების ფაზების შეფასებიდან: ფაზის შედეგების შეფასებით – გამოყენებული ტექნოლოგიების შეფასებამდე.

მეტრიკების მიზანი პროგრამული უზრუნველყოფის შემუშავებისას არის შეცდომების პროგნოზირება და ხარჯების შეფასება, რის მიხედვითაც განასხვავება ხდება წინა, პარალელურ- და რეტროსპექტულ გამოყენებას შორის.

➢ შეზღუდვა

ძირითადად, მეტრიკები, რომლებიც მართვადია, არის ერთგანზომილებიანი. ეს ამარტივებს მათ გამოყენებას. როგორც წესი, იგი მიიღწევა თითოეული მეტრიკის ერთი ხედვით (view) შემცირებით. ეს ნიშნავს, რომ სხვა ხედვები ამავდროულად არ განიხილება იმავე ხარისხით.

1. *მენეჯმენტის ხედვა:*
   - პროგრამული უზრუნველყოფის განვითარების ხარჯები (შეთავაზება, ხარჯების მინიმიზაცია);
   - პროდუქტიულობის გაზრდა (ბიზნეს-პროცესები ან ბიზნეს-მეთოდები, გამოცდილების მრუდი [23]);
   - რისკები (ბაზრის პოზიცია, დრო ბაზარზე – ახალი პროდუქტის 1-ელი გამოშვებიდან მის ფართოდ დამკვიდრებამდე);
   - სერტიფიკაცია (მარკეტინგი).
2. *დეველოპერის ხედვა:*
   - კითხვადობა – Readability (მომსახურება, განმეორებითი გამოყენება);
   - ეფექტურობა და შედეგიანობა;
   - *ნდობა* (trust) – *შეცდომის კოეფიციენტი* (ErrorQuotient) ხარისხის მენეჯმენტში (ან შეცდომის სიხშირე) არის დეფექტური ელემენტების ფარდობითი წილი მთლიანთან მიმართებაში, ანუ ის ფარდობითი სიხშირე, რომელთან დაკავშირებითაც ხდება შეცდომა პროდუქტის, სერვისის, წარმოების პროცესის ან მუშაობის ხარისხში [24]; *MTBF* (Mean time between failures – საშუალო დრო მტყუნებებს შორის (სთ), მუშაობის დრო მტყუნებამდე (საიმედოობის მაჩვენებელი) [25]);, *ტესტები*);
3. *დამკვეთის ხედვა:*
   - ხარჯთაღრიცხვა (ბიუჯეტის დაცვა, მიწოდების თარიღების დაცვა);
   - ხარისხი (საიმედოობა, სისწორე);
   - ინვესტიციის დაბრუნება (თანხლება, გაფართოება).

➢ კლასიფიკაცია

შეფასების სხვადასხვა ასპექტისთვის არსებობს დიზაინის (პროექტირების) მეტრიკა, ეკონომიკური მეტრიკა, საკომუნიკაციო მეტრიკა და ა.შ. მეტრიკები შეიძლება განისაზღვროს სხვადასხვა კლასისათვის, რომელიც მიუთითებს გაზომვის ან შეფასების ობიექტზე:

1. *პროცესის მეტრიკა:*
   - რესურსების ხარჯი (თანამშრომლები, დრო, ხარჯები);
   - შეცდომა;
   - საკომუნიკაციო ხარჯები.
2. *პროდუქტის მეტრიკა:*
   - ზომა (კოდის სტრიქონები, განმეორებითი გამოყენება, პროცედურები ...);
   - გამოთვილითი სირთულე – არის ინფორმატიკაში ალგორითმების თეორიის ცნება, რომელიც გულისხმობს სამუშაოს მოცულობის დამოკიდებულების ფუნქციას საწყის (შესატან) მონაცემთა რაოდენობასთან [26];
   - კითხვადობა (სტილი);
   - დიზაინის ხარისხი (მოდულარობა, ერთიანობა, შეკავშირება ...);
   - პროდუქტის ხარისხი (ტესტის შედეგები, ტესტური გადაფარვა [27]...).
3. *ხარჯების მეტრიკა:*
   - *ხარჯების სტაბილურობა;*
   - *ხარჯების განაწილება;*

- *პროდუქტიულობა;*
- *ხარჯები - დროული მიწოდება.*

4. *პროექტის ხანგრძლივობის მეტრიკა:*
   - განვითარების (დეველოპმენტის) დრო;
   - განვითარების საშუალო დრო;
   - ეტაპების შესრულების ანალიზი (Milestone Trend Analyse [28]) - პროექტების შესრულების გრაფიკი ეტაპების მიხედვით (პროექტების მენეჯმენტში);
   - პუნქტუალობა (დროული შესრულება).
5. *კომპლექსურობის (სირთულის) მეტრიკა* :
   - პროგრამული უზრუნველყოფის ზომა;
   - დასრულების ხარისხი.
6. *აპლიკაციის მეტრიკა:*
   - ტრეინინგის ხარჯი;
   - მომხმარებლის კმაყოფილება

➢ ხარისხის კრიტერიუმები

მხოლოდ პროგრამული უზრუნველყოფის წარმოების ფაზიდან მიღებული მეტრიკა არ არის ხარისხის კრიტერიუმი. როგორც წესი, ხარისხის მახასიათებლები ფასდება მომხმარებლის მოთხოვნების შესრულებისა და მათი გამოყენების შესაბამისად. შედეგების გადაცემა და გაზომილი მნიშვნელობების ასახვა მნიშვნელოვანია მომხმარებლისათვის:

- *ობიექტურობა*: არ აქვს სუბიექტური გავლენა მზომავი პირისგან;

- *საიმედოობა:* განმეორებისას იგივე შედეგების მიღება;
- *სტანდარტიზაცია:* გაზომვის შედეგების სკალა და შესაბამისობის (შედარების) სკალა;
- *შესაბამისობა:* ზომა (measure - გაზომვა) შეიძლება დადგეს სხვა ზომებთან მიმართებაში;
- *ეკონომიურობა:* მინიმალური ხარჯები;
- *სარგებლობა:* პრაქტიკული მოთხოვნილებების შესრულების გაზომვა;
- *ვალიდურობა:* გაზომვის სიდიდეებიდან სხვა პარამეტრებამდე (სირთულე).

1.5. პროგრამული სისტემების რაოდენობრივი შეფასების მეტრიკები

მეტრიკები – ინსტრუმენტებია, რომლებიც მიზნად ისახავს გადაწყვეტილების მიღების პროცესის გამარტივებას, მწარმოებლურობის და პასუხისმგებლობის დონეთა ამაღლებას, შესაბამისად, დასმული პრობლემის და მისი გადაწყვეტის ამოცანებთან დაკავშირებული მონაცემების შეგროვების, დამუშავების და ასახვის მეთოდების საფუძველზე. შედარებით გამოყენებადი და ცნობილი მეტრიკები შემდეგია:

- *კოდის სტრიქონების რაოდენობა* (Lines Of Code – LOC) [29]. გამოიყენება პროგრამის საწყისი ტექსტის მოცულობის დასადგენად მისი სტრიქონების რაოდენობის განსაზღვრის საფუძველზე. ეს მაჩვენებელი გამოიყენება პროგრამის შემუ-

შავების დანახარჯების პროგნოზირების მიზნით დაპროგრამების კონკრეტულ ენაზე, ან შრომის ნაყოფიერების შესაფასებლად პროგრამის შედგენის შემდეგ;

- *ფუნქციონალური წერტილების ანალიზი* (Function Point Analysis – FPA) – არის მეთოდი, რომელიც რაოდენობრივად განსაზღვრავს პროგრამაში მოცემულ ფუნქციებს იმ ტერმინებით, რომლებიც მნიშვნელოვანია პროგრამის მომხმარებლებისათვის. FP ითვალისწინებს სპეციფიკაციის მოთხოვნების შესაბამისად დამუშავებული (განვითარებული) ფუნქციების რაოდენობას [30].
- *განვითარების მოდელის ღირებულება* (COnstructive COst MOdel – COCOMO) არის პროგრამული უზრუნველყოფის შემუშავების ხარჯთაღრიცხვის ალგორითმი, რომელიც შეიმუშავა ამერიკელმა მეცნიერმა ბ. ბოემ (B. Boehm - სამხრეთ კალიფორნიის უნივერსიტეტი, პროგრამული ინჟინერის პროფესორი). მოდელი იყენებს მარტივი რეგრესიის ფორმულას, რიგი პროექტებიდან შეგროვილი მონაცემების მიხედვით განსაზღვრული პარამეტრებით [31].
- *ციკლომატიურობის სირთულე* (Cyclomatic complexity) არის პროგრამის სირთულის გასაზომი მეტრიკა, მისი სტრუქტურული (ან ტოპოლოგიური) საზომი. პროგრამის საწყის კოდში (ბლოკსქემის ანალოგიურად) იზომება წრფივი, დამოუკიდებელი გზების რაოდენობა დასაწყისიდან დასასრულამდე. იგი შეიმუშავა თომას ჯ. მაკკეიამ 1976 წელს [32]; თუ პროგრამაში არაა ციკლი (მაგალითად, for..., while...), ან პირობითი განშტოება (if...else...), მაშინ ციკლომატიური სირთულე 1-ის ტოლია. თუ არის ერთი if...else... ბლოკი,

მაშინ სირთულე 2-ია (ორი გზა: 1 – true და 2-false -სკენ) და ა.შ. ციკლომატური სირთულე ასევე შეიძლება გამოთვლილ იქნას პროგრამის ფარგლებში ინდივიდუალური ფუნქციების, მოდულების, მეთოდებისა თუ კლასებისთვის. სტრუქტურული პროგრამის მათემატიკური ციკლომატური სირთულე განისაზღვრება ორიენტირებული გრაფის საშუალებით, რომლის წვეროები პროგრამის ბლოკებია, შეერთებული წიბოებით, თუ მართვა შეიძლება გადავიდეს ერთი ბლოკიდან მეორეზე. ამ შემთხვევაში სირთულის განსაზღვრა ხდება ფორმულით [32]:

$$M = E - N + 2P,$$

სადაც:

M - ციკლომატური სირთულეა,
E - წიბოების რაოდენობა გრაფაში,
N - წვეროების რაოდენობა გრაფაში,
P - კავშირის კომპონენტების რაოდენობა.

პროგრამის ტესტირების პროცესში ციკლომატური სირთულე შეიძლება გამოყენებულ იქნას მეორე ამოცანისათვისაც, კერძოდ, რამდენი ტესტი იქნება საჭირო დასატესტი კოდის სრულად დასაფარად.

- *ჰოლსტედის სირთულის ზომები* (Halstead Complexity Measures) – ესაა პროგრამული უზრუნველყოფის რეალიზაციის (დანერგვის) სირთულის შეფასება, წინასწარ, პროექტირების (დიზაინის) ეტაპზე. პროგრამული სირთულის გაზომვის ერთ-ერთი სტატიკური, ანალიტიკური მეთოდია, რომელიც შემოიტანა მ. ჰოლსტედმა (M. Halstead,) 1977 წელს

[33]. მისი კონცეფცია მდგომარეობდა პროგრამული უზრუნველყოფის დეველოპმენტის ემპირიული მეცნიერების შექმნის შესახებ. მან დაადგინა, რომ პროგრამული უზრუნველყოფის მეტრიკები უნდა ასახავდეს რეალიზაციას (დანერგვას) ან ალგორითმების გამოსახვას სხვადასხვა ენაზე, მაგრამ დამოუკიდებელი უნდა იყოს მათი შესრულებიდან კონკრეტულ პლატფორმაზე. ეს მეტრიკები, ამიტომაც, გამოითვლება კოდიდან სტატიკურად. ჰოლსტედის მიზანი იყო პროგრამული უზრუნველყოფის გაზომვადი თვისებების და მათ შორის ურთიერთობების დადგენა [34].

ჰოლსტედის მეტრიკები იყენებს ვარაუდს, რომ შესრულებადი პროგრამის ნაწილები შედგება ოპერატორებისა და ოპერანდებისაგან. მაგალითად, ცვლადები და მუდმივები განიხილება, როგორც ოპერანდები; საკვანძო სიტყვები, ლოგიკური და შედარებითი ოპერატორები და ა.შ., როგორც ოპერატორები.

შემდეგ, თითოეული პროგრამისთვის ფორმირდება ასეთი ძირითადი ზომები:

- $\eta 1$ = განსხვავებული ოპერატორების რაოდენობა;
- $\eta 2$ = განსხვავებული ოპერანდების რაოდენობა;
- $N1$ = ოპერატორთა საერთო რაოდენობა;
- $N2$ = ოპერანდების საერთო რაოდენობა

ამ რიცხვებიდან შეიძლება გამოითვალოს რამდენიმე მაჩვენებელი:

- პროგრამის ლექსიკა: $\eta = \eta_1 + \eta_2$
- პროგრამის სიგრძე: $N = N_1 + N_2$

- გაანგარიშებული პროგრამის სიგრძე: $\widehat{N} = \eta_1 * \log_2 \eta_1 + \eta_2 * \log_2 \eta_2$
- მოცულობა: $V = N * \log_2 \eta$
- სირთულე: $D = \frac{\eta_1}{2} * \frac{N_2}{\eta_2}$
- დანახარჯი: $E = D \times V$

სირთულის ზომა უკავშირდება პროგრამის დაწერის სირთულეს ან მის გაგებას, მაგალითად, კოდის მიმოხილვის დროს. დანახარჯების ზომა გადაიყვანება კოდის ფაქტობრივი დაწერის დროში შემდეგი დამოკიდებულებით;

$$T = \frac{E}{18}$$

ჰოლსტედმა ექსპერიმენტულად დაადგინა, რომ ხარჯების რაოდენობის გაყოფით 18-ზე იძლევა დროის მიახლოებით მნიშვნელობას წამებში [34]. ამ ლიტერატურულ წყაროში განიხილება კოდის სირთულის განსაზღვრის საზომი საშუალებები: C, C ++, Java და C # ენებისათვის.

გაითვლება ასევე მიღებული შეცდომების რაოდენობა (B), რომელიც შეესაბამება პროგრამის მთლიან სირთულეს. აქ ჰოლსტედი იძლევა შემდეგ გამოსახულებას წარმოებული შეცდომების რაოდენობისთვის:

$$B = \frac{E^{\frac{2}{3}}}{3000}$$

ან ბოლო პერიოდის შემთხვევისთვის:

$$B = \frac{V}{3000}$$

- *მართვის ნაკადზე ორიენტირებული მეტრიკები* (Control Flow Oriented Metrics). მართვის ნაკადი (control flow)

პროგრამირებაში არის გამოთვლების (შესრულების) მიმდევრობის მოწესრიგების ხერხი (მაგალითად, განშტოება, ციკლი და სხვ.). იგი სტრუქტურული პროგრამირების ერთ-ერთი ძირითადი საკითხია [10]. ნაკადების მართვაზე ორიენტირებული პროგრამების ტესტირების მეტრიკები (ან გადაფარვის მეტრიკები - Coverage metrics) შეისწავლის: ოპერატორების გადაფარვას (statement coverage), შტოების გადაფარვას (branch coverage), გზების გადაფარვას (path coverage) და პირობების გადაფარვას (condition coverage). ასეთი ტესტირების მეთოდები ეფუძნება პროგრამის ნაკადების მართვის გრაფებს, მაგალითად, ტესტირების თეთრი-ყუთის სახით, ვინაიდან პროგრამის სტრუქტურა აქ ცნობილი უნდა იყოს.

- არსებული მეტრიკების კომბინაციით ვითარდება ახალი მეტრიკები, რომელთა ნაწილი აისახება პროგრამულ ინჟინერიაში. ამის მაგალითია C.R.A.P. (Change, Risk, Analysis, Predictions) (შეცვლა. რისკი. ანალიზი. პროგნოზი) მეტრიკა, რომ შენარჩუნდეს არსებული კოდის ტანი [35]. Unit-ტესტირებისას, მაგალითად, C.R.A.P.-ინდექსი ასახავს თუ რამდენად არის unit-ტესტით დაფარული კოდი. რაც მეტადაა დაფარული, მით მცირეა C.R.A.P.-ინდექსის მნიშვნელობა. მისი თვითნებურად შემცირებამ შეიძლება გამოიწვიოს სისტემის სირთულის გაზრდა, რაც არაა სასურველი.
- *ინფორმაციის უსაფრთხოების მეტრიკა* (Information Security Metrics) – გამოიყენება სისტემებსა და ინფრასტრუქტურებში ინფორმაციის უსაფრთხოების დონის შესაფასებლად, შესაძლებელი უნდა იყოს უსაფრთხოების გაზომვა. უსაფრთხოების ინდიკატორები ობიექტური

რაოდენობრივი საზომია, რომლის საფუძველზე შესაძლებელია გადაწყვეტილების მიღება უსაფრთხოების შესახებ, როგორც მონაცემთა შეგროვების ეტაპზე, ასევე ექსპლუატაციის დროს.

ლიტერატურულ და ინტერნეტულ წყაროებში მრავლადაა გადმოცემული უსაფრთხოების მეტრიკების მნიშვნელობა, განსაკუთრებით ორგანიზაციული მართვის (კორპორაციების) სფეროში [36]. ამ ნაშრომში მოცემულია უსაფრთხოების მეტრიკების მიმოხილვა და მისი განსაზღვრება, მოთხოვნილებები, ატრიბუტები, უპირატესობები, ზომები, ტიპები, პრობლემები/ასპექტები, აგრეთვე კლასიფიკაციის საკითხები. განხილულია უსაფრთხოების მეტრიკების ურთიერთობა რისკის მენეჯმენტთან და ა.შ.

უსაფრთხოების მეტრიკები, რომელთა საზომია მწარმოებლურობა, ძირითადად, კლასიფიცირდება ორ ჯგუფში:

1) უსაფრთხოების მეტრიკები, დაკავშირებული ეფექტურობასთან (შეაფასოს, თუ რამდენად არის მიზნები მიღწეული;

2) უსაფრთხოების მეტრიკები, დაკავშირებული შედეგიანობასთან (რაც ასახავს პროპორციულობას მიღწეულ მიზნებსა და მიღებულ შედეგებს შორის).

უსაფრთხოების მეტრიკის გამოყენება ადასტურებს, რომ ორგანიზაცია იყენებს პროაქტიურ (proactive) უსაფრთხო მეთოდებს. უსაფრთხოების ეს მეტრიკები ინფორმაციას უწევს ორგანიზაციაში დანერგილი პროცესების, პროცედურების და კონტროლის საშუალებების ეფექტიანობას.

1.6. პროგრამის ტესტირების პროცესი, ვალიდაცია და ვერიფიკაცია

როგორც აღვნიშნეთ, ტესტირების პროცესი გვიჩვენებს პროგრამული პაკეტების ობიექტურ სახეს. ტესტირება მოიცავს პროგრამული აპლიკაციის შესრულებაზე გაშვების პროცესს, რომელიც გამიზნულია იპოვნოს პროგრამული შეცდომა ან დეფექტი.

შევნიშნოთ, რომ კარგი ტესტი არის ის, რომელსაც აქვს, მაღალი ალბათობით, შესაძლებლობა, რომ იპოვნოს აღმოუჩენელი შეცდომა [37, 38].

პროგრამული უზრუნველყოფის ტესტირება ასევე არის პროცესი, რომლის დროსაც მოწმდება (ვერიფიცირდება) და მტკიცდება (ვალიდირდება) მოვლენა, რომ კომპიუტერული აპლიკაცია:

- არის დამკვეთის მოცემული მოთხოვნების შესაბამისი;
- მუშაობს აღწერის (სპეციფიკაციების) შესაბამისად;
- შესაძლებელია შევიდეს ხმარებაში (დაინერგოს);
- აკმაყოფილებს დაინტერესებული მხარის (დამკვეთის) მოთხოვნებს.

რას ნიშნავს პროგრამის ვალიდაცია და ვერიფიკაცია ?

ვალიდაცია არის პროცესი, რომლის დროსაც მოწმდება რომ პროდუქტის დიზაინი აკმაყოფილებს, ეთანადება დასმულ მოთხოვნებს, ანუ პროგრამული უზრუნველყოფა აკმაყოფილებს მომხმარებლის (დამკვეთის) მოთხოვნებს.

ვერიფიკაცია კი არის პროცესი, რომლის დროსაც პროგრამული უზრუნველყოფა მოწმდება შექმნის ეტაპზე,

ანუ აკმაყოფილებს თუ არა იგი წინასწარ განსაზღვრულ სტანდარტულ მოთხოვნებს (სპეციფიკაციებს).

ვალიდაცია და ვერიფიკაცია არ არის ერთიდაიგივე, ხშირად მათზე არასწორი შეხედულება აქვთ. მათ შორის განსხვავება შემდეგნაირად ჩამოყალიბდა [39]:

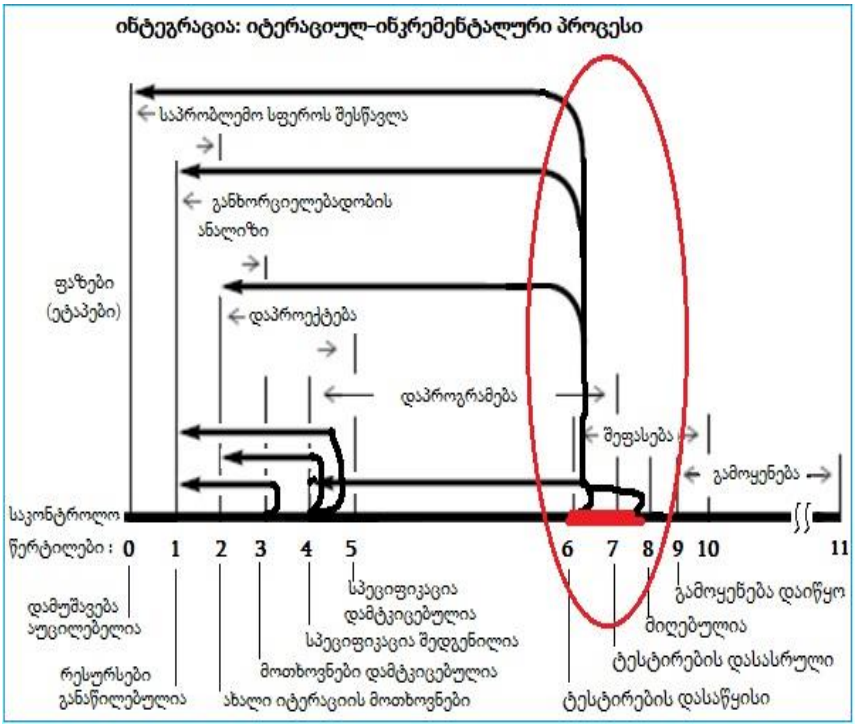
- ვალიდაცია:- ვქმნით ჩვენ *სწორ პროდუქტს* ?
- ვერიფიკაცია: - *ვქმნით* ჩვენ პროდუქტს *სწორად* ?

ამგვარად, ვალიდაცია უზრუნველყოფს, რომ პროდუქტი ეთანადებოდეს მომხმარებლის მოთხოვნებს. ვერიფიკაცია კი უზრონველყოფს, რომ პროდუქტი შეიქმნას მოთხოვნების შესაბამისად.

პროგრამული უზრუნველყოფის შექმნის პროცესის მართვა მისი სასიცოცხლო ციკლის საფუძველზე, მათ შორის ტესტირების ეტაპიც (საკონტროლო წერტილი იხ. ნახ.1.6) უნდა გამოვიყენოთ ყოველთვის, საიმედო პროგრამის მისაღებად.

ტესტირებას არ შეუძლია გამოავლინოს პროგრამის ყველა დეფექტი. მისი ძირითადი მიზანია, რომ გამოავლინოს პროგრამული შეცდომები, რათა მოხდეს აღმოჩენილი დეფექტების გასწორება. პროგრამული კოდის ტესტირებისას, რომლის დროსაც მისი გაშვება ხდება სხვადასხვა გარემოში, მოწმდება პროგრამული კოდის პირობები: აკეთებს ის იმას, რაც არის წინასწარ განსაზღვრული რომ გააკეთოს ? რა რესურსები სჭირდება მას ამის გასაკეთებლად და ა. შ. თანამედროვე ეპოქაში პროგრამული უზრუნველყოფის შექმნის პროცესში, ტესტირების ორგანიზება შეიძლება განცალკევებული იყოს პროგრამირების ჯგუფისგან, ე.წ.

ტესტირების ჯგუფი [1]. განარჩევენ რამდენიმე როლს ტესტირების ჯგუფში: პროექტის მენეჯერი, ჯგუფის ლიდერი, ტესტერი და სხვ.



ნახ.1.6. პროგრამული სისტემის სასიცოცხლო ციკლი ტესტირებით

ყველა პროგრამულ პროდუქტს აქვს გამოყენების შესაბამისი სფერო. მაგალითად, ვიდეო თამაშების პროგრამული უზრუნველყოფის სფერო არის მთლიანად განსხვავებული საფინანსო ბანკის პროგრამის გამოყენების სფეროსგან. მაშასადამე, როდესაც კომპანია ქმნის ან

ინვესტიციას დებს პროგრამულ პროდუქტში, იგი შეფასდება, თუ როგორ მოერგება მომხმარებელთა მოთხოვნებს, თუ იქნება მისაღები პროგრამის გამოყენების სფეროსთვის, მისი მყიდველებისთვის და დაინტერესებული პირებისთვის.

პროგრამების ტესტირება არის სწორედ ამ კრიტერიუმებით შეფასების პროცესი [37,40]. ზემოაღნიშნულის გარდა არსებობს ტესტირების განსხვავებული განსაზღვრებები. მაგალითად, ტესტირება არის პროცესი, როდესაც მოწმდება კომპიუტერული პროგრამის სისწორე, სისრულე და ხარისხი.

ტესტირება არის „კომპიუტერული პროდუქტის დაკითხვის პროცესი იმისთვის, რომ შეფასდეს იგი". ტესტერი კითხვებს უსვამს აპლიკაციას და პროგრამა პასუხობს რაღაც რეაქციით [41].

კომპიუტერული პროგრამის ტესტირებისას ხარისხის განმსაზღვრელი ატრიბუტებია:

- საიმედოობა;
- სტაბილურობა;
- შენარჩუნებადობა.

კომპიუტერული აპლიკაცია შეიძლება იყოს მცდარი, რადგან იგი შექმნილია ადამიანის მიერ. ამიტომ, პროგრამის შექმნის თანმხლები პროცესი უნდა იყოს ხარისხის კონტროლი.

პროგრამისტი, საშუალოდ ხარჯავს თავისი დროის 40% ტესტირებაზე და დანარჩენ დროს ახმარს პროგრამის შექმნას. თუმცა არსებობს ისეთი სპეციალური პროგრამები, რომელთა ტესტირებაც 3-5 ჯერ ძვირი ჯდება ვიდრე მათი შექმნა. მაგალითად სიცოცხლისთვის საშიში პროგრამები (ფრენის

კონტროლი, რეაქტორის მონიტორინგი, სამედიცინო აპარატურა), რომელთა ტესტირებაც გაცილებით მეტ რესურსს მოითხოვს ვიდრე სხვა მოქმედებები.

1.7. ტესტირების ძირითადი პრინციპები

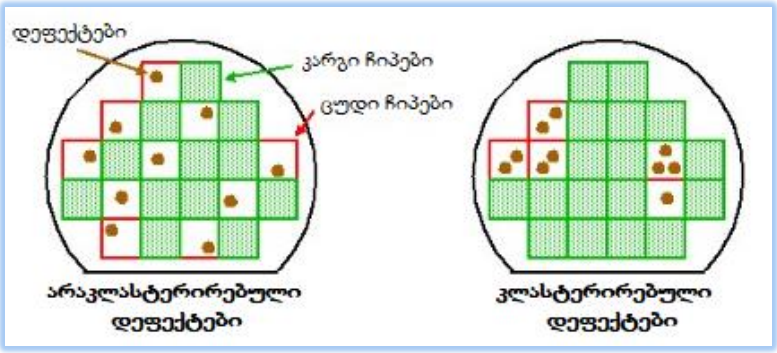
პროგრამული უზრუნველყოფის ტესტირების დროს ძალზე მნიშვნელოვანია ოპტიმალური ტესტის შედეგების მიღწევა, მიზნისგან გადახრის გარეშე. მაგრამ როგორ უნდა განისაზღვროს, რომ არჩეული სტრატეგია სწორია. ამისათვის საჭიროა ტესტირების რამდენიმე ძირითადი პრინციპის დაცვა. ქვემოთ მოცემულია ტესტირების შვიდი ზოგადი პრინციპი, რომლებიც ფართოდ არის გამოყენებული პროგრამული უზრუნველყოფის ინდუსტრიაში [42,43].

1) *ტესტირება აჩვენებს დეფექტების არსებობას* : ტესტირება აჩვენებს დეფექტების ანუ ნაკლოვანებების არსებობას. ყოველი პროგრამული პროდუქტი ან აპლიკაცია გადაეცემა წარმოებას მას შემდეგ, რაც იგი გაივლის საკმარისი რაოდენობის კონტროლს ტესტირების სხვადასხვა გუნდებში, სხვადასხვა ეტაპზე ხდება მათი ტესტირება, მაგალიტად, როგორიცაა სისტემის ინტეგრაციის ტესტირება, მომხმარებელთა მიღების ტესტირება და ბეტა ტესტირება და ა.შ.

2) *ამომწურავი ტესტირება შეუძლებელია*: ხშირად ტესტირების რაოდენობა საგრძნობლად დიდია, აქვს კომბინატორული ხასიათი. საჭიროა ტესტირების ოპტიმიზაცია, ანუ ტესტირების ოპტიმალური რაოდენობის განსაზღვრა აპლიკაციის რისკის შეფასების საფუძველზე. როგორ უნდა განისაზღვროს ეს რისკი - ამაშია პრობლემა! ერთ-ერთი

გადაწყვეტაა გაიხსნას ერთდროულად 10 სხვადასხვა აპლიკაცია და პროცესები წარიმართოს პარალელურად. დეფექტების აღმოჩენა შესაძლებელია მულტიამოცანების რეჟიმშიც და საჭიროა თითოეულის დეტალური ანალიზი. ამას კი მივყავართ შემდეგ პრინციპამდე – დეფექტების კლასტერიზაციამდე;

3) *დეფექტების კლასტერიზაცია:* გულისხმობს მოდულების მცირე რაოდენობას, რომელიც შეიცავს დეფექტების უმეტესობას. ძირითადად, დეფექტები ერთნაირად არ არის განაწილებული მთელს აპლიკაციაში, უფრო მეტიც, ისინი კონცენტრირდება ან ცენტრალიზებულია რამდენიმე ფუნქციონალში (ნახ.1.7) [44].



ნახ.1.7. დეფექტების კლასტერიზაციის ზოგადი ფრაგმენტი

ზოგჯერ ეს ხდება აპლიკაციის სირთულის გამო, კოდირება შეიძლება იყოს რთული ან დამაბნეველი (ჩახლართული), დეველოპერმა შეიძლება დაუშვას შეცდომა, რაც გავლენას მოახდენს მხოლოდ კონკრეტულ ფუნქციონალზე ან მოდულზე.

დეფექტების კლასტერიზაცია ემყარება „პარეტოს პრინციპს", რომელიც ასევე ცნობილია როგორც 80-20 წესი. ეს ნიშნავს, რომ აღმოჩენილი დეფექტების 80% განპირობებულია აპლიკაციაში არსებული მოდულების 20%-ით. პარეტოს პრინციპის ცნება თავდაპირველად განსაზღვრა იტალიელმა ეკონომისტმა - ვილფროდო პარეტომ (Vilfrodo Pareto) [45].

4) *„პესტიციდების“ პარადოქსი:* სოფლის მეურნეობაში გამოიყენება ქიმიური ხსნარები - პესტიციდები, რომლითაც ებრძვიან მავნებლებს, მღრღნელებს და სარეველა ბალახებს. დროთა განმავლობაში ისინი ეგუებიან პესტიციდებს და აღარ ნადგურდებიან... ასევეა პროგრამულ ტესტირებაშიც. კლასტერების ეტაპზე აღმოჩენილი დეფექტები და გაწმენდილი კოდი, არსებული ტესტ-პროგრამების ხშირად გამოყენების შემდეგ ვეღარ აღმოაჩენს ახალ დეფექტებს, ისინი „უძლურნი“ ხდება. ამიტომ საჭიროა ახალი ტესტ-ფუნქციების შექმნა და მათი გამოყენება, ეს პროცესი მუდმივია ! ე.ი. დეფექტების სრული აღმოფხვრა და უშეცდომო, იდეალური კოდის არსებობა არ არსებობს !

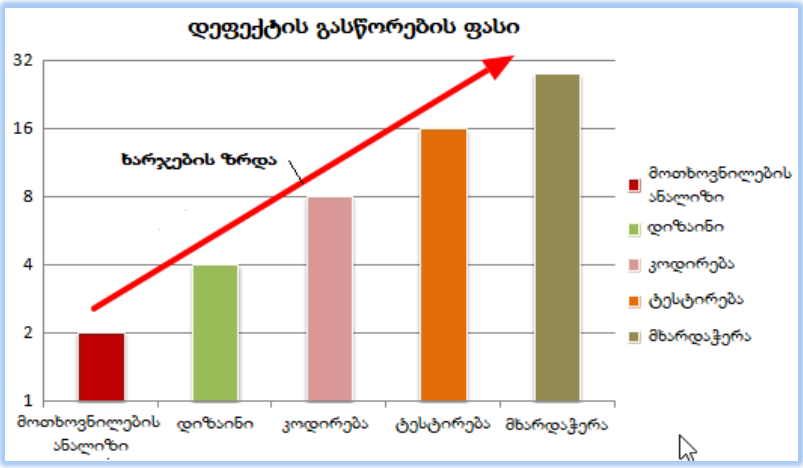
მაშასადამე, სისტემატიურად უნდა მზადდებოდეს ახალი ტესტი, რათა შესაძლებელი იყოს აპლიკაციის დანარჩენი (ან უკვე დატესტილი) მოდულების ახალი პოტენციური დეფექტების აღმოჩენა. მთლიანი ტესტქეისების რაოდენობა მნიშვნელოვნად მატულობს და მოითხოვს უფრო მეტ ხარჯებს და შესრულებისთვის საჭირო დროს. ეს აშკარად მოქმედებს პროექტის ვადებზე და, რაც მთავარია, პროექტის ბიუჯეტზე.

ამ პრობლემის დასაძლევად, ზედმეტი ტესტ-ქეისების განხილვა და შემდეგ ამოღება შესაძლებელია. არსებობს მრავალი ტესტ-ქეისი, რომლებიც უსარგებლო ხდება ახალი ტესტის დამატებისა და არსებულის შეცვლის შემდეგ. მაშინ, ტესტ-ქეისების მთლიანი რაოდენობის შემცირებასთან ერთად უზრუნველყოფილია ბიზნესის ყველა მოთხოვნა. შესაბამისად, ხარისხის კომპრომისი არ არსებობს.

5) *შეცდომის არარსებობა – სიცრუეა:* შესაძლებელია, რომ პროგრამული უზრუნველყოფა, რომელიც 99% -ით უშეცდომოა, ჯერ კიდევ გამოუსადეგარია. ეს შეიძლება იმ შემთხვევაში მოხდეს, თუ სისტემა საფუძვლიანად არის გამოცდილი არასწორი მოთხოვნისთვის. პროგრამული უზრუნველყოფის ტესტირება არა მხოლოდ დეფექტების პოვნაა, არამედ მისი გადამოწმებააცაა, რომ იგი აკმაყოფილებს ბიზნესის მოთხოვნებს. ანუ, დეფექტის არარსებობის მტკიცება არის შედომა თუ სისტემა უვარგისია გამოსაყენებლად, ანუ ვერ აკმაყოფილებს დამკვეთის მოთხოვნებს. ამ პრობლემის გადასაჭრელად, ტესტირების შემდეგი პრინციპს - ადრეული ტესტირებას შევეხოთ;

6) *ადრეული ტესტირება* : ტესტირება უნდა დაიწყოს რაც შეიძლება ადრე, პროგრამული უზრუნველყოფის განვითა-რების სასიცოცხლო ციკლში. ისე, რომ მოთხოვნების ან დიზაინის ფაზის ნებისმიერი დეფექტი ადრეულ ეტაპზე იყოს გამოვლენილი. ტესტირების ადრეულ ეტაპზე გაცილებით იაფია დეფექტის გამოსწორება. რამდენად ადრე უნდა დაიწყოს ტესტირება ? მიზანშეწონილია, რომ შეცდომის პოვნა დაიწყოს მოთხოვნების განსაზღვრის

მომენტში. 1.8 ნახაზზე ნაჩვენებია პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის ეტაპების შესაბამისად ტესტირების პროცესის ხარჯების ზრდის დინამიკა მარცხნიდან მარჯვნივ, ანუ თუ როგორ იზრდება დეფექტების გამოსწორების ღირებულება: მოთხოვნების ანალიზის, დიზაინის, დეველოპმენტის, ტესტირებისა და დანერგვის შემდეგ მხარდაჭერის ეტაპებზე [45].



1.7. ტესტირების ხარჯების დინამიკა აპლიკაციის განვითარების სასიცოცხლო ციკლის მიხედვით

7) *ტესტირება დამოკიდებულია კონტექსტზე.* კომპიუტერულმა ტექნიკამ და ინფორმაციულმა ტექნოლოგიებმა დაიპყრო თითქმის ყველა სფერო (domain), სახელმწიფო და კერძო ბიზნესის ორგანიზაციები. პროგრამული აპლიკაციების ბაზარზეც წარმოდგენილია შესაბამისი სფეროების მხარდამჭერი სისტემები, მაგალითად, როგორიცაა: საბანკო, სადაზღვევო, სამედიცინო, განათლების, სამართალდაცვის,

სოფლის მეურნეობის, სატრანსპორტო, ტურისტული, სარეკლამო და მრავალი სხვ. თითოეულ სფეროს აქვს იერარქიულად დაქვემდებარებული მრავალი ობიექტი და, შესაბამისად, მრავალი პროგრამული აპლიკაცია. თითოეული დომეინის პროგრამებს აქვს თავისი განსხვავებული მოთხოვნები, ფუნქციები, სხვადასხვა ტესტირების მიზანი, რისკი, ტექნიკა და ა.შ. განსხვავებულ სფეროთა აპლიკაციები განსხვავებულად ტესტირდება. ამრიგად ტესტირება ხდება მხოლოდ ამ სფეროს და მისი პროგრამის ***კონტექსტის*** საფუძველზე. მაგალითად, საბანკო აპლიკაციის ტესტირება განსხვავებულია ნებისმიერი ელექტრონული კომერციის ან სარეკლამო აპლიკაციის ტესტირებისგან. თითოეული ტიპის აპლიკაციასთან დაკავშირებული რისკი განსხვავებულია, ამიტომ ეფექტური არ არის ერთიდაიმავე მეთოდის, ტექნიკის და ტესტირების ტიპის გამოყენება ყველა ტიპის აპლიკაციის შესამოწმებლად.

პროგრამული უზრუნველყოფის ტესტირება აუცილებელი ნაბიჯია პროგრამულ დეველოპმენტის სასიცოცხლო ციკლში, რადგან იგი ამოწმებს, მუშაობს თუ არა პროგრამა წარმატებით საბოლოო მომხმარებლისთვის. ტესტირების ეფექტურად და საიმედოდ შესრულების მიზნით, ყველამ უნდა გაიცნობიეროს პროგრამული ტესტირების ეს შვიდი პრინციპი, რომელიც ცნობილია, როგორც პროგრამული აპლიკაციების ტესტირების საყრდენი.

საერთოდ, ტერმინი პრინციპი ნიშნავს წესებს ან კანონებს, რომელთა დაცვაც საჭიროა. აქედან გამომდინარე, პროგრამული უზრუნველყოფის ტესტირების ინდუსტრიაში ყველამ უნდა დაიცვას ზემოაღწერილი შვიდი პრინციპი.

1.8. პროგრამის ბაგი, ტესტ-ქეისები და ტესტ-სუიტები

პროგრამული შეცდომა (software bug – ბაგი) არის პრობლემა, რომელიც იწვევს პროგრამის ავარიულ შეწყვეტას ან არასწორი შედეგების გამოტანას. იგი გამოწვეულია პროგრამის კოდის აგებისას არასაკმარისი ან მცდარი ლოგიკის საფუძველზე [46]. პროგრამული ბაგი შეიძლება იყოს შეცდომა, ხარვეზი, დეფექტი ან გაუმართაობა, რასაც მოჰყვება კოდის წარუმატებელი დამთავრება ან გადახრა მოსალოდნელი შედეგებისგან. ბაგების უმეტესობა გამოწვეულია კოდის ან მის დიზაინში ადამიანის მიერ დაშვებული შეცდომების გამო. პროგრამა ამბობს, რომ არის buggy, როდესაც ის შეიცავს ბევრ შეცდომას, რომლებიც გავლენას ახდენს პროგრამის ფუნქციონირებაზე და იწვევს არასწორ შედეგს.

შეცდომის გასწორება (Bug Fix) არის ცვლილება სისტემაში ან პროდუქტში, რომლის დანიშნულებაა პროგრამული შეცდომის / მტყუნების (მწყობრიდან გამოსვლის) დამუშავება. პროგრამული ბაგების მრავალი განსხვავებული ტიპი ქმნის სისტემის დანერგვის შეცდომებს, რაც აუცილებლად მოითხოვს ამ შეცდომების გასწორებას. ეს საკითხი წარმატებით წყდება დეველოპერების ან IT-სპეციალისტების სხვა გუნდის მიერ [47].

ბაგების გასწორება ასევე ცნობილია როგორც პროგრამის დროებითი ფიქსაცია (PTF - program temporary fix).

ამგვარად, ბაგების უმრავლესობა წარმოიქმნება შეცდომებისგან, რომლებიც დაშვებულია ადამიანის მიერ შექმნილ პროგრამულ კოდში ან დიზაინში. აპლიკაცია,

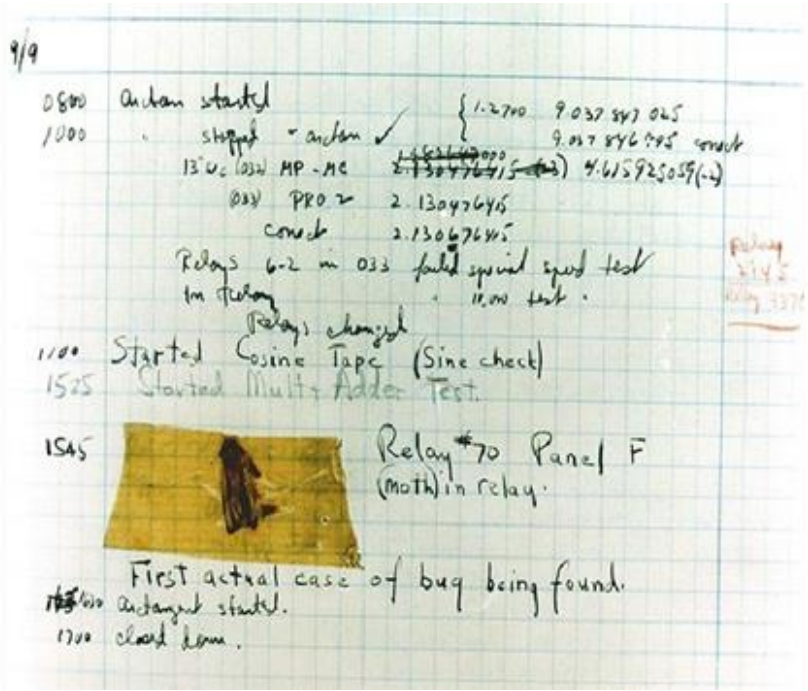
რომელიც შეიცავს ბევრ ბაგს, რომლებიც ხელს უშლის მის ფუნქციონალობას, ეწოდება „ბაგებიანი“ პროგრამა [38].

პროგრამის ფუნქციონირებისას ზოგიერთ ბაგს აქვს „უჩინარი" ეფექტი და შეიძლება იარსებოს დიდი ხნის მანძილზე. ბაგმა შეიძლება გამოიწვიოს პროგრამის გათიშვა, გაჩერება, „დაკიდება“ და სხვა.

ბაგის მოქმედების შედეგი შეიძლება ძალზედ სერიოზული იყოს. მაგალითად, რაკეტა Ariane-5 განადგურდა გაშვებიდან 1 წუთზე ნაკლებ დროში, რომლის მიზეზი იყო მისი კომპიუტერული უზრუნველყოფის „ბაგი“. 1994 წელს, სამხედრო საჰაერო ვერტმფრენი Chinook დაეჯახა მწვერვალს, რომელიც გამოწვეული იყო პროგრამული უზრუნველყოფის ბაგიდან, რითაც იმართებოდა ვერტმფრენის ძრავა [48].

კომპიუტერულ შეცდომას „ბაგი“ უწოდა ვინმე Grace Hopper–მა, რომელიც იყო კომპიუტერის დამწყები სპეციალისტი. 1946 წელს Grace Hopper–ი მუშაობდა გამოთვლით ლაბორატორიაში Mark-II კომპიუტერზე. მანქანაზე მომუშავე ოპერატორებმა მიაკვლიეს შეცდომის მიზეზს. როგორც გაირკვა ეს შეცდომა გამოწვეული იყო სქემაზე „დასახლებული" ხოჭოსგან. ეს ხოჭო ფრთხილად აიყვანეს და დასვეს ჩანაწერების ჟურნალზე (ნახ.1.8).

ასე წარმოიქმნა სიტყვა „ბაგი“, რომელიც ინგლისურად ხოჭოს ნიშნავს. Hopper არ იყო ის, ვინც ხოჭო იპოვნა. ერთ–ერთი ოპერატორი რომელმაც იგი იპოვნა იყო William Bruke, რომელმაც გასართობად შეინახა მწერი წარწერით: "პირველი ნამდვილი შემთხვევა არსებული ბაგის პოვნისა".



ნახ.1.8. ბაგის წარმოშობის ისტორია

Hopper–ს უყვარდა ამ ამბის მოყოლა და სწორედ მისი წყალობით შემორჩა ეს ამბავი ისტორიას. აღნიშნული ჩანაწერების ჟურნალი თავის მწერით ინახება ამერიკის Smithsonian-ის ნაციონალურ მუზეუმში [48].

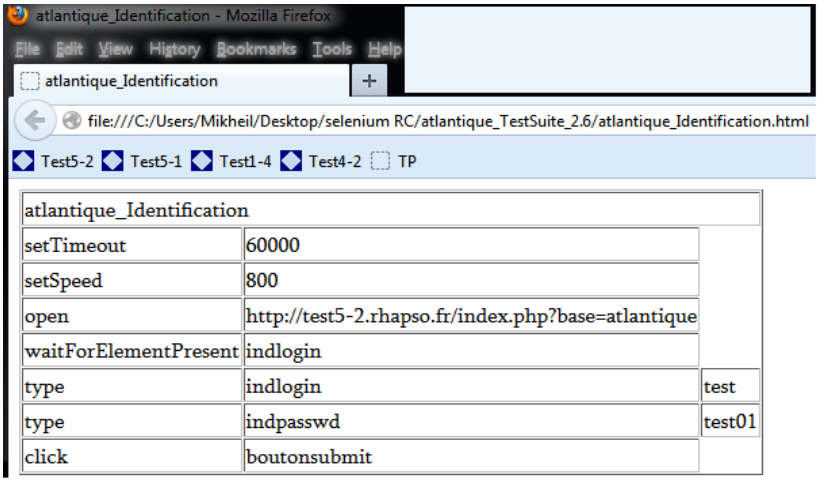
➢ ტესტ-ქეისები და ტესტ-სუიტები

ტესტ-ქეისი არის იმ ცვლადების, პირობების, მოქმედებების ერთობლიობა, რომელთა საშუალებით მოწმდება პროგრამული უზრუნველყოფის მუშაობის სისწორე. ტესტ-ქეისების საშუალებით ადგენენ სისტემის მუშაობის ხარისხს.

დაწერილ, შექმნილ ტესტ-ქეისს ეწოდება *ტესტ-სკრიპტი*. იგი ხელოვნურად ან ავტომატურად ჩაწერილი სკრიპტია, რომელიც ახდენს შესაბამის ტესტ-ქეისში მოცემული ინსტრუქციების შესრულებას.

პროგრამული უზრუნველყოფის ტესტირებისას შესაძლებელია დაიწეროს ხელოვნური ტესტ-სკრიპტი, რომელიც შესრულდება ადამიანის მიერ, ან შესაძლებელია გაკეთდეს ტესტ-ქეისში მოცემული ინსტრუქციების ავტომატიზაცია.

1.9 ნახაზზე წარმოდგენილია Selenium ტესტი, რომელიც ავტომატური ტესტ–სკრიპტია [38].



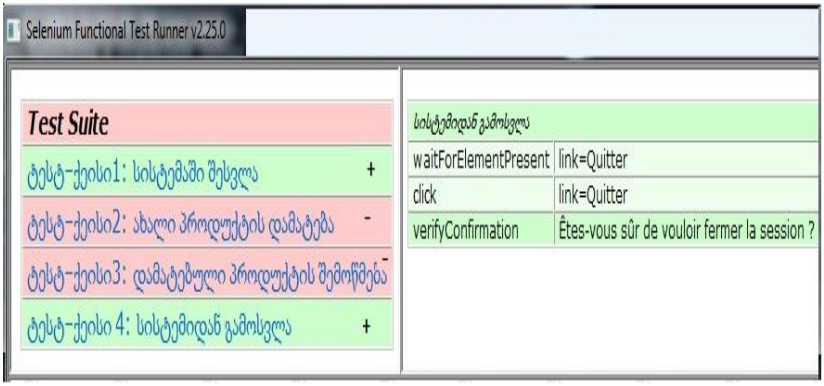
ნახ.1.9. ტესტ–სკრიპტი

ტესტ-სუიტი არის ტესტ-ქეისების ერთობლიობა. სისტემის ერთი მთლიანი ფუნქციის ტესტირების მიზნით საჭიროა ტესტ-ქეისების გაერთიანება სხვადასხვა ჯგუფებად, ტესტ-სუიტებად [38].

განვიხილოთ ტესტ-სუიტის მაგალითი, რომელიც შედგება 4 ტესტ-სკრიპტისგან:

- ტესტ-ქეისი 1: სისტემაში შესვლა;
- ტესტ-ქეისი 2: ახალი პროდუქტის დამატება;
- ტესტ-ქეისი 3: დამატებული პროდუქტის შემოწმება;
- ტესტ-ქეისი 4: სისტემიდან გამოსვლა.

განხილულ ტესტ-სუიტში მნიშვნელოვანია ტესტ-ქეისების მიმდევრობა, თითოეული ტესტის წარმატებით დასრულება. მაგალითად, წარმოვიდგინოთ, რომ ტესტ-სუიტის შესრულებისას არ შესრულდა ახალი პროდუქტის დამატება. ამ შემთხვევაში მომდევნო ტესტი – „დამატებული პროდუქტის შემოწმება" „ჩავარდება“, რაც ლოგიკურია – თუ ვერ შეიქმნა პროდუქტი, შესაბამისად ვერ მოხდება მისი შემოწმებაც. ამ შემთხვევაში ამბობენ, რომ სუიტში შემავალი ტესტ-ქეისები დამოკიდებულია ერთმანეთზე, ანუ შემდგომი ტესტ-სკრიპტის შესრულება დამოკიდებულია წინა ტესტ-სკრიპტის შესრულების შედეგზე (ნახ.1.10).



ნახ.1.10. ტესტ-სუიტი

მწვანე ფერით (მონიშნულია „+“) მოცემულია წარმატებით შესრულებული ტესტ–სკრიპტები, ხოლო წითელ ფერში წარმოდგენილია „ჩავარდნილი“ ტესტები (მონიშნულია „-“).

მოცემული სურათიდან შეგვიძლია დავასკვნათ, რომ ჩავარდა „ახალი პროდუქტის დამატების“ ტესტი, შესაბამისად ჩავარდა მასზე დამოკიდებული მომდევნო ტესტი – „დამატებული პროდუქტის შემოწმება“, ხოლო წარმატებით შესრულებულა სისტემაში „შესვლა – გამოსვლის“ ტესტები [38].

1.9. პროგრამული აპლიკაციების ტესტირების ტიპები და მათი კლასიფიკაცია

ორგანიზაციული მართვის (ანუ კორპორაციული მენეჯმენტის) სისტემის მხარდამჭერი პროგრამული უზრუნველყოფა (მოკლედ, კორპორაციული პროგრამული აპლიკაცია) არის მართვის ავტომატიზებული სისტემების ერთ-ერთი მნიშვნელოვანი კლასი, რომელიც მოიცავს პროდუქციის (ან მომსახურების) მარკეტინგული კვლევის, სტრატეგიული და ოპერატიული დაგეგმვის, ტექნოლოგიური პროცესების ავტომატიზაციის, ელექტრონული დოკუმენტბუნვის, პროდუქციის წარმოების, აღრიცხვის, გასაღების, ეკონომიკური ანალიზის, ფინანსური და საბუღალტრო აღრიცხვის პროცესების ავტომატიზაციის, კადრების მენეჯმენეტის და სხვ. ამოცანების კომპლექსური პროგრამების შექმნას [49-51]. იგი, თანამედროვე ტერმინოლოგიით, ERP/CRM სისტემებს მიეკუთვნება, რომელთა მიზანი და ორიენტაცია კორპორაციის ბიზნესის ხელშეწყობას ემსახურება [3,52].

განსაკუთრებით მნიშვნელოვანია კორპორაციული პროგრამული აპლიკაციების ხარისხის მართვისა და ინფორმაციული უსაფრთხოების დონის სრულყოფა; Desktop აპლიკაციები (სამაგიდო ანუ Windows დანართები), რომლებიც ადამიანის მიერ გამოიყენება ისეთ მოთხოვნილებათა დასაკმაყოფილებლად, როგორიცაა: გამოთვლები, თამაშები, ტრეინინგები, გართობა და სხვა.

არსებობს სხვადასხვა პლატფორმა (ე.წ. ოპერაციული სისტემები), რომლებიც კომპიუტერულ მოწყობილობათა რესურსების გამოყენებით უზრუნველყოფს პროგრამული აპლიკაციების მართვას, სისტემების ინტეგრაციას [36,53].

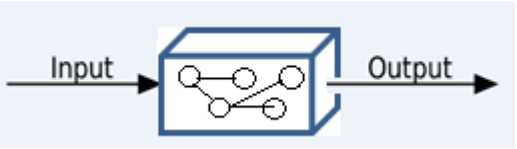
სხვადასხვა ტიპის კომპიუტერულ აპლიკაციას აქვს განსხვავებული არქიტექტურა. შესაბამისად განსხვავდება მათი შექმნის პროცესიც. მაშასადამე, იმისათვის, რომ ვიზრუნოთ განსხვავებული ტიპის პროგრამული უზრუნველყოფის შექმნაზე, ამისათვის არსებობს მრავალი ტიპის პროგრამული ტესტირება [55,56].

განვიხილოთ პროგრამული უზრუნველყოფის ტესტირების ძირითადი ტიპები:

- *თეთრი ყუთის ტესტირება (White box testing)* – კომპიუტერული პროგრამის ტესტირების მეთოდი, რომლის დროსაც ტესტირების პროცესი მიმდინარეობს პროგრამული უზრუნველყოფის შიგა სტრუქტურის დონეზე.

„თეთრი ყუთის“ ტესტირება მოითხოვს პროგრამის შიგა სტრუქტურის, პროგრამული კოდის ცოდნას. ძირითადად, „თეთრი ყუთის" ტესტირებას ახორციელებენ დეველოპერ-პროგრამისტები, რადგან, როგორც აღვნიშნეთ, მის დროს

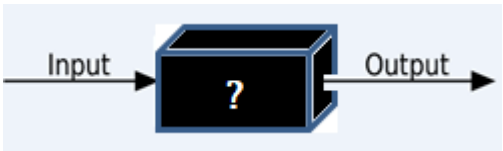
აპლიკაციის შემოწმება ხდება პროგრამული კოდის დონეზე (ნახ.1.11).



ნახ.1.11. „თეთრი ყუთი“-ს ტესტირება

მსგავსი ტიპის ტესტირება დაფუძნებულია პროგრმულ კოდის ცოდნაზე, კომპონენტებსა და მათ შორის კავშირებზე, ციკლებზე და ა.შ. [40].

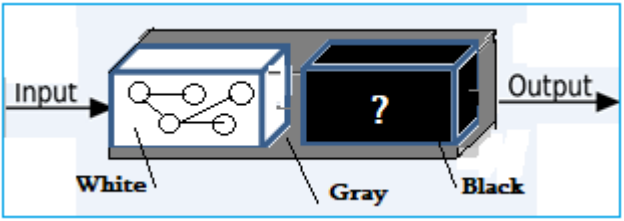
• *შავი ყუთის ტესტირება (Black box testing)* – ტესტირების მეთოდია, რომლის დროსაც ხდება აპლიკაციის ფუნქციონალის ტესტირება შიგა სტრუქტურის გამოკვლევის გარეშე. ამ დროს პროგრამული კოდის ან არქიტექტურის ცოდნა არაა საჭირო. „შავი ყუთის" ტიპის ტესტირებისას ტესტერი ამოწმებს თუ *რა გააკეთა პროგრამამ* და არ აინტერესებს თუ როგორ მიიღო შედეგი. მაგალითად, კომპიუტერულ პროგრამას შესასვლელზე ვაძლევთ რაღაც მონაცემებს, ანუ ვაწვდით Input-ს, ხდება ამ მონაცემების დამუშავება და შედეგად ვიღებთ Output-ს (ნახ.1.12). აპლიკაციის შიგა პროცესები უგულებელყოფილია.



ნახ.1.12. „შავი ყუთი“-ს ტესტირება

საბოლოოდ, შეგვიძლია დავასკვნათ, რომ „შავი ყუთის" ტესტირება არის ტესტირების ტიპი, რომლის დროსაც ხდება პროგრამის ფუნქციონალის შემოწმება შიგა სტრუქტურის გამოკვლევის გარეშე.

- *რუხი ფერის ყუთის ტესტირება (Gray box testing)* – ტესტირების მეთოდია, როდესაც გამოიყენება თეთრი და შავი ყუთის ტესტირების კომბინაცია (ნახ.1.13).



ნახ.1.13. „რუხი ყუთი"-ს ტესტირება

ამ ტესტირების მიზანია დეფექტების პოვნა, თუ იგი გამოწვეულია პროგრამების არასწორი სტრუქტურის ან აპლიკაციის არასათანადო გამოყენების გამო. რუხი ყუთის ტესტირებას ძირითადად იყენებენ უსაფრთხოების და ფუნქციონალის შეფასებაში, ვებ გვერდებისა და ვებ სერვისების შემოწმებისას [57].

რუხი ყუთის ტესტირების შესრულების ბიჯებია:

– ბიჯი 1: შესატან მონაცემთა ერთობლიობის დადგენა;
– ბიჯი 2: შედეგების (გამომავალი მონაცემების) განსაზღვრა;
– ბიჯი 3: ძირითადი გზების (paths) იდენტიფიცირება;
– ბიჯი 4: ქვეფუნქციების განსაზღვრა;
– ბიჯი 5: ქვეფუნქციებისთვის შემავალი მონაცემების დადგენა;

– ბიჯი 6: ქვეფუნქციებისთვის შედეგების (გამომავალი მონაცემების) დადგენა;
– ბიჯი 7: ტესტ-ქეისის შესრულება ქვეფუნქციებისთვის;
– ბიჯი 8: ქვეფუნქციებისთვის სწორი შედეგების ვერიფიკაცია;
– ბიჯი 9: 4 და 8 ბიჯების გამეორება სხვა ქვეფუნქციებისთვის;
– ბიჯი 10: 7 და 8 ბიჯების გამეორება სხვა ქვეფუნქციებისთვის.

ამგვარად, თეთრ, შავ და რუხი ყუთებს შორის არსებობს თვისობრივი განსხვავება, რომელიც ასე შეიძლება ჩამოყალიბდეს:

- *თეთრი ყუთის ტესტირება* – სტრუქტურული ტესტირებაა, კოდის ცოდნაზე დაფუძნებული, გამჭვირვალე;

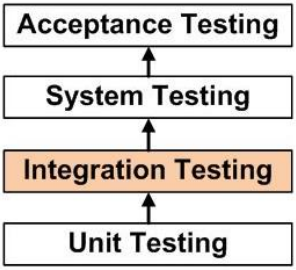
- *შავი ყუთის ტესტირება* – ფუნქციური ტესტირებაა, მონაცემთა საფუძველზე დაფუძნებული, გაუმჭვირვალე;

- *რუხი ყუთის ტესტირება* – ნახევრად-გამჭვირვალე, ნახევრად-სტრუქტურული ტესტირება, რადგან ტესტერს აქვს შეზღუდული ცოდნა კოდირების შესახებ.

- *მოდულური ტესტირება (Unit testing)* – მცირე ზომის ტესტირების მეთოდია, რომლის დროსაც მიმდინარეობს მარტივი მოდულების ან ფუნქციების ტესტირება [59]. აღნიშნული ტიპის ტესტირებას ასრულებს პროგრამისტი და არა ტესტერი, რადგან იგი მოითხოვს პროგრამის შიგა დიზაინის, კოდის დეტალურ ცოდნას. ზოგადად, მსგავსი ტიპის ტესტირებას მიმართავენ მაშინ, როდესაც აპლიკაციის არქიტექტურის დიზაინი დასრულებულია;

- *ინტეგრაციული ტესტირება (Integration Testing)* - პროგრამული უზრუნველყოფის ტესტირების დონეა, სადაც ინდივიდუალური მოდულები გაერთიანებულია და

ტესტირდება ჯგუფურად (ნახ.1.14) [59]. ამ დონის ტესტირების მიზანია შეცდომების გამოვლენა ინტეგრირებულ მოდულებს შორის ურთიერთქმედებაში. ინტეგრაციის ტესტირების დასახმარებლად იყენებენ ტესტის დრაივერებსა (test drivers) და ტესტის სახშობებს (test stubs).



ნახ.1.15. ტესტ-დონეები

სახშობი გამოიყენება ინტეგრაციის ტესტირების დროს დაღმავალი ტექნოლოგიით (Top-down), ქვედა დონის მოდულების ქცევის მოდელირების მიზნით, რომელიც ჯერ კიდევ არ არის ინტეგრირებული. სახშობი არის მოდული, რომელიც დროებით ანაცვლებს ჯერ არარსებულ მოდულს და იძლევა იმავე შედეგს, რასაც რეალური მოდული.

- *ზრდადი ტესტირება (Incremental testing)* – ტესტირების მეთოდი, რომლის დროსაც პირველ ეტაპზე მიმდინარეობს ქვესისტემების დამოუკიდებლელი ტესტირება, შემდგომ ეტაპზე ტესტირების პროცესი გადადის გაერთიანებულ ქვესისტემებზე და გრძელდება მანამ, სანამ არ მივიღებთ მთლიან სისტემას. ზრდადი ტესტირების მეთოდიკა არის ინტეგრირებული ტესტირების ნაირსახეობა.
- *ფუნქციონალური ტესტირება (Functional testing)* – „შავი ყუთის" ტიპის ტესტირებაა, რომლის დროსაც ტესტირდება პროგრამის ფუნქციები შესასვლელზე მიცემული მონაცემებით და გამოსასვლელზე მიღებული შედეგებით. მის დროს განიხილება პროგრამის შიგა სტრუქტურაც. ფუნქციური ტესტირება მოიცავს პროგრამის სამომხმარებლო

ინტერფეისის, მონაცემთა ბაზის და პროგრამული უსაფრთხოების ტესტირებას. ფუნქციური ტესტირება შეიძლება განხორციელდეს ხელოვნურად ან ავტომატურად [42].

- *სისტემური ტესტირება (System testing)* – არის დასრულებული, მთლიანი კომპიუტერული პროდუქტის ტესტირების მეთოდი. ზოგადად, სისტემური ტესტირება მოიცავს სხვადასხვა ტესტების სერიას, რომელთაც აქვს ერთადერთი მიზანი: დაატესტიროს კომპიუტერზე ბაზირებული მთლიანი პროგრამული სისტემა.
- *რეგრესიული ტესტირება (Regression testing)* – მისი მიზანია დაადასტუროს, რომ ახალმა პროგრამულმა ცვლილებამ არ გამოიწვია უკვე არსებული და შემოწმებული ფუნქციურობის „გაფუჭება". „Regression" ტესტირების დროს მიმდინარეობს უკვე გაშვებული ტესტების ხელახალი გაშვება. აღნიშნული ტესტირების ტიპი უზრუნველყოფს, რომ ახალ პროგრამულ ცვლილებას არ აქვს გვერდითი მოვლენები არსებულ ფუნქციაზე. „Regression" ტესტების გაშვების აუცილებლობა დგება მაშინ, როდესაც ხდება ქვემოთ მითითებული ერთ-ერთი მოვლენა:
    - იცვლება მომხმარებლის მოთხოვნები, მაშასადამე იცვლება კოდი მოთხოვნების შესაბამისად;
    - პროგრამას ემატება ახალი მახასიათებელი;
    - ფიქსირდება პროგრამული დეფექტი.

მსგავსი ტიპის ტესტირებისას გამოყენებადია ავტომატური მატესტირებელი ინსტრუმენტები, რადგან ყოველი პროგრამული ცვლილებისას მიმდინარეობს ერთიდაიმავე ტესტების მრავალჯერადი გაშვება [40].

- *თანხმობის ტესტირება (Acceptance testing)* – კლიენტის მიერ შესრულებული ტესტირება, რომლის დროსაც ხდება დასრულებული სისტემის შემოწმება, ეთანადება თუ არა სისტემა დამკვეთის (მყიდველის) მოთხოვნებს. იგი არის ტესტირების ერთ-ერთი ბოლო ფაზა მანამ, სანამ მოხდება აპლიკაციის კომერციულ წარმოებაში გაშვება [42].
- *დატვირთვით ტესტირება (Load testing)* – ესაა აპლიკაციის ტესტირება რთულად დასამუშავებელი მონაცემებით, მაგალითად Web-საიტის დატვირთვით ტესტირება. მისი მიზანია განისაზღვროს თუ რა წერტილში ქვეითდება, ნელდება ან „ეკიდება“ სისტემის მუშაობა.
- *სტრესული ტესტირება (Stress testing)* – ტესტირების მეთოდია, რომლის დროსაც ხორციელდება სისტემის „სტრესული" რეჟიმით მუშაობა, რათა შემოწმდეს როდის ან როგორ „ჩავარდება“ იგი. პროგრამის საწყის მონაცემებად განიხილება დიდი რიცხვები, ან მონაცემთა ბაზის რთული მოთხოვნების გაშვება და ა.შ.
- *გამოყენებადი ტესტირება (Usability testing)* – საბოლოო მომხმარებლის მიერ ყველაზე ხშირად გამოყენებადი ფუნქციების ტესტირება. მისი მიზანია გამოყენებად ფუნქციებში გამოააშკარაოს პროგრამული დეფექტები [42].
- *უსაფრთხოების ტესტირება (Security testing)* – არის ტესტირების მეთოდი, რომლის დროსაც მოწმდება თუ როგორ არის დაცული სისტემა არასანქცირებული გარე თუ შიგა წვდომისგან, წინასწარგანზრახული დაზიანებისგან და ა.შ. მის დროს ხორციელდება სისტემის ყველა შესაძლო სუსტი წერტილის ტესტირება, რათა არასანქცირებული შეტევისას არ მოხდეს ინფორმაციის წაშლა, ინფორმაციის მოპარვა, პროგრამის მწყობრიდან გამოყვანა და ა.შ.

II თავი

პროგრამული აპლიკაციების ხარისხის მართვის, დიზაინის და უსაფრთხოების კონცეფციები

წინამდებარე თაცში განხილულია კორპორაციული მენეჯმენტის სისტემების მხარდამჭერი პროგრამული აპლიკაციების აგების საკითხები სტრუქტურული და ობიექტ-ორიენტირებული მეთოდების საფუძველზე. წარმოდგენილია ორგანიზაციული მართვის პროგრამული სისტემების ხარისხის შეფასების, აგრეთვე ინფორმაციული უსაფრთხოების, პროგრამების ოპტიმიზაციის, მომხმარებელთა ინტერფეისების (UI/UX) და მათი ქცევის პროცესების დიზაინის რეალიზაცის მეთოდების გამოყენების კონცეფციები. განხილულია ვერსიების კონტროლის სისტემა კონტენტ-მენეჯმენტის მაგალითზე, ვებ-ჰოსტინგის სერვისების ტიპები და ვირტუალური კერძო ქსელები ინფორმაციული უსაფრთხოების თვალსაზრისით.

2.1. პროგრამული აპლიკაციების აგება ხარისხის მართვის კონცეფციით

ორგანიზაციული მართვის (ანუ კორპორაციული მენეჯმენტის) სისტემის მხარდამჭერი პროგრამული უზრუნველყოფა (მოკლედ, კორპორაციული პროგრამული აპლიკაცია) არის მართვის ავტომატიზებული სისტემების ერთ-ერთი მნიშვნელოვანი კლასი [49]. რომელიც მოიცავს პროდუქციის (ან მომსახურების) მარკეტინგული კვლევის, სტრატეგიული და ოპერატიული დაგეგმვის, ტექნოლოგიური პროცესების ავტომატიზაციის, ელექტრონული

დოკუმენტბუნვის, პროდუქციის წარმოების, აღრიცხვის, გასაღების, ეკონომიკური ანალიზის, ფინანსური და საბუღალტრო აღრიცხვის პროცესების ავტომატიზაციის, კადრების მართვის და სხვ. ამოცანების კომპლექსური პროგრამების შექმნას [50,51]. იგი, თანამედროვე ტერმინოლოგიით, ERP/CRM სისტემებს მიეკუთვნება, რომელთა მიზანი და ორიენტაცია კორპორაციის ბიზნესის ხელშეწყობას ემსახურება [52].

განსაკუთრებით მნიშვნელოვანია კორპორაციული პროგრამული აპლიკაციების ხარისხის მართვისა და ინფორმაციული უსაფრთხოების დონის სრულყოფა.

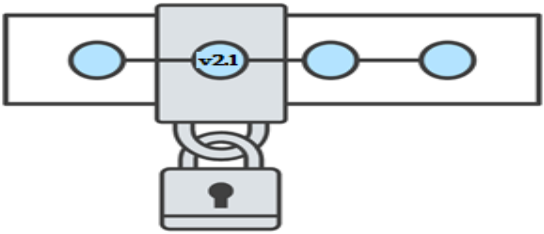
წინამდებარე თავში გადმოცემულია აღნიშნული საკითხების მოკლე მიმოხილვა და დასაბუთებული რეკომენდაციები ბიზნეს-მომხმარებლებისა და IT-სპეციალისტებისთვის, გამოყენებითი პროგრამული აპლიკაციების ინტერფეისების დიზაინის, უსაფრთხო მონაცემთა საცავებისა და ინტერნეტ/ინტრანეტ ქსელების აგების და ექსპლუატაციის ამოცანების გადასაწყვეტად.

რგორც ცნობილია, გამოყენებითი პროგრამული აპლიკაცია (Applied Software Application), აგებული სტრუქტურული და ობიექტ-ორიენტირებული მეთოდების საფუძველზე, შედგება იერარქიულად განაწილებული პროგრამული მოდულებისაგან (კოდებისაგან), რომელთა დანიშნულება გარკვეული ფუნქციური ამოცანების გადაწყვეტაა კომპიუტერული ტექნიკის საშუალებით [10].

თვით პროგრამული კოდი კი, ზოგადად, შედგება ცვლადების, ფუნქციების, კლასების, მეთოდებისა და

ლოგიკური ოპერატორებისგან. სხვადასხვა დეველოპერი ერთიდაიმავე ფუნქციურ პროგრამას სხვადასხვა სახით ქმნის, შედეგი, ხშირად, ერთიდაიგივეა. ასეთ შემთხვევაში გასათვალისწინებელია *პროგრამის ხარისხი,* თუ როგორაა კოდი აგებული და როგორ მიიღება შედეგი. პროგრამის (აპლიკაციის) ხარისხის შესაფასებლად არსებობს სპეციალური მეტრიკები (Software Measurement and Metrics), რომლებიც ჩვენ შედარებით დეტალურად პირველ თავში წარმოვადგინეთ. ამჯერად მათ მოკლედ განვიხილავთ თანმხლებ ძირითად ცნებებთან და გამოყენების ასპექტებთან ერთად [1,49].

- *ვერსიების კონტროლის სისტემა (Version Control System -VCS)* – არის პროგრამული ინსტრუმენტი, რომელიც პროექტებზე მომუშავე დეველოპერების გუნდს აძლევს საშუალებას დროთა განმავლობაში შეცვალონ კოდი (ნახ.2.1).



ნახ.2.1. ვერსიების კონტროლის (VC) სისტემის ლოგო

განვმარტოთ უფრო დეტალურად, თუ რას წარმოადგენს Version control და შემდეგ დავუბრუნდეთ მეტრიკებს. VC ახდენს ყველა იმ მოდიფიკაციის შენახვას, რომელიც განხორციელდა კოდში (ინახავს სპეციალურ მონაცემთა

ბაზაში). თუ მოხდება შეცდომა და კოდი აღარ იმუშავებს, შესაძლებელია წინა ვერსიაზე დაბრუნება [60]. არსებობს პროგრამის მოდიფიცირების გადახედვის შესაძლებლობა, თუ რომელმა დეველოპერმა რა ცვლილება შეიტანა. ყოველ ცვლილებაზე დეველოპერის მიერ ხდება კონკრეტული სახელის დარქმევა.

კონკრეტული აპლიკაციის ფაილების განთავსება ხდება აღნიშნული სისტემის ერთ კონკრეტულ ადგილას, რომელთან წვდომა, ჩამოტვირთვა და მუშაობა რამდენიმე დეველოპერს შეუძლია ერთდროულად კოდის სხვადასხვა ნაწილზე. მაგალითად, ერთმა მოახდინოს ახალი ფუნქციის ჩაშენება, მეორემ გაასწოროს შეცდომა, მიღებული შედეგები კი საბოლოოდ იმ ერთ კონკრეტულ მისამართზე აიტვირთოს, რომელიც საბოლოო პროდუქტს იძლევა. ვერსიების კონტროლის პროგრამა მუშაობს დამოუკიდებელი აპლიკაციის სახით, მაგრამ ხშირად იგი ჩაშენებულია სხვადასხვა ტიპის პროგრამულ უზრუნველყოფაში. მაგალითად, ტექსტურ რედაქტორებში, ელ-ცხრილებსა და კონტენტ-მენეჯმენტის სისტემებში (ნახ.2.2).

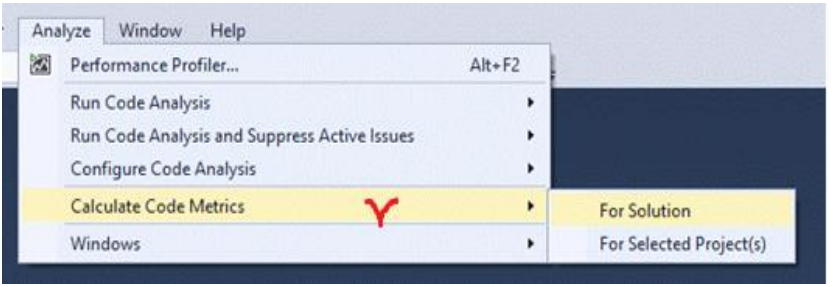
შინაარსის (კონტენტის) მართვის სისტემა (CMS) არის პროგრამული აპლიკაცია, რომლის საშუალებითაც შეიძლება ციფრული შინაარსის შექმნა და მოდიფიცირება. CMS ჩვეულებრივ გამოიყენება კორპორაციული (საწარმოო) შინაარსის მართვისა და ვებ-შინაარსის მენეჯმენტისათვის. მაგალითად, ცნობილია ასეთი კონტენტ-მენეჯმენტის სისტემები: HubSpot Website Platform, WordPress, Sitefinity, Kentico, Mura, Joomla!, Crownpeak, idev CMS, Oracle WebCenter

Content, Evoq Content [61]. შეიძლება Ms SharePoint-იც განხილულ იქნას როგორც კორპორაციული CMS {62}.

ნახ.2.2. კონტენტ-მენეჯმენტის (CMS) სისტემის ლოგო

➢ *კოდის მეტრიკები (Code Metrics ან Software Metrics)* – არის საზომი, რომლის საშუალებითაც შეიძლება პროგრამული უზრუნველყოფის ზოგიერთი მახასიათებლის ან მისი სპეციფიკაციის რაოდენობრივი მნიშვნელობის მიღება.

Code Metrics Tool – ინსტრუმენტია, რომელიც აფასებს, თუ რომელი ნაწილია გაკეთებული კოდში ცუდად, მაგალითად: კოდში დიდი მეთოდებია; კოდი მეორდება; კლასები დიდია ან კოდში „მკვდარი კოდია“ (იგი არაფერს ასრულებს). კოდის შეფასების მიზნით საჭიროა პროგრამაში Calculade Code Metrics-ის ამოქმედება (ნახ.2.3) [63].



ნახ.2.3. კოდის შეფასების პროგრამა

- *ვებ-ჰოსტინგის სერვისი (Web-Hosting)* – არის ინტერნეტ ჰოსტინგის სერვისის ტიპი. იგი საშუალებას იძლევა პიროვნებამ ან კორპორაციამ თავისი ვე-ბსაიტი ხელმისაწვდომი გახადოს მსოფლიო ქსელში (WWW). თვით ვებ-ჰოსტები კი „მასპინძელი" კომპანიებია, რომლებიც უზრუნველყოფს კლიენტებისათვის ადგილს სერვერზე. ისინი ფლობს საკუთარ (ან იჯარის საფუძველზე) სერვერულ სივრცეებს და შესაბამის ინტერნეტ სერვისებს [64].

მეტრიკების ანალიზის საფუძველზე სწორად დაწერილი ვებ-აპლიკაციის კოდი და მისი შედეგები ფასეული რომ გახდეს, საჭიროა იგი ქსელში განთავსდეს, შესაძლებელი იყოს მასთან წვდომა სხვა კომპიუტერებიდანაც. წინააღმდეგ შემთხვევაში ეს აპლიკაცია უსარგებლოდ დარჩება.

ვებ-აპლიკაციის განსათავსებლად ინტერნეტ სივრცეში (მაგალითად, გლობალურ ქსელში), საჭიროა გარკვეული სივრცის ყიდვა. ვებ-ჰოსტზე განთავსებულია FTP File upload, მონაცემთა ბაზები, ელ-ფოსტის მომხმარებლები (ნახ.2.4). ერთ სერვერზე თავსდება რამდენიმე საიტი, რომელთა გარჩევაც IP მისამართებით ხდება.

მაგალთად, google ip მისამართია: 172.217.22.14 (ან მისი შესაბამისი google.com). აღნიშნული საკითხის შესრულებას ახდენს DNS-სერვერი.

ნახ.2.4. Web-Hosting

➢ *დომენების სახელთა სისტემა (Domain Name System - DNS)* – არის კომპიუტერული განაწილებული სისტემა, რომლის მთავარი ფუნქცია ინტერნეტ დომენების შესახებ ინფორმაციის მოწოდებაა. მაგალითად, დომენის სახელებია: google.com, wikipedia.org, youtube.com და სხვ. ყველა ვებ-ბრაუზერი დაფუძნებულია DNS პრინციპზე, რათა მოხდეს სწრაფი წვდომა მომხმარებელთა მოთხოვნილ ინფორმაციაზე [65,66].

➢ *HTTPS vs HTTP (Hyper Text Transfer Protocol)* – კომუნიკაციის არხია კლიენტსა და სერვერს შორის მოთხოვნა-პასუხების *(Request/Response)* შესასრულებლად). ყველა ბრაუზერს აქვს ჩაშენებული XMLHttpRequest (XHR). ესაა Javascript ობიექტი, რომელიც გამოიყენება მონაცემების

მიმოცვლისთვის ვებ-ბრაუზერსა და ვებ-სერვერს შორის. XHR მონაცემები სხვადსხვა ტიპისაა, როგორიცაა HTML, CSS, XML, JSON აგრეთვე ჩვეულებრივი ტექსტის სახით. XHR-ით შესაძლებელია: გვერდის განახლება გადატვირთვის გარეშე; მონაცემების მოთხოვნა (ან მიღება) სერვერისგან, მას შემდეგ რაც გვერდი უკვე ჩატვირთულია; მონაცემების გაგზავნა სერვერზე - დაფარულად. XHR ობიექტი არის AJAX და JSON-ის ძირითადი კონცეფცია [67].

გარდა http არხისა, არსებობს https არხიც. ორივე URL-ის შემადგენელი ნაწილია, რომელიც ბრაუზერის მისამართის ადგილს შეესაბამება.

ინფორმაციული უსაფრთხოების თვალსაზრისით HTTP და HTTPS არხებს შორის მნიშვნელოვანი განსხვავებაა. კერძოდ, როდესაც იყენებენ HTTP მოთხოვნას მოთხოვნას, მაშინ ამ დროს მომხმარებლის მიერ შეყვანილი პაროლი არაა დაშიფრული და თუ ჰაკერი მოახერხებს წვდომას მის მიერ შეყვანილ მონაცემთან, იგი ავტომატურად გაიგებს თუ რა პაროლი იქნა გადაცემული და მონაცემებიც მის ხელში აღმოჩნდება.

HTTPS იყენებს SSL-ს (Secure Sockets Layer). ესაა ქსელის პროტოკოლი მონაცემთა უსაფრთხო გადაცემისათვის [68]. უსაფრთხოების ამ ტექნოლოგიით ინფორმაცია სერვერსა და კლიენტს შორის იგზავნება *დაშიფრულად*. ავტორიზაციის გავლის შემდეგ საიტზე, გაგზავნილი პაროლი დაშიფრულად გადაიცემა, თუ მასთან წვდომას მოახდენს ჰაკერი, ეს ინფორმაცია მისთვის მაინც გაუგებარი აღმოჩნდება.

➢ ვირტუალური კერძო ქსელი (*Virtual Private Network - VPN)* – არის ტექნოლოგია, რომლის დანიშნულებაა ქსელში ინფორმაციის დაცულად გადაცემა. იგი იყენებს კრიპტო-გრაფიულ საშუალებებს: დაშიფვრა, აუთენტიფიკაცია, საჯარო საკვანძო ინფრასტრუქტურა, ლოგიკური ქსელის საშუალებით გადაცემულ შეტყობინებებში განმეორებებისგან და ცვლილებებისგან დაცვა [69]. VPN-ის გამოყენებისას ჰაკერებს არ შეუძლია კლიენტის საიდუმლო მონაცემების მიღება. VPN გვთავაზობს უსაფრთხო კავშირს საჯარო ქსელში (როგორიცაა Wi-fi ტრანსპორტში, სასტუმროში, კაფეში) ასევე სახლის ქსელში [70].

2.2. მომხმარებლის ინტერფეისი (UI/UX) პროგრამულ აპლიკაციაში

*მომხმარებლის ინტერფეისი / მომხმარებლის გამოცდილება (**User Interface - UI) / User Experience - UX)*** – მნიშვნელოვანი ცნებებია ინფორმაციული სისტემების დაპროექტებისა და აგების სფეროში [71].

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

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

2.5 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.2.5. 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.

2.3. პროგრამული აპლიკაციის ოპტიმიზაციის კონცეფცია

პროგრამული აპლიკაციის ოპტიმიზაცია – არის პროგრამული სისტემის ზუსტი აწყობა და მოდიფიკაცია მისი ეფექტურობის სრულყოფის მიზნით, კერძოდ, მისი მწარმოებლურობის ასამაღლებლად.

პროგრამული კოდის ოპტიმიზაცია ემყარება შემდეგ სამ პრინციპს:

- ბუნებრიობა;
- მწარმოებლურობა და
- დახარჯული დრო [72].

- *ბუნებრიობა* გულისხმობს ისეთ კოდს, რომელიც არის მოწესრიგებული, მოდულური და ადვილად წასაკითხი. თითოეული მოდული ბუნებრივად უნდა ჩაშენდეს მთლიან პროგრამაში. შესაძლებელი უნდა იყოს კოდის ცალკეული ფუნქციების მარტივად რედაქტირება, ინტეგრირება ან წაშლა;
- ოპტიმიზაციის შედეგად, მიღებულ უნდა იქნას პროგრამის ეფექტურობის (*მწარმოებლურობის*) ზრდა. როგორც წესი, კარგად ოპტიმიზებული პროგრამა ორიგინალ ვერსიასთან შედარებით ზრდის სწრაფქმედებას მინიმუმ 20-30% -ით;
- ოპტიმიზაცია და შემდგომი *გასწორება* უნდა მოხდეს მოკლე პერიოდის განმავლობაში. ოპტიმალური პირობებია ის, რაც არ უნდა აღემატებოდეს თვით პროგრამული პროდუქტის დაწერისას *დახარჯული დროის* 10–15%, წინააღმდეგ შემთხვევაში, ეს იქნება წამგებიანი.

კოდის ოპტიმიზაციის მიზნით მასში უნდა მოიძებნოს „ვიწრო ადგილი" (bottleneck – „ბოთლის ყელი“), რომელზეც არის დამოკიდებული სწრაფქმედება (მწარმოებლურობა).

ესაა კოდის კრიტიკული წერტილი, რომელიც უნდა გაუმჯობესდეს, რომ არ მოხდეს რესურსების დაკარგვა. ვიწრო ადგილების გამოსავლენად გამოიყენება სპეციალური ანალიზური პროგრამები – ***პროფაილერები (profilers)***. ესაა პროგრამული ინსტრუმენტი, რომელიც აანალიზებს პროგრამის მსვლელობის ქცევას [73]. იგი ეხმარება დეველოპერს პროგრამების დინამიკური ანალიზისა და შედარების გზით პრობლემური ადგილების აღმოჩენაში.

მაგალითად, იგი ზომავს მეხსიერების სივრცეს ან პროგრამის შესრულების დროით სირთულეს, ფუნქციების გამოძახების სიხშირეს და ხანგრძლივობას. შემდეგ კი შეიძლება მიღებულ იქნეს ზომები კოდის სტრუქტურული და ალგორითმული სრულყოფისათვის, რაც ხელს უწყობს *პროგრამის ოპტიმიზაციას.*

პროფაილერები იყენებს სხვადასხვა მეთოდებს, როგორიცაა მაგალითად, მოვლენებზე ბაზირებული, სტატისტიკური, ინსტრუმენტული და მოდელირების მეთოდები.

წიგნის მომდევნო თავში დეტალურად შევეხებით პროგრამული აპლიკაციების ოპტიმიზაციის საკითხებს.

ამგვარად, კორპორაციული მენეჯმენტის პროცესების ავტომატიზაცია მოითხოვს მძლავრი, მოქნილი, საიმედო და უსაფრთხო პროგრამულ-აპარატურული ქსელური ტექნოლოგიების გამოყენებას, რომლის პროგრამული და ინფორმაციული უზრუნველყოფა აგებულ იქნება თანამედროვე, მაღალი ხარისხის მხარდამჭერი პროგრამული აპლიკაციებისა და დიდ მონაცემთა საცავების საფუძველზე. გადაწყვეტილების მიღების ბიზნეს-პროცესების ხელშემწყობი პროგრამული სისტემების ხარისხის შეფასება, პროგრამების ოპტიმიზაცია, მომხმარებელთა ინტერფეისების და მათი ქცევის პროცესების დიზაინის (UI/UX) ჰარმონიული რეალიზაცია უსაფრთხო, ვირტუალური ქსელური რესურსების საფუძველზე, მნიშვნელოვნად შეუწყობს ხელს ორგანიზაციის ბიზნესის მდგრად განვითარებას.

III თავი
კომპიუტერული პროგრამების ოპტიმიზაციის პრინციპები და მეთოდები

კომპიუტერულ მეცნიერებაში (Computer Science) პროგრამის ან პროგრამული უზრუნველყოფის ოპტიმიზაცია არის პროგრამული სისტემის მოდიფიცირების პროცესი, რომ მისმა ცალკეულმა ნაწილმა და მთლიანადაც უფრო ეფექტურად (სწრაფად) იმუშაოს ან გამოიყენოს ნაკლები რესურსი (მაგალითად, მეხსიერება) [74].

ამგვარად, თუ ზოგადად ჩამოვაყალიბებთ, კომპიუტერული პროგრამა შეიძლება ისე ოპტიმიზირდეს, რომ ის უფრო სწრაფად შესრულდეს (დრო – T*opt*), ან გახადოს იგი ნაკლები მეხსიერების (შენახვის მოცულობა – V*opt*) ან სხვა რესურსების (მაგალითად, გამოყენების ნაკლები ენერგია – P*opt*) მომთხოვნი.

ხშირად ამ ძირითადი მაჩვენებლების მნიშვნელობათა კომპრომისული გადაწყვეტაა შესაძლებელი, ვინაიდან ისინი ერთმანეთის მიმართ ორთოგონალურია. კერძოდ, პროგრამის შესრულების დროის შემცირება ხშირად მეხსიერების ზრდით მიიღწევა და პირიქით. ეს პრობლემა სპეციალური კვლევის ობიექტია კონკრეტული საპრობლემო სფეროსა და მისი პროგრამული უზრუნველოფის სპეციფიკაციების შემუშავების პროცესში [75]. აქ განვიხილავთ პროგრამული სისტემების ოპტიმიზაციის კლასიკურ და ახალ მიდგომებს, მათ ძირითად პრინციპებს და ფუნქციონირების მწარმოებლურობის შეფასების რაოდენობრივ მეთოდებს.

3.1. ოპტიმიზაციის ძირითადი პრინციპები

კომპიუტერული პროგრამების ან პროგრამული უზრუნველყოფის ტესტირების პროცესი, ძირითადად სამ საკითხს ეხება, ესენია: ბუნებრიობა, მწარმოებლურობა და დახარჯული დრო (რესურსები) [76].

- *ბუნებრიობა* – პროგამული კოდი უნდა იყოს მარტივი, გასაგები (ადვილად წასაკითხი), მოდულური. თითოეული მოდული ბუნებრივად (უპრობლემოდ) უნდა მიუერთდეს პროგრამას. პროგრამული კოდი, თავისთავად მარტივად უნდა ექვემდებარებოდეს რედაქტირებას, ინტეგრირებას ან ცალკეული ფუნქციების ამოღებას. კოდის ცვლიებამ მის ერთ ნაწილში არ უნდა გამოიწვიოს სერიოზული ცვლილებები მის სხვა ბაწილებში;
- *მწარმოებლურობა* – პროგამული პროდუქტის ოპტიმიზაციით უნდა ამაღლდეს კოდის ფუნქციონირების სწრაფქმედება მის საწყის ვერსიასთან შედარებით. მწარმოებლურობის შეფასება (ტესტირება), პროგრამული სისტემების ხარისხის უზრუნველყოფის შემთხვევაში, ემსახურება მისი სხვა ატრიბუტების გამოკვლევას, გაზომვას და შემოწმებას, მაგალითად, როგორიცაა მასშტაბირებადობა (scalability), საიმედოობა (reliability) და რესურსების გამოყენება.

პროგრამული უზრუნველყოფის მწარმოებლურობის ტესტირება არის კომპიუტერული მეცნიერების პრაქტიკა, რომელიც ნერგავს მწარმოებლურობის სტანდარტებს მათი დაპროექტების, სისტემის

არქიტექტურის განსაზღვრის და იმპლემენტაციის ეტაპებზე;

- *დახარჯული დრო* – პროგრამის ოპტიმიზაცია და მისი შემდგომი გამართვა უნდა იკავებდეს დროის მცირე პერიოდს. პირობითად მიღებულია, რომ ოპტიმიზაცია/გამართვის დრო იყოს 10-15% იმ დროისა, რომელიც თვით საწყისი კოდის დაწერაზე დაიხარჯა. წინააღმდეგ სემთხვევაში იგი იქნება არარენტაბელური [76].

3.2. პროგრამული სისტემების ოპტიმიზაციის მეთოდები

3.2.1. დამახსოვრების მეთოდი და პროფაილერი

„მემორიზაციის“ მეთოდი (Memory - მეხსიერება, Memoization - დამახსოვრება) ნიშნავს კოდის რომელიმე ფუნქციის შესრულების საფუძველზე მიღებული შედეგების შენახვას მეხსიერებაში, რათა შესაძლებელი იყოს მათი მრავალჯერადი გამოყენება ხელმეორე გადაანგარიშების გარეშე [76,77]. ეს კი მნიშვნელოვნად ამაღლებს მთლიანი პროგრამის ფუნქციონირების სისწრაფეს, ანუ მწარმოებლურობას.

აღნიშნული მეთოდის მუშაობა მარტივი ალგორითმით ხდება, კერძოდ, სანამ რომელმე კონკრეტული ფუნქცია დაიწყებს მუშაობას, წინასწარ მოწმდება პირობა - ხომ არ ყოფილა იგი უკვე გამოყენებული. შედეგად ორი ვარიანტია:

1) ფუნქცია პირველადაა ამუშავებული, მაშინ ის გაითვლის მნიშვნელობას და შეინახავს მეხსიერებაში;

2) ფუნქცია უკვე იყო შესრულებული, მაშინ აიღება შენახული მნიშვნელობა, ხელახალი გაანგარისების პროცესის გარეშე.

- **პროფაილერი (Profiler)**

პროფაილერები (Profilers) პროგრამირების ინსტრუმენტებია, რომლებიც აანალიზებს პროგრამის ქცევას მისი მუშაობისას. პროგრამული უზრუნველყოფის დეველოპმენტის დროს კოდში შეიძლება იყოს პრობლემური არეები (უბნები), რაც გამოწვეულია არაეფექტური პროგრამირებით [73]. პროფაილერი-პროგრამა ეხმარება დეველოპერს სწორედ ასეთი პრობლემური უბნების გამოვლენაში გაშვებული პროგრამების ანალიზისა და შედარების გზით. შედეგად მიღებულ იქნება რაოდენობრივი შეფასებები (ზომები) კოდის სტრუქტურული და ალგორითმული გაუმჯობესებისთვის.

ამგვარად, პროგრამულ ინჟინერიაში „პროფილირება“ ("program profiling", "software profiling")" არის პროგრამის დინამიკური ანალიზის ფორმა, რომელიც ზომავს, მაგალითად, პროგრამის სივრცეს (მეხსიერებას) ან შესრულების დროს, კონკრეტული ინსტრუქციების გამოყენებას ან ფუნქციების გამოძახების სიხშირეს და ხანგრძლივობას.

ინფორმაციის პროფილირება ხელს უწყობს პროგრამის ოპტიმიზაციას. პროგრამების ავტომატური ტესტირების პროცესში პროფაილერის (ანუ კოდის შესრულების ანალიზატორის) გამოყენება სასურველია კოდის იმ მნიშვნელოვანი სექციების (უბნების) საპოვნელად, რომლებიც *ყველაზე მეტ რესურსს* მოიხმარს. კოდის არამნიშვნელოვანი

უბნების ოპტიმიზაცია ნაკლებად უწყობს ხელს პროგრამის მწარმოებლურობის გაუმჯობესებას.

3.2.2. კეშირების მეთოდი

კეშირების მეთოდი (caching method) არის მონაცემთა დროებითი შენახვის ხერხი მომხმარებლის მანქანის მეხსიერებაში. ესაა შუალედური ბუფერი, მასთან მიმართვა (წვდომა) გაცილებით სწრაფად ხდება, ვიდრე ყოველ ჯერზე სერვერთან ან მონაცემთა ბაზაასთან კავშირით [76]. კეშირების მეთოდის გამოყენებით საგრძნობლად იზრდება საიტებთან და ონლაინ სისტემებთან მუშაობის სისწრაფე. კეშ-მეხსიერება შეიძლება იყოს კომპიუტერის დისკოზე ან ოპერატიულ მეხსიერებაში გამოყოფილი ნაწილი. პროგრამა, მუშაობის პროცესში ჯერ მიმართავს კეშ-მეხსიერებას მონაცემებისათვის და თუ აქ ვერ იპოვა, მაშინ გადადის მონაცემთა ძირითადი საცავისკენ.

3.2.3. პარალელური ალგორითმების მეთოდი და პარალელური კომპიუტინგი

პარალელური ალგორითმების (Parallel algorithms) მეთოდი გამოყენება მულტიპროცესორული არქიტექტურის სისტემებისთვის [76, 78].

ასეთი ალგორითმები რეალიზებულია კომპიუტერული პროგრამების სახით და შესაძლებელია მათი ერთდროული (პარალელური) მუშაობა სხვადასხვა პროცესორზე, რაც მნიშვნელოვნად ამაღლებს მთლიანი პროგრამული უზრუნ-ველყოფის მწარმოებლურაბას.

ეს მეთოდი განსაკუთრებულია პროგრამული სისტემების ოპტიმიზაციისათვის და მოითხოვს გადასაწყვეტი ამოცანების ბიზნეს-პროცესების მიზეზ-შედეგობრივი მიმართებების კვლევას. ასეთი კვლევებისათვის მოსახერხებელია, მაგალითად, ისეთი მათემატიკური, გრაფო-ანალიზური მოდელების გამოყენება, როგორიცაა პეტრის ქსელები, კერძოდ განაწილებული ალგორითმების თეორია [79, 80].

განაწილებული ალგორითმები (distributed algorithms) ქვეკლასია პარალელური ალგორითმებისა. მათი გამოყენება მიზანშეწონილია კლასტურული კომპიუტინგისა და განაწილებულ გამოთვლით გარემოში მუშაობისათვის, სადაც საჭიროა დამატებითი პრობლემების გადაწყვეტა, რაც სცილდება კლასიკურ პარალელურ ალგორითმებს.

პარალელური გამოთვლები (Parallel computing) არის გამოთვლების ტიპი, რომელშიც მრავალი გამოთვლა ან პროცესების შესრულება ერთდროულად ხორციელდება. მისი კონცეფცია იმაშია, რომ დიდი პრობლემები ხშირად შეიძლება დაიყოს უფრო მცირე ზომისად, რომელთა მოგვარება შესაძლებელია იმავე დროში. პარალელური გამოთვლების რამდენიმე ფორმა არსებობს, კერძოდ, პარალელიზმი ბიტების დონეზე, ბრძანებების დონეზე, მონაცემების დონეზე და დავალებების დონეზე [81].

3.2.4. კონკურენტული კომპიუტინგის მეთოდი

პარალელური კომპიუტინგი მჭიდრო კავშირშია კონკურენტულ კომპიუტინგთან. პარალელური კომპიუტინგი (parallel computing) და კონკურენტული კომპიუტინგი (concurrent computing) ხშირად გამოიყენება ერთად, თუმცა კონფლიქტური სიტუაციებიც არსებობს მათ შორის პარალელური გამოთვლების დონეებზე [81].

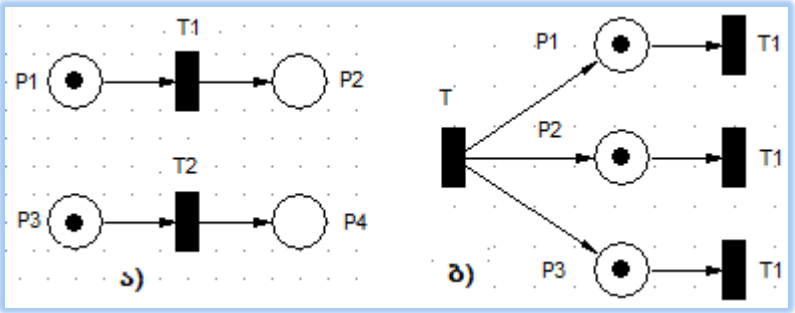
კონკურენტული პროგრამა არის განსაზღვრული ქმედებები, რომლებიც შეიძლება შესრულდეს ერთდროულად. განსაკუთრებული ყურადღება ეთმობა იმ პროგრამებს, რომლებიც ითვლება კონკურენტულად, ანუ ეს ის პროგრამებია, რომლებიც აგებულია რეალური ობიექტების მართვის ან მოდელირებისათვის და რომლებიც მოიცავს პარალელურ პროცესებს. პარალელური პროგრამირების დროს პარალელური დამუშავება მიიღწევა ტექნიკის პარალელიზმის გზით, მაგალითად, ისინი ერთდროულად სრულდება პროცესორის ორ ცალკეულ ბირთვზე.

კონკურენტულ და პარალელურ პროგრამირებას შორის არსებობს განსხვავება {82}. პარალელური პროგრამირებისას, პარალელური დამუშავება მიიღწევა ტექნიკის პარალელიზმის გზით, მაგალითად, ერთდროულად ორი პროცესორის (T_1, და T_2) ორ ცალკეულ ბირთვზე შესრულება (ნახ.3.1-ა).

კონკურენტული პროგრამირებისას კი ტექნიკის პარალელიზმი არაა (ნახ.3.1-ბ) [83-85].

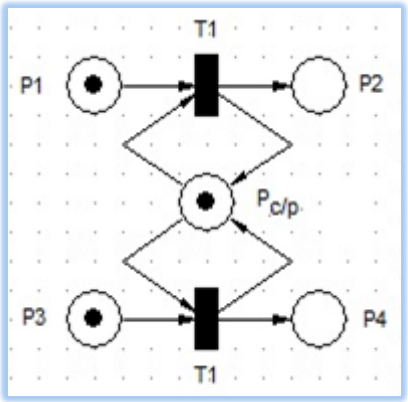
პარალელურ რიგში მდგარი პროცესები კონკურენტულია, ანუ ან ერთი მიიღებს T_1 - რესურსს, ან მეორე ან

მესამე და ა.შ. ამგვარად, T_1 მოემსახურება ყველა დავალებას, ოღონდ დროითი დაყოვნებით [86].



ნახ.3.1. პარალელური (ა) და კონკურენტული (ბ) პროგრამირების მოდელების შედარება პეტრის ქსელის გრაფით

3.2 ნახაზზე ნაჩვენებია კონკურენტული პროცესების პრიორიტეტებით მართვა (Pc/p). ერთი პროცესორი ემსახურებაველა პროცესს.



ნახ.3.2. კონკურენტული პროგრამირების მოდელი პროცესთა პრიორიტეტების მართვით ($P_{c/p}$)

ამგვარად, კონკურენტული სქემის მოდელირებისას „კონკურენციაა“, როდესაც დავალებები სრულდება მონაცვლეობით (ერთი რესურით) და არა თანმიმდევრობით, ერთმანეთის მიყოლებით. „პარალელიზმია“, როდესაც ეს ამოცანები სრულდება ერთდროულად - პარალელურად (სხვადასხვა რესურსით).

3.2.5. „ზარმაცული“ გამოთვლების მეთოდი

„ზარმაცული" შეფასება ან *„გამოძახება საჭიროებისამებრ“* (Lazy evaluation or call-by-need) პროგრამირების ენის თეორიაში, არის შეფასების ევოლუციური სტრატეგია, რომელიც აყოვნებს შეფასებას მანამდე, სანამ მისი მნიშვნელობა არ იქნება საჭირო (არამკაცრი შეფასება), აგრეთვე გამორიცხავს განმეორებით შეფასებას (ერთობლივი გამოყენება) [76,87].

ასეთი სტრატეგია მნიშვნელოვნად ამცირებს ჩასატარებელი ანგარიშების მოცულობას, რადგან არააუცილებელი გათვლების პროცესები არ შესრულდება. მეთოდმა რომ იმუშაოს, საჭიროა შესაბამისი ფუნქციის ოპერანდებს შორის დამოკიდებულებები აღიწეროს. შედეგად მიიღება კოდი, რომელიც ამოქმედდება მხოლოდ მაშინ, როცა ის აუცილებლად საჭირო იქნება.

„ზარმაცული“ შეფასების უპირატესობებია:

- მართვის ნაკადის (სტრუქტურების) განსაზღვრის შესაძლებლობა აბსტრაქციის სახით;

- პოტენციურად მონაცემთა უსასრულო სტრუქტურების განსაზღვრის შესაძლებლობა. ეს საშუალებას იძლევა ზოგიერთი ალგორითმი განხორციელდეს უფრო მარტივად;
- მწარმოებლურობა იზრდება არასაჭირო გამოთვლების თავიდან აცილების გზით და შეცდომების აღმოფხვრით შედგენილი გამოსახულებების შეფასებისას.

კომპიუტერული დაპროგრამების დროს *„მოუთმენელი“ შეფასება - (Eager evaluation)* ან ცნობილი როგორც „მკაცრი შეფასება“ – არის შეფასების სტრატეგია, რომელიც გამოიყენება თითქმის ყველა ტრადიციული პროგრამირების ენის მიერ. მოუთმენელი შეფასების დროს გამოსახულება ფასდება მაშინვე, როგორც კი იგი დაუკავშირდება ცვლადს [88]. შეფასების ასეთი მეთოდი საპირისპირო ალტერნატივაა „ზარმაცული შეფასებისა, რომელზეც ზემოთ გვქონდა საუბარი. მოუთმენელი შეფასების დროს გვაქვს ასეთი შედეგები:

- კოდი, რომელიც ადვილად გასაგებია მისი შესრულების მიმდევრობის თვალსაზრისით, არ შეცვლის თავის ქცევას შესრულების კონტექსტის ცვლილების საფუძველზე;
- აქვს კოდის გამართვის უფრო მარტივი პროცესი, შეფასების სხვა სტრატეგიებთან შედარებით;
- კოდის შესრულების მწარმოებლურობაზე პასუხის-მგებლობა გადახრილია დეველოპერებისკენ, რაც მოითხოვს კოდის ოპტიმიზაციის ფრთხილად შესრულების პროცესს.

3.3. პროგრამისთვის პროცესორის დროის ოპტიმიზაცია

- ***არითმეტიკული ოპერაციები.***

კომპიუტერული პროგრამების დიდი უმრავლესობა ფუნქციური ამოცანების გადასწყვეტად იყენებს არითმეტიკულ გაანგარიშებებს. არითმეტიკული და ლოგიკური ოპერაციების დაპროგრამებაში არის გარკვეული სარეზერვო რესურსები მთლიანი პროგრამის სწარფქმედების ასამაღლებლად. შეიძლება ითქვას, რომ სხვადასხვა არითმეტიკული ოპერაციის შესრულების სისწრაფე მნიშვნელოვნად განსხვავებულია. ყველაზე სწრაფია შეკრებისა და გამოკლების ოპერაცია. შედარებით ნელია გამრავლება, ხოლო უფრო ნელი - გაყოფა [74].

მაგალითად, თუ გვაქვს გამოსახულება $\frac{x}{a}$, სადაც a – კონსტანტაა, მცოცავმძიმიანი არგუმენტებისთვის გაანგარიშება მოხდება უფრო სწრაფად თუ გამოვიყენებთ ასეთ ეკვივალენტურ გამოსახულებას $x * b$, სადაც $b = \frac{1}{a}$ – კონსტანტაა, რომელიც გაითვლება პროგრამის კომპილაციის ეტაპზე (აქ ნელი გაყოფის ოპერაცია შეიცვალა შედარებით სწრაფი გამრავლების ოპერაციით). მთელრიცხვა არგუმენტისთვის გამოსახულება 2*x უფრო სწრაფად შესრულდება ასეთი სახით: x + x (გამრავლების ნელი ოპერაცია შეიცვალა შეკრების სწრაფი ოპერაციით) და ა.შ.

არითმეტიკული გამოსახულებების დაპროგრამებისას მოფიქრებულ უნდა იქნას ისეთი ფორმა, რომელშიც ნელი

ოპერაციების რაოდენობა იქნება მინიმალური. მაგალითად, გვაქვს ასეთი გამოსახულება:

$$ax^4 + bx^3 + cx^2 + dx + e$$

სადაც არის 10 გამრავლების ოპერაცია (ნელი), და 4 შეკრებისა (სწრაფი). იგივე გამოსახულება ეკვივალენტური ფორმით ჩაწერილია გორნერის მეთოდით (ინგლისელი მათემატიკოსი William George Horner) [89]:

$$\big(\big((ax + b)x + c\big)x + d\big)x + e$$

წარმოდგენილი ფორმა შეიცავს გამრავლების 4 ოპერაციას და 4 - შეკრებისას. შედეგად შემცირდა ოპერაციების რაოდენობა, თითქმის ორჯერ. ეს კი საგრძნობლად შეამცირებს გაანგარიშების ჯამურ დროს, რაც ოპტიმიზაციის კარგი მაგალითია.

ასეთი ოპტიმიზაციები არის ალგორითმული და, ჩვეულებისამებრ, მათ კომპილატორები არ ასრულებს ავტომატურად.

3.4. სიმძლავრის შემცირება

გამოთვლითი დავალებების შესრულება შესაძლებელია სხვადასხვა ხერხით, განსხვავებული ეფექტურობით. უფრო ეფექტური ვერსია *ექვივალენტური ფუნქციურობით* არის ცნობილი როგორც *სიმძლავრის შემცირება* [74]. მაგალითად, შემდეგი C კოდის ჩანაწერი, რომელიც ითვლია მთელი რიცხვების ჯამს 1-დან N-მდე :

```
int i, sum = 0;
for (i = 1; i <= N; ++i) {
  sum += i;
```

```
}
printf("sum: %d\n", sum);
```

ეს კოდი შეიძლება შეიცვალოს მათემატიკური შემდეგი ფორმულით:

```
int sum = N * (1 + N) / 2;
printf("sum: %d\n", sum);
```

ოპტიმიზაცია, რომელიც ზოგჯერ ავტომატურად ხორციელდება ოპტიმიზაციის კომპილატორით, არის ალგორითმის შერჩევის მეთოდი, რომელიც გამოთვლების თვალსაზრისით უფრო ეფექტურია, ამავე დროს მან უნდა შეინარჩუნოს იგივე ფუნქციონალობა. ხშირად კოდის შესრულების მნიშვნელოვანი გაუმჯობესება შეიძლება მიღწეული იქნეს გარეშე ფუნქციების ამოღებით.

ოპტიმიზაცია ყოველთვის არაა ცხადი ან ინტუიციურად გასაგები პროცესი. ზემოთ მოყვანილ მაგალითში, „ოპტიმიზირებული" ვერსია შეიძლება სინამდვილეში უფრო ნელი იყოს ვიდრე თავდაპირველი ვერსია, თუ N იქნება საკმაოდ მცირე, ხოლო კონკრეტული აპარატურა უფრო სწრაფია დამატებების და ციკლების შესასრულებლად, ვიდრე გამრავლება და გაყოფა.

3.5. მაკროსები

მაკროების გამოყენებით კოდის შემუშავებისას ოპტიმიზაცია სხვადასხვა ენაზე იღებს სხვადასხვა ფორმას [74].

მაგალითად, C++ და C-ზე მაკროსი ხორციელდება მარკერებით. დღესდღეობით, ჩაშენებული ფუნქციები

შეიძლება გამოყენებულ იქნას, ხშირ შემთხვევაში, როგორც უსაფრთხო ალტერნატივა. ორივე შემთხვევაში, ჩაშენებულ ფუნქციას შეუძლია კომპილატორის საშუალებით გაიაროს შემდგომი ოპტიმიზაცია.

3.6. პროგრამების ავტომატური და ხელით ოპტიმიზაცია

ოპტიმიზაცია შესაძლებელია ავტომატიზებით – კომპილატორის მიერ ან შესრულდეს ხელით – პროგრამისტის მიერ [74,76]. სასურველი შედეგის მიღწევა მხოლოდ ლოკალური ოპტიმიზაციით შეზღუდულია, ხოლო გლობალური ოპტიმიზაციისთვის – მეტი. ყველაზე მძლავრი ოპტიმიზაცია არის უკეთესი ალგორითმის პოვნა. როგორც წესი, მთელი სისტემის ოპტიმიზაცია სრულდება პროგრამისტების მიერ, რადგან იგი ძალზე რთულია ავტომატიზებული ოპტიმიზატორისათვის. ამ სიტუაციაში, პროგრამისტი ან სისტემის ადმინისტრატორი ცხადად ცვლის კოდს, ისე რომ საერთო სისტემა უკეთესად მუშაობდეს. მათ შეუძლიათ უკეთესი ეფექტურობის მიღწევა, მაგრამ ეს გაცილებით ძვირია, ვიდრე ავტომატური ოპტიმიზაცია.

რადგან მრავალი პარამეტრი ახდენს გავლენას პროგრამის მწარმოებლურობაზე (განსაკუთრებით პარალელურ გამოთვლით სისტემებში), ამიტომ პროგრამის ოპტიმიზაციის სივრცე დიდია და დასმული პრობლემის გადასაჭრელად იყენებენ მეტა-ევრისტიკასა და მანქანურ დასწავლას (ნახ.3.3) [90].

ნახ.3.3. მეტა-ევრისტიკის და მანქანური დასწავლის მეთოდების (ალგორითმების) კლასიფიკაცია

ქვემოთ განვიხილავთ მეტა-ევრისტიკისა და მანქანური დასწავლის აღნიშნული მეთოდების და ალგორითმების დანიშნულებას და გამოყენების არეალს, რადგან მათ განსაკუთრებული მნიშვნელობა შეიძინეს *პარალელური გამოთვლების* სისტემების ამოცანების ეფექტურად გადაწყვეტის თვალსაზრისით, შესაბამისი პროგრამული უზრუნველყოფის ოპტიმიზაციის საფუძველზე.

3.6.1. პროგრამების ოპტიმიზაციის მეტა-ევრისტიკული ალგორითმები

ევრისტიკული მართვის მეთოდები ემსახურება ოპტიმიზაციის პრობლემების სწრაფ გადაწყვეტას მიახლოებითი მნიშვნელობების განსაზღვრის გზით მაშინ, როდესაც სხვა მეთოდები ძალზე ნელია ან ვერ გვაძლევს ზუსტ შედეგებს [90].

ევრისტიკული მეთოდები გამოიყენება კონკრეტული პრობლემების და ამოცანების გადასაწყვეტად, იგი პრობლემურ-ორიენტირებულია. *მეტა-ევრისტიკა* კი – ზოგადია, მულტიპრობლემური, ამიტომაც მისი გამოყენების არეალი უფრო ფართოა. მისი დახმარებით შესაძლებელია ოპტიმიზაციის ამოცანების გადაწყვეტის თითქმის ოპტიმალური მნიშვნელობების განსაზღვრა, გამოთვლითი რესურსების და საპრობლემო სფეროს შესახებ ცოდნის არსებობის შეზღუდულ პირობებში.

განვიხილოთ მოკლედ ზოგიერთი ზემოაღნიშნული მეთოდი:

- *გენეტიკური ალგორითმი (Genetic algorithm* – GA*)* – არის მეტა-ევრისტიკული ძებნის ტიპის ალგორითმი, რომელიც გამოიყენება ოპტიმიზაციისა და მოდელირების პრობლემების გადასაჭრელად, სასურველი პარამეტრების შემთხვევითი შერჩევით, კომბინირებისა და ცვალებადობით [92]. იგი იყენებს ბუნებრივი გადარჩევის მსგავს მექანიზმებს და მიეკუთვნება ევოლუციური გამოთვლების ალგორითმების კლასს. მაგალითად, მემკვიდრეობითობა, მუტაცია, შერჩევა და გადაკვეთა („ჯვრისწერა“).

გენეტიკური პროგრამირება (Genetic programming – GP) – არის გენეტიკური ალგორითმების მონათესავე ტექნიკა, რომელიც *ოპტიმიზირებას უკეთებს* თვით კომპიუტერულ *პროგრამებს* და არა პროგრამის ფუნქციათა პარამეტრებს. გენეტიკური პროგრამირება ხშირად იყენებს მონაცემთა ხისმაგვარ სტრუქტურებს პროგრამების წარმოსადგენად, ნაცვლად. მონაცემთა სიების სტრუქტურისა, რაც დამახასიათებელია გენეტიკური ალგორითმებისთვის [93];

- *წრთობის იმიტაცია (Simulated annealing – SA)* – არის ალბათური მეთოდი მოცემული ფუნქციის გლობალური ოპტიმუმის აპროქსიმაციისთვის [94]. კერძოდ, ესაა მეტაევრისტიკული მეთოდი გლობალურ ოპტიმუმთან მიახლოებისათვის ძიების დიდ სივრცეში. იგი ხშირად გამოიყენება დისკრეტული საძიებო ადგილის შემთხვევაში (მაგალითად, ოპერაციათა კვლევაში, კომივოიაჟერის ამოცანა) იმ პრობლემებისთვის, სადაც მიახლოებითი გლობალური ოპტიმუმის პოვნა უფრო მნიშვნელოვანია, ვიდრე ზუსტი ლოკალური ოპტიმუმის პოვნა ფიქსირებულ დროში;

- *დიფერენციალური ევოლუცია (Differential evolution – DE)* – არის მრავალგანზომილებიანი მათემატიკური ოპტიმიზაციის მეთოდი, რომელიც მიეკუთვნება სტოქასტური ოპტიმიზაციის ალგორითმების კლასს (ანუ მუშაობს შემთხვევითი რიცხვების გამოყენებით) და იყენებს გენეტიკური ალგორითმების ზოგიერთ იდეას [95], მაგრამ, მათგან განსხვავებით, არ მოითხოვს ცვლადებთან მუშაობას ბინარულ კოდში. ესაა ოპტიმიზაციის პირდაპირი მეთოდი, ანუ ის მოითხოვს მხოლოდ მიზნობრივ ფუნქციათა გამოთვლის

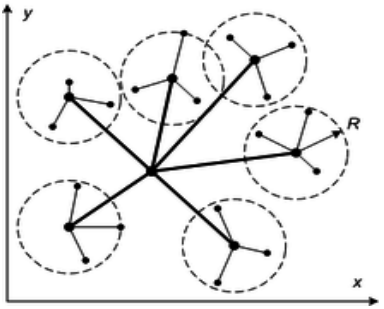
შესაძლებლობას, მაგრამ არა მათი წარმოებულებისა. დიფერენციალური ევოლუციის მეთოდი მიზნად ისახავს მრავალი ცვლადის გლობალური მინიმუმის (ან მაქსიმუმის) პოვნას არადიფერენცირებადი, არაწრფივი, მულტიმოდუ-ლური (შესაძლოა, ლოკალური ექსტრემუმის მრავალი მნიშვნელობით) ფუნქციებისთვის. მეთოდი მარტივია მისი განხორციელებისა და გამოყენებისთვის (შეიცავს რამდენიმე ასარჩევ მმართველ პარამეტრს), ადვილად პარალელურდება;

- *ჭიანჭველების კოლონის ოპტიმიზაცია (Ant colony optimization – ACO)* – არის პოპულაციაზე ბაზირებული მეტა-ევრისტიკული მეთოდი [96]. მისი საშუალებითა შესაძ-ლებელია რთულ ოპტიმიზაციასთან დაკავშირებული პრობლემების მიახლოებითი გადაწყვეტა. ACO-ში, პროგრა-მული უზრუნველყოფის აგენტების ერთობლიობა, სახელწოდებით „ხელოვნური ჭიანჭველები“, ეძებს კარგ გადაწყვეტილებებს მოცემული ოპტიმიზაციის პრობლემის მოსაგვარებლად. ამ მიდგომის საფუძველია ჭიანჭველების ქცევის მოდელის გაანალიზება და გამოყენება, როცა ისინი ეძებენ კოლონიდან საკვების წყაროსკენ მიმავალ გზებს. აღნიშნული მეტა-ევრისტიკული ოპტიმიზაციის მეთოდი შეიძლება გამოყენებულ იქნას გრაფებზე მარშრუტების მოძიების მსგავსი ამოცანების გადასაჭრელად;

- *ფუტკრის კოლონის ალგორითმი (Artificial bee colony optimization – ABC)* – არის ერთ-ერთი პოლინომიური მეტა-ევრისტიკული ალგორითმი კომპიუტერულ მეცნიერებაში და ოპერაციათა კვლევის სფეროში ოპტიმიზაციის პრობლემების გადასაჭრელად. ის მიეკუთვნება სტოქასტური ბიონიკური

ალგორითმების (მაგალითად, მონტე-კარლოს მეთოდი) კატეგორიას, ბუნებაში ნექტარის შეგროვებისას, ფუტკრის კოლონის ქცევის იმიტაციაზე დაყრდნობით (ნახ.3.4) [97].

3.4 ნახაზზე ნაჩვენებია ABC-ს დაზვერვის ორგანზომილებიანი სივრცის სტრატეგიის სქემა (მსხვილი ხაზები – მზვერავი ფუტკრების გაფრენის ტრაექტორიები, წვრილი ხაზები – მუშა ფუტკრების მიერ გადაწყვეტილების დაზუსტება.



ნახ.3.4. ABC-ს ოპტიმიზაციის სტრატეგია

- *ნაწილაკების გროვის ოპტიმიზაცია (Particle Swarm Optimization - PSO)* - არის კომპიუტერული მეცნიერებებში გამოთვლითი მეთოდი, რომელიც ოპტიმიზაციას უკეთებს პრობლემას იტერაციულად, ცდილობს გააუმჯობესოს მისაღები გადაწყვეტილება ხარისხის მოცემული კრიტერიუმების გათვალისწინებით [98].

PSO ალგორითმის ძირითადი ვარიანტი მუშაობს, აქვს შესაძლო გადაწყვეტილებების (ნაწილაკების) პოპულაცია (გროვა - Swarm). ეს ნაწილაკები გადაადგილდება საძიებო სივრცეში რამდენიმე მარტივი ფორმულის მიხედვით [99]. ნაწილაკთა მოძრაობები განისაზღვრება როგორც საკუთრივ ყველაზე ცნობილი პოზიციის მიხედვით საძიებო სივრცეში, ასევე მთელი გროვის ყველაზე ცნობილი პოზიციით. როცა მოძებნილ

იქნება გაუმჯობესებული პოზიციები, აქეთკენ წარიმართება გროვის მოძრაობა. პროცესი მეორდება და ამით საიმედო, მაგრამ არა-გარანტირებულია, საბოლოო შედეგების დამაკმაყოფილებელი მნიშვნელობები.

- *ლოკალური ძებნა (Local search - LS)* – არის ზოგადი ტერმინი კომბინატორული ოპტიმიზაციის რიგი მეტა-ევრისტიკული ძებნის მეთოდების. მეთოდი მრავალი ვარიანტით გამოიყენება რთული ოპტიმიზაციის პრობლემების გადასაჭრელად (მაგალითად, კომივოიაჟერის ამოცანა). ძირითადი პრინციპი მდგომარეობს უკეთესი შედეგის პოვნაში მოცემული საწყისი გადაწყვეტილების საფუძველზე, ამ უკანასკნელის ლოკალური ცვლილებებით განსახილველ სამეზობლო არეში [100]. ლოკალური ძებნა (სამეზობლოში) იღებს ამოცანის პოტენციურ გადაწყვეტას და, უკეთესი შედეგის პოვნის იმედით, ამოწმებს მის უშუალო მეზობლებს (ე.ი. გადაწყვეტილებები, რომლებიც მსგავსია, გარდა რამდენიმე ძალიან მცირე დეტალისა). ლოკალური ძებნის მეთოდებს აქვს სუბოპტიმალურ ადგილებში „გაჭედვის" ტენდენცია (რასაც ტაბუ-ძებნის მეთოდი აგვარებს);
- *ტაბუ-ძებნა* ან *ძებნა აკრძალვებით (Tabu search - TS)* – არის მეტა-ევრისტიკული ძებნის მეთოდი, რომელიც იყენებს ლოკალურ ძებნას მათემატიკური ოპტიმიზაციისათვის [101]. აკრძალული (ტაბუ) ძებნა აუმჯობესებს ლოკალური ძებნის მწარმოებლურობას, მისი ძირითადი წესის შესუსტებით. ტაბუ-ძებნაში შემოღებულია აკრძალვები (ანუ ისინი ტაბუდადებულია), რათა თავიდან იქნას აცილებული უკვე განხილული გადაწყვეტილებების ხელმეორედ ძებნა.

ტაბუ-ძებნის რეალიზაცია იყენებს სტრუქტურებს, რომლებიც აღწერს მონახულებულ გადაწყვეტილებებს ან მომხმარებლის წესების ერთობლიობას. თუ რომელიმე პოტენციური გადაწყვეტილება უკვე იქნა განხილული დროის მოკლე პერიოდში, ან თუ ის არღვევს წესებს, იგი მოინიშნება როგორც „ტაბუ" და მას ალგორითმი აღარ განიხილავს;

- *ჰარმონიის ძებნა (Harmony search - HS)* არის მეტა-ევრისტიკული ძებნის ალგორითმი, რომელიც ცდილობს მიბაძოს მუსიკოსთა იმპროვიზაციის პროცესს სასიამოვნო ჰარმონიის პოვნაში. შეინიშნება, რომ HS-მა აჩვენა პერსპექ-ტიული შედეგები ოპტიმიზაციის რთული ამოცანების გადასაწყვეტად. ამიტომაც შემუშავდა ამ ალგორითმის განსხვავებული ვერსიები [102.103].

3.6.2. პროგრამების ოპტიმიზაციის მანქანური დასწავლის ალგორითმები

როგორც ზემოთ აღვნიშნეთ, პროგრამული უზრუნველყოფის ოპტიმიზაციის მიზნით, განსაკუთრებით პარალელური კომპიუტინგის სფეროში, აქტიურად გამოიყენება მეტაევრისტიკული და მანქანური დასწავლის ალგორითმები [90]. წინა პარაგრაფში ჩვენ მოკლედ განვიხილეთ ევრისტიკული მიდგომები, ახლა კი მანქანური დასწავლის მეთოდებს შევეხებით, კერძოდ, პროგრამული სისტემების ოპტიმიზაციის თვალსაზრისით.

მანქანური დასწავლის პროგრამები მუშაობს პროგნოზირების მოდელის შექმნის საფუძველზე მასწავლი მონაცემების ერთობლიობიდან. ეს მოდელები შემდეგ გამოიყენება პროგნოზირების მიზნით მონაცემების საფუძველზე. მანქანური დასწავლის ზოგიერთი პოპულარული ალგორითმია: რეგრესიული, გადაწყვეტილების ხე, ვექტორული მანქანის მხარდამჭერი, ბაიესური დასკვნა, შემთხვევითი ტყე და ხელოვნური ნეირონული ქსელები (ნახ.3.3).

პროგნოზირების მოდელის დასწავლის მეთოდისაგან დამოკიდებულებით, მანქანური დასწავლა შეიძლება იყოს მართვადი ან არამართვადი.

მართვადი მანქანური დასწავლის შემთხვევაში პროგნოზირების მოდელი სწავლობს არსებულ მაგალითებზე, ანუ შემავალი და გამომავალი მონაცემები დასწავლის მონაცემთა ერთობლიობაში ცნობილია. მართვად დასწავლაში გამოიყენება *კლასიფიკაციის მეთოდები* ცალკეული პასუხების პროგნოზირებისათვის, აგრეთვე რეგრესიის მეთოდები -

უწყვეტი პასუხებისთვის (მაგალითად, ტემპერატურის ცვლილება). უკეთესი ალგორითმის შერჩევა დამოკიდებულია შემავალ მონაცემთა რაოდენობასა და ტიპზე, აგრეთვე საურველ შედეგზე და მათი გამოყენების სახეზე.

მანქანური დასწავლის *არამართვადი* მოდელებისათვის პრაქტიკულად უცნობია თუ როგორი სახე აქვს შედეგებს. აქ სწორი შედეგები (მასწავლი მონცემების ერთობლიობა) არ გამოიყენება მოდელის დასწავლისათვის. მოდელის მიზანია მონაცემებში *ფარული შაბლონების პოვნა* მასწავლ მონცემთა ერთობლიობის სტატისტიკური თვისებების საფუძველზე. არამართვადი დასწავლა შეიძლება გამოყენებულ იქნას, მაგალითად, მონაცემთა კლასტერიზაციის პრობლემის გადასაწყვეტად სხვადასხვა სფეროში.

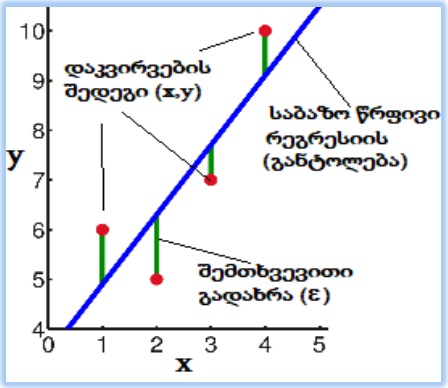
მანქანური დასწავლა (Machine learning – ML) – არის სამეცნიერო მიმართულება, რომელიც კომპიუტინგის სისტემებს, სრულყოფის მიზნით, აძლევს თვითგანსწავლის შესაძლებლობას ავტომატიზებულ რეჟიმში. მისი მეთოდები და ალგორითმები ორიენტირებულია ისეთი პროგრამული სისტემების შექმნაზე, რომლებსაც გამოცდილების საფუძველზე და ხელმისაწვდომი მონაცემების ბაზაზე შეუძლია ახალი ცოდნის შეძენა, ანუ თვითსრულყოფა. იგი ხელოვნური ინტელექტის პრაქტიკული გამოყენების სფეროა, ჭკვიანი პროგრამული აპლიკაციების ასაგებად [90,104].

ამ მიმართულებით, როგორც ლიტერატურულ წყაროებშია აღნიშნული, პერსპექტიულად ითვლება შემდეგი მეთოდები ან ალგორითმები [90] (ნახ.3.3):

- *რეგრესიის ალგორითმები (Regression algorithms - RA)* – მიეკუთვნება მართვადი მანქანური დასწავლის ალგორითმების ოჯახს. რეგრესიის ალგორითმები აპროგნოზირებს გამომავალ მნიშვნელობებს (შედეგებს), სისტემის საწყისი (შესატანი) მონაცემების საფუძველზე. იგი გამოიყენება მრავალ სფეროში: ჯანდაცვა, განათლება, ფინანსები, ტრანსპორტი და ა.შ. [105].

მაგალითისათვის განვიხილოთ წრფივი რეგრესიის და ლოგისტიკური რეგრესიის ალგორითმები.

- *წრფივი რეგრესია (Linear Regression - LR).* – სტატისტიკაში გამოიყენება რეგრესიული მოდელი ერთი დამოკიდებული y ცვლადისა ერთ (ან რამდენიმე) სხვა დამოუკიდებელ x ცვლადზე (ფაქტორები, რეგრესორები) (ნახ.3.5) [106,107].



ნახ.3.5. წრფივი რეგრესია

წრფივი დამოკიდებულების ფუნქცია ჩაიწერება შემდეგი გამოსახულებით:

$$y = b_1x_1 + b_2x_2 + \ldots + b_nx_n + \varepsilon$$

სადაც b და ε კოეფიციენტები გაითვლება უმცირეს კვადრატთა მეთოდით. იგი მდგომარეობს შემდეგში, რომ

ყოველი დამუკიდებელი (x) და დამოკიდებული (y) ცვლადების წყვილებისთვის (x_1 , y_1), (x_2 , y_2), . . . (x_n , y_n) ვეძებთ წრფივი რეგრესიის გამოყენებით ისეთ $y = \varepsilon + bx$ წრფეს, რომელშიც x დაკვირვებული მნიშვნელობების შეტანის შემდეგ, y-ის შესაბამისი მნიშვნელობებიდან გადახრათა კვადრატების ჯამი იქნება მინიმალური:

$$\sum_{i=1}^{n}(y_i - \varepsilon - bx_i)^2 \rightarrow min$$

- *ლოგისტიკური რეგრესია ან ლოგისტიკური მოდელი (Logistic regression - Logit model)* – არის სტატისტიკური მოდელი, რომელიც გამოიყენება მოვლენის ალბათობის პროგნოზირებისთვის, მისი შედარებით ლოგისტიკურ მრუდთან [108]. ეს რეგრესია იძლევა პასუხს ბინარული მოვლენის ალბათობის სახით (1 ან 0).

y=1 ხდომილობის განხორციელების ალბათობა გამოითვლება ფორმულით:

$$p = \frac{1}{1 + e^z}$$

სადაც

$$z = b_1x_1 + b_2x_2 + \ldots + b_nx_n$$

ლოგისტიკური რეგრესიის ამოცანა სწორედ ამ b_i -ური კოეფიციენტების გამოთვლაა.

- *გადაწყვეტილების ხე (Decision tree - DT)* და *გადაწყვეტილების ხის სწავლება (Decision tree learning)* – არის პროგნოზული მოდელირების ერთ-ერთი მიდგომა, რომლის გამოყენების სფეროებია: სტატისტიკა, მონაცემთა მოპოვება (ან ინტელექტუალური ანალიზი) და მანქანური დასწავლა. ეს

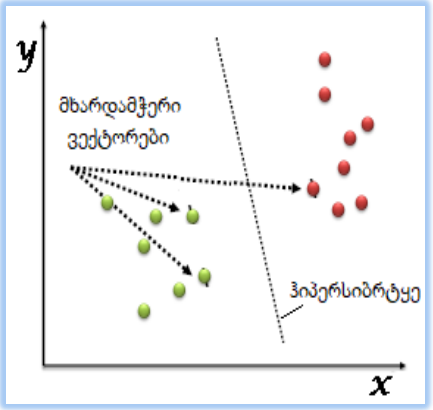
მიდგომა იყენებს გადაწყვეტილების ხეს როგორც პროგნოზირების მოდელს, რათა ელემენტზე დაკვირვებებიდან (ასახული ხის შტოებში) გადავიდეს დასკვნებამდე ელემენტის მიზნობრივი მნიშვნელობის შესახებ (ასახული ხის ფოთლებში). ხისმაგვარ მოდელებს, რომლებშიც მიზნობრივ ცვლადს შეუძლია მიიღოს მნიშვნელობათა *დისკრეტული* ერთობლიობა, უწოდებენ *კლასიფიკაციის ხეებს*. ამ ხისმაგვარ სტრუქტურებში ფოთლები არის კლასის ჭდეები (class labels), ხოლო შტოები – ფუნქციათა მახასიათებლების ერთობლიობა, რომლებსაც მივყავართ ამ კლასის ჭდეებთან. გადაწყვეტილების ხეებს, რომლებშიც მიზნობრივ ცვლადს შეუძლია მიიღოს უწყვეტი მნიშვნელობები (ნამდვილი რიცხვები), ეწოდება *რეგრესიული ხეები*.

ალტერნატიულ გადაწყვეტილებათა ანალიზის პროცესში გადაწყვეტილების ხე გამოიყენება როგორც ვიზუალური და ცხადი ინსტრუმენტული საშუალება საბოლოო შედეგის (გადაწყვეტილების) მისაღებად.

ხე შეიძლება „განსწავლილ" იქნას სიმრავლის დაყოფით ქვესიმრავლეებად, შესაბამის ატრიბუტთა მნიშვნელობების შემოწმების საფუძველზე. ეს პროცესი მეორდება რეკურსიულად თითოეული ქვესიმრავლისთვის და მას ეწოდება რეკურსიული სექტორი (დანაყოფი). რეკურსია წყდება მაშინ, როდესაც ქვესიმრავლეს ხის კვანძში აქვს მიზნობრივი ცვლადის იგივე მნიშვნელობა (აღარ ემატება პროგნოზის ახალი მნიშვნელობა). ასეთი ინდუქციური top-down მეთოდი ხშირად გამოიყენება მონაცემთა გადაწყვეტილების ხეების სწავლების პროცესში [108].

- *მხარდამჭერი ვექტორული მანქანა (Support Vector Machine – SVM)* – არის მანქანური დასწავლის მართვადი ალგორითმი, რომელიც შეიძლება გამოყენებულ იქნას როგორც კლასიფიკაციის (უმეტესად), ასევე რეგრესიული ამოცანებისთვის. SVM ალგორითმში მონაცემთა თითოეული ელემენტი აიგება როგორც წერტილი n-განზომილებიან სივრცეში (სადაც n მახასიათებლების რაოდენობაა). ამასთანავე, თითოეული მახასიათებლის მნიშვნელობა არის კონკრეტული კოორდინატის მნიშვნელობა. შემდეგ, სრულდება კლასიფიკაცია იმ ჰიპერ-სიბრტყის პოვნით, რომელიც კარგად განასხვავებს ორ კლასს (ნახ.3.6) [110].

მხარდამჭერი ვექტორები – ინდივიდუალური დაკვირვების კოორდინატებია. SVM კლასიფიკატორი არის საზღვარი, რომელიც უკეთესად გამოყოფს ორ კლასს (ჰიპერ–სიბრტყე /ხაზი).



ნახ.3.6. მხარდამჭერი ვექტორები

მანქანური დასწავლის ალგორითმების განსახორციელებლად ფართოდ გამოიყენება Python ენაში scikit-Learn ბიბლიოთეკა. SVM ალგორითმისათვისაც ხელმისაწვდომია ეს ბიბლიოთეკა [111].

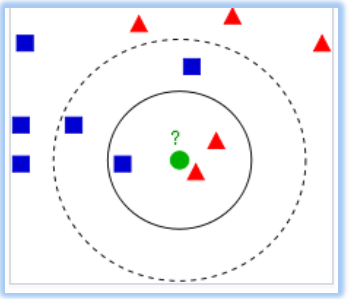
- ბაიესის დასკვნა (Bayesian inference - BI) – არის სტატისტიკური დასკვნის მეთოდი, რომლის დროსაც ბაიესის თეორემა გამოიყენება ჰიპოთეზის ალბათობის განახლების მიზნით, როდესაც მეტი მტკიცებულება ან ინფორმაციაა მიღებული. ბაიესის დასკვნის მნიშვნელოვანი ნაწილია პარამეტრების და მოდელების დადგენა [112, 113].

ბაიესის ოპტიმიზაციის საკითხი პროგრამული კოდებისთვის მეტად მნიშვნელოვანია, როგორც დეველოპმენტის, ასევე ტესტირების პროცესში [114]. ამ ნაშრომში ნაჩვენებია, მაგალითად, თუ როგორ გამოვიყენოთ ბაიესის მეთოდი რთული ოპტიმიზაციის პრობლემების კვლევაში. კერძოდ, გლობალური ოპტიმიზაციაში, როგორც რთულ ამოცანაში, რომელიც მოიცავს შავ ყუთს და ხშირად არაწრფივ, ხმაურიან და ძვირი გამოთვლების მიზნობრივ ფუნქციებს – შესასრულებლად. ბაიესის ოპტიმიზაცია უზრუნველყოფს გლობალური ოპტიმიზაციის ალბათურ-პრინციპულ მეთოდს. როგორ განვახორციელოთ ბაიესის ოპტიმიზაცია და როგორ გამოვიყენოთ იგი ღია პროგრამული კოდისთვის.

- *K-უახლოესი მეზობელი (K-nearest neighbor KNN)* - არის მეტრიკული ალგორითმი ობიექტების ავტომატური კლასიფიკაციის ან რეგრესიისათვის.

კლასიფიკაციის მეთოდის გამოყენების შემთხვევაში, ობიექტი მიეკუთვნება იმ კლასს, რომელიც ყველაზე გავრცელებულია ამ ელემენტის K მეზობლებს შორის, რომელთა კლასები უკვე ცნობილია (ნახ.3.7).

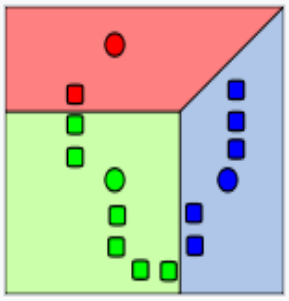
რეგრესიის მეთოდის გამოყენების შემთხვევაში, ობიექტს ენიჭება საშუალო მნიშვნელობა K უახლოეს ობიექტებზე, რომელთა მნიშვნელობები უკვე ცნობილია.



ნახ.3.7. K-უახლოესი მაზობელი

ალგორითმის გამოყენება შესაძლებელია დიდი რაოდენობით ატრიბუტების მქონე ნიმუშების (მრავალგანზომილებიან) ნიმუშებზე. ამისათვის, გამოყენებამდე უნდა დადგინდეს მანძილის ფუნქცია; ასეთი ფუნქციის კლასიკური ვერსია ევკლიდეს მეტრიკაა - ესაა მანძილი ორ წერტილს შორის ევკლიდეს სივრცეში, რომელიც გაითვლება პითაგორას თეორემით.

- *k- საშუალოების კლასტერიზაცია (k-means clustering)* – არის ვექტორული კვანტირების (შეკუმშვის) მეთოდი, რომელიც გამოიყენება მონაცემთა მოპოვების (Data Mining) კლასტერულ ანალიზში (მაგალითად, მონაცემთა ბაზაში „მსგავსი“ სტრუქტურების აღმოჩენის მეთოდი) [115]. k-საშუალოების კლასტერიზაციის მიზანია n დაკვირვებების დაყოფა k კლასტერად, რომლებშიც თითოეული დაკვირვება მიეკუთვნება კლასტერს უახლოესი საშუალო მნიშვნელობით და წარმოადგენს კლასტერის პროტოტიპს. შედეგად მონაცემთა სივრცე იყოფა ვორონოის უჯრედებად (Voronoi cells) (ნახ.3.8). k-საშუალოებს მინიმუმამდე დაყავს



ნახ.3.8. k-საშუალოები კლასტერიზაცია

დისპერსიები კლასტერის შიგნით (ევკლიდური მანძილის კვადრატი). საშუალო მნიშვნელობა ახდენს კვადრატული შეცდომების ოპტიმიზაციას, ხოლო გეომეტრიული მედიანა ამცირებს ევკლიდურ მანძილებს.

k-საშუალოების კლასტერიზაციის ალგორითმი პროგრამულად რეალიზებულია სხვადასხვა ენაზე, მათ შორის არის Free/Open Source Software ვერსიებიც [115]. მაგალითად, Accord.NET ფრეიმვორკი, რომელშიც k-means და სხვა ვერსიები რეალიზებულია C# ენაზე. ისინი გამოიყენება სამეცნიერო გამოთვლებისათვის .NET-ში [115].

პროგრამები ძირითადად განსხვავდება მწარმოებლურობით, (შესრულების სწრაფქმედებით), მაგალითად, 10 წამიდან – რამდენიმე საათამდე დიაპაზონში [117]. განსხვავება არის, რა თქმა უნდა, შესრულების ხარისხშიც და შედეგების სიზუსტეშიც.

- *შემთხვევითი ტყე (Random forest)* – არის მანქანური დასწავლის მართვადი ალგორითმი, რომელიც გამოიყენება როგორც კლასიფიკაციის, ასევე რეგრესიისთვის. ანალოგიურად, ეს ალგორითმი ქმნის გადაწყვეტილების ხეებს მონაცემთა სახეების ამოსარჩევად, შემდეგ კი თითოეული მათგანისგან იღებს პროგნოზს და, საბოლოოდ, კენჭისყრით ირჩევს საუკეთესო შედეგს [118].

შემთხვევითი ტყის ალგორითმის მუშაობა ხორციელდება, ზოგადად, შემდეგი ბიჯებით:

1-ბიჯი: შემთხვევითი ნიმუშების შერჩევის დაწყება მოცემულ მონაცემთა ერთობლიობიდან;

2-ბიჯი: ეს ალგორითმი ააგებს გადაწყვეტილების ხეს ყველა ნიმუშისათვის. შემდეგ ის მიიღებს პროგნოზირების შედეგს ყველა გადაწყვეტილების ხისგან;

3-ბიჯი: ხმის მიცემა განხორციელდება ყველა პროგნოზირებული შედეგისთვის;

4-ბიჯი: აირჩევა ყველაზე მეტი ხმით პროგნოზირებული შედეგი, როგორც საბოლოო.

➢ *უპირატესობები:*

– დიდი რაოდენობის მახასიათებლებისა და კლასების მქონე მონაცემთა ეფექტურად დამუშავების უნარი;

– მახასიათებლების (ატრიბუტის) მნიშვნელობათა მასშტაბირებისადმი (და სხვა გარდაქმნებისადმი) არამგრძნობიარობა;

– დისკრეტული და უწყვეტი მახასიათებლების თანაბრად კარგი დამუშავების უნარი;

– მოდელში ცალკეული მახასიათებლის მნიშვნელობის შეფასების მეთოდების არსებობა;

– მოდელის შესაძლებლობის შინაგანი შეფასების უნარი განზოგადებისადმი;

– მაღალი პარალელიზმი და მასშტაბურობა.

➢ *ნაკლოვანებები:*

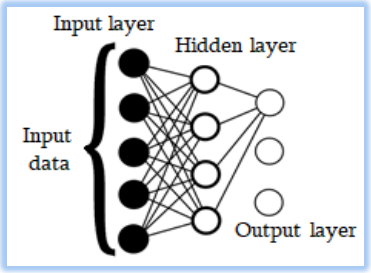
– სირთულე არის „შემთხვევითი ტყის" ალგორითმების მთავარი მინუსი;

– შემთხვევითი ტყეების აგება გაცილებით რთული და შრომატევადია, ვიდრე გადაწყვეტილების ხეებისა;

– შემთხვევითი ტყის ალგორითმის რეალიზაციისთვის საჭიროა მეტი გამოთვლითი რესურსი;

– პროგნოზირების პროცესი შემთხვევითი ტყეების გამოყენებით გაცილებით შრომატევადია სხვა ალგორითმებთან შედარებით.

- *ხელოვნური ნეირონული ქსელები (Artificial Neural Networks ANN)* – მანქანური დასწავლის მეთოდია, ხელოვნური ინტელექტის – AI) ერთ-ერთი პროგრამა. მანქანური დასწავლის ალგორითმებს შეუძლია ცხადად გაუმჯობესება, დაპროგრამების გარეშე. სხვა სიტყვებით რომ ვთქვათ, მათ შეუძლია კანონზომიერებათა პოვნა მონაცემებში და გამოიყენონ ისინი მომავალში ახალი ამოცანების გადასაწყვეტად [119]. ANN ქსელი მარტივი პროცესორების (ხელოვნური ნეირონების) დაკავშირებული და ერთმანეთთან ურთიერთმოქმედი სისტემაა (ნახ.3.9). ასეთი ქსელის თითოეული პროცესორი მხოლოდ იმ სიგნალებს ეხმიანება, რომლებსაც იგი პერიოდულად იღებს და რომლებსაც პერიოდულად უგზავნის სხვა პროცესორებს. ამ დიდ ქსელში, რომელსაც აქვს მართვადი ურთიერთქმედება, ეს ინდივიდუალურად მარტივი პროცესორები ერთად ახერხებენ საკმაოდ რთული დავალებების შესრულებას.



ნახ.3.9. მარტივი ნეირონული ქსელის მოდელი

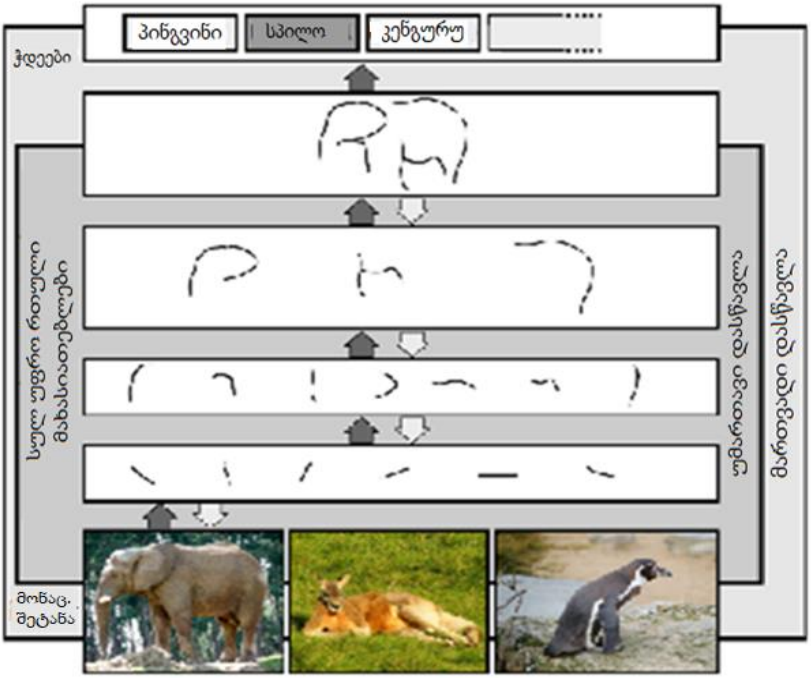
დასწავლის ალგორითმების შემუშავების შემდეგ შესაძლებელია მიღებული მოდელების პრაქტიკული მიზნებისთვის გამოყენება: პროგნოზირების, სახეთა ამოცნობის, მართვის და სხვა სახის ამოცანებში.

ხელოვნური ნეირონული ქსელის ალგორითმები შთაგონებულია ადამიანის ტვინის მიერ. ხელოვნური ნეირონები ურთიერთდაკავშირებულია და ერთმანეთთან ურთიერთობს [120]. თითოეული კავშირი შეწონილია წინა დასწავლის მოვლენებთან და მონაცემთა ყოველი ახალი შეყვანის შედეგად ხდება მეტი განსწავლა. მრავალი განსხვავებული ალგორითმი ასოცირდება ხელოვნურ ნეირონულ ქსელთან და ერთ-ერთი ყველაზე მნიშვნელოვანია „ღრმა დასწავლა' (Deep Learning). ეს განსაკუთრებით ეხება ბევრად უფრო რთული რთული ნეირონული ქსელების აგებას.

- *ღრმა სწავლება (Deep Learning – DL)* - არის მანქანური დასწავლის მეთოდების ფართო ოჯახის ნაწილი, რომელიც ბაზირებულია ხელოვნურ ნეირონულ ქსელებზე, წარმოდგენის დასწავლით. დასწავლა შეიძლება იყოს მართვადი, ნახევრად-მართვადი ან უმართავი [121].

მთავარი განსხვავება მანქანურ დასწავლასა და ღრმა დასწავლას შორის მდგომარეობს სისტემაში მონაცემთა ასახვის მეთოდებსა და ხერხებში. მანქანური დასწავლის ალგორითმები თითქმის ყოველთვის მოითხოვს სტრუქტურირებულ მონაცემებს, ხოლო ღრმა დასწავლის ქსელები კი ეყრდნობა ხელოვნურ ნეირონულ ქსელებს (ANN - artificial neural networks).

ღრმა დასწავლა, როგორც მანქანური დასწავლის ალგორითმების კლასი, იყენებს რამდენიმე შრეს, რათა საწყისი მონაცემებიდან (ქვედა დონე) თანდათანობით ამოიღოს ზედა დონისთვის მახასიათებლები (წიბოები) [122,123] (ნახ.3.10).



ნახ.3.10. იერარქიული ობიექტების ფენოვანი დასწავლის სქემატური მიმოხილვა [123]

მაგალითად, გამოსახულებების დამუშავებისას ქვედა შრეებს შეუძლია წიბოების იდენტიფიცირება, ხოლო ზედა შრეებს – ადამიანისთვის გასაგები ცნებების იდენტიფიცირება, როგორიცაა რიცხვები, ასოები და სახეები [123].

3.7. მომხმარებლის ინტერფეისის პროგრამის ოპტიმიზაციის მეთოდი React-ის ბაზაზე

React (ან ReactJS) არის JavaScript-ის ბიბლიოთეკა ღია საწყისი კოდით, რომლითაც ეფექტურად იქმნება მომხმარებლის ინტერფეისები, ერთგვერდიანი აპლიკაციებისათვის [124, 125]. იგი, როგორც ვებ-ფრეიმვორკი, გამოიყენება ვებსაიტის და მობილური აპლიკაციების წარმოდგენის (View) შრის დასამუშავებლად. React გვაძლევს ასევე საშუალებას შეიქმნას სამომხმარებლო ინტერფეისის (UI – User Interface) მრავალჯერადად გამოყენებადი (reusable) კომპონენტები.

ამ პარაგრაფში გაანალიზებულია React-ის თანამედროვე ვებ-ტექნოლოგიები front-end და back-end აპლიკაციების აგების თვალსაზრისით. განსაკუთრებული ყურადღება გამახვილებულია JavaScript ენის საფუძველზე შექმნილი ინსტრუმენტებისა და ახალი ტექნოლოგიების ინტეგრაციის საკითხებზე. კერძოდ, Angular, NodeJS, Ajax, ReactJS და სხვა ფრეიმვორკები და ბიბლიოთეკები [77]. მათ საფუძველზე მნიშვნელოვნად გაუმჯობესდა front-end და back-end ვებაპლიკაციების როგორც *ხარისხობრივი*, ასევე *მწარმოებლურობის (სწრაფქმედების) მახასიათებლები*.

ვებ-ფრეიმვორკები შექმნილია იმისათვის, რომ მომხმარებლებმა შეძლონ ვებ–აპლიკაციების სწრაფად აგება. გარდა ამისა, ისინი გვთავაზობს მონაცემთა ბაზასთან წვდომას, Pattern მექანიზმებს, პრეზენტაციისა და კოდის სუფთა განცალკევებას Model–View–Controller-ის (MVC) ან Model–View–Presenter-ის (MVP) გამოყენებით, როგორც არქიტექტურული ნიმუში [126, 127],

3.7.1. რეაქტიული პროგრამირების არსი და MVC არქიტექტურა

რეაქტიული პროგრამირება (Reactive Programming) არის დეკლარაციული პროგრამირების პარადიგმა, მომხმარებელთა ინტერფეისების, ანიმაციების ან სიმულაციის (იმიტაციური მოდელირების) სისტემების შექმნის მოქნილი კონცეფციის მქონე ინსტრუმენტული საშუალება. მისი არსი მდგომარეობს იმაში, რომ უნდა არსებობდეს მონაცემთა სტატიკური და დინამიკური ნაკადების ადვილად გამოსახვის შესაძლებლობა, აგრეთვე შესრულების მოდელის მიერ ცვლილებების ავტომატური გავრცელება მონაცემთა ნაკადების საფუძველზე. ანუ, *შედეგი ყოველთვის ხელახლა განისაზღვრება ავტომატურად, როცა მონაცემთა საწყისი ნაკადები იცვლება* [77].

რეაქტიული პროგრამირების საილუსტრაციოდ განიხილავენ თანამედროვე ცხრილურ პროცესორებს (Ms Excel), სადაც (მაგალითად, A20) უჯრას აქვს სტრიქონული მნიშვნელობა ფორმულის სახით: " = B5+C17". მისი მნიშვნელობა დამოკიდებულია B5 და C17 უჯრების მნიშვნელობებზე. თუ შეიცვალა ამ ორი უჯრიდან ერთ-ერთის ან ორივეს მნიშვნელობა, *მაშინვე ავტომატურად იცვლება* A20 უჯრის მნიშვნელობაც.

რეაქტიული პროგრამირების სხვა კლასიკური მაგალითია MVC არქიტექტურა (Model-View-Controller), სადაც შესაძლებელია ავტომატურად აისახოს ცვლილებები Model-იდან View-ში და პირიქით, View-დან Model-ში, Controller-ის დახმარებით.

ახლა განვიხილოთ MVC არქიტექტურის ზოგიერთი საკითხი უფრო დეტალურად.

MVC აპლიკაციებში ურთიერთქმედება გულისხმობს ნატურალურ ციკლს მომხმარებლის მოქმედებებსა (use actions) და view-პრეზენტაციის დონის განახლებებს შორის. View ლოგიკურ შრეზე მდგომარეობის (stateless) შენახვა არ ხდება, რაც კარგად შეესაბამება ვებ-აპლიკაციების HTTP პროტოკოლზე Request და Response გზავნილებით მუშაობის პრინციპს [38].

MVC არქიტექტურა სავალდებულოს ხდის აპლიკაციის ლოგიკურ დანაწილებას (Separation of Concerns) – დომეინ მოდელი და კონტროლერის მოდელი გამიჯნულია პრეზენტაციის დონიდან. ეს ნიშნავს, რომ HTML კოდი განცალკევებულია დანარჩენი პროგრამული კოდისგან. შესაბამისად, ტესტირება და აპლიკაციის მხარდაჭერა გაიოლებულია.

ფუნქციონალის ლოგიკური დაყოფის მიდგომით MVC ტექნოლოგია განსხვავდება სტანდარტული ვებ-ფორმების ტექნოლოგიისგან (Web Forms), სადაც მომხმარებლის მიერ შეტანილი მონაცემები (user input) გადაეწოდება ვებ-გვერდს (View) და უშუალოდ ვებ-ფორმა არის პასუხისმგებელი შეტანილი ინფორმაციის გაადამუშავებასა და პასუხის დაბრუნებაზე.

MVC არქიტექტუის მქონე აპლიკაციის ფუნქციონალი გადანაწილებულია 3 ძირითად კომპონენტში, რომლებიც აპლიკაციას შემდეგ ლოგიკურ დონეებად ყოფს: *მოდელი (Model), პრეზენტაციის დონე (View) და კონტროლერი (Controller).* განვიხილოთ ეს დონეები ცალ-ცალკე.

- **Model (მოდელი)**

მოდელი არის აპლიკაციის მონაცემთა დომეინი, ანუ მონაცემთა სიმრავლე, რომელთანაც მუშაობს მომხმარებელი.

დომეინ მოდელი შედგება ბიზნეს დომეინის მონაცემთა სიმრავლის, ოპერაციების, მონაცემთა ტრანსფორმირებისა და მართვის წესებისგან. მოდელის ფუნქციებში არ შედის გრაფიკული ელემენტების გამოტანა ან request-მოთხოვნების დამუშავება (view და controller დონეების ფუნქცია). მოდელი წარმოადგენს რეალური სამყაროს ობიექტების, წესებისა და ოპერაციების ასახვას პროგრამულ უზრუნველყოფაში. ASP.NET Framework-ის ენაზე მოდელი გულისხმობს [38]:

- C# ტიპების (Classes, Structs, Enums და სხვა) ერთობლიობას, რაც ცნობილია როგორც დომეინის ტიპი (Domain Type);
- დომეინში არსებულ ობიექტებზე შესაძლო ოპერაციები აღწერილია დომეინის ტიპის მეთოდებიში (Domain Method);
- დომეინის წესები იმპლემენტირებულია დომეინის მეთოდების შიგნით.

მოდელი არის აპლიკაციის შიგნით არსებული Entity-ერთეულებისა და ბიზნეს წესების ერთობლიობა. დომეინის შემუშავების მიდგომა ბევრ პრობლემურ საკითხს ჭრის. თუ საჭირო გახდა მონაცემის რაიმე ბიზნეს ერთეულის, პროცესის ან წესის შეცვლა – დომეინ ლოგიკური დონე წარმოადგენს იმ ერთადერთ ადგილს, სადაც აპლიკაციის კოდში ცვლილება მოხდება.

ზოგადი პრაქტიკით მიღებულია დომეინ მოდელის ცალკე ბიბლიოთეკად (C# Assembly) გატანა და აპლიკაციის სხვა პროექტებში მისი ჩამატება Reference-სახით (ლოგიკური შრეების ურთიერთ-გამიჯვნის პრონციპი).

- **View (მიმოხილვა)**

პრეზენტაციის შრე არის მომხმარებლის ინტერფეისის (User Interface – UI) იმპლემენტაცია. ამ კომპონენტის ფუნქციაა მონაცემთა ასახვა, გამოტანა მომხმარებლისთვის, ზოგადად view არის შაბლონი, რომელიც უზრუნველყოფს Runtime რეჟიმში საბოლოო HTML-ის გენერაციას [128].

- **Controller (კონტროლერი)**

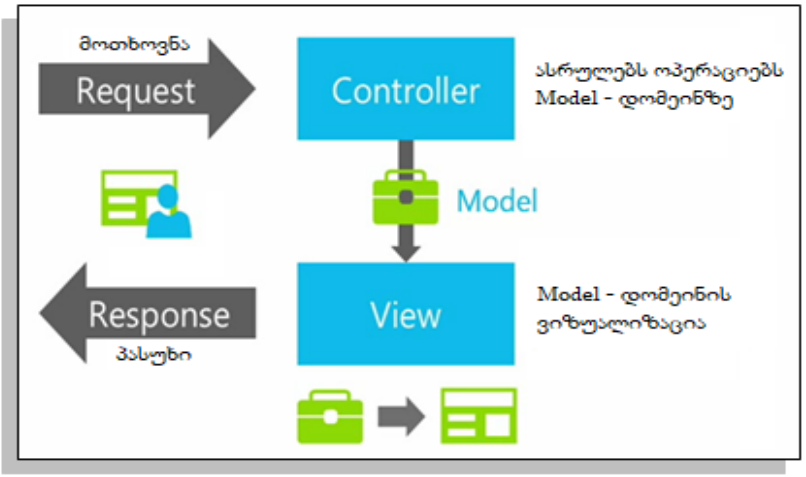
კონტროლერის დანიშნულება არის მომხმარებლის მიერ შეტანილი ინფორმაციის დამუშავება (Input Control), მომხმარებლის ინტერაქტიულობით უზრუნველყოფა და დაკავშირება სისტემასთან (View - კომპონენტის დაკავშირება Model-კომპონენტთან).

კონტროლერი კითხულობს მონაცემებს ფორმიდან, ამოწმებს შეტანილი მონაცემების ვალიდურობას და უგზავნის მათ model-კომპონენტს. Controler-ობიექტი წყვეტს რომელი view-კომპონენტი ასახოს მომხმარებლის მიერ გადაცემული ინფორმაციის საპასუხოდ [129].

MVC არქიტექტურაში Controller-კლასები მემკვიდრეობით მოდის System.Web.Mvc.Controller კლასიდან. Controller-კლასის თითოეული ღია მეთდი (public method) არის ე.წ. Action-მეთოდი, რომელიც მარშრუტიზაციის

სისტემით (ASP.NET Routing System) ასოცირებულია კონფიგურირებად URL-მისამართთან.

როდესაც Request-მოთხოვნა URL-მისამართით მიაკითხავს მასთან ასოციერებულ Action-მეთოდს, Controller-კლასში დაიწყება ოპერაციების ჩატარება დომეინ მოდელზე (Domain Model), რის შემდეგაც კონტროლერის Action-მეთოდი ჩატვირთავს HTML View-ობექტს და დაუბრუნებს Response-პასუხს. 3.11 ნახაზზე ასახულია ეს პროცესები.



ნახ.3.11. MVC პროცესები

ამგავარად, MVC მოდელით აპლიკაციის ლოგიკის ურთიერთგამიჯვნა ამარტივებს დასმული ამოცანის სირთულის მართვას, რადგან შესაძლებლობას იძლევა ერთ კონკრეტულ ასპექტზე გაკეთდეს ფოკუსირება. მაგალითად, მუშაობა შეიძლება View-კომპონენტზე ბიზნეს ლოგიკისგან დამოუკიდებლად. ეს ამარტივებს აპლიკაციის *ტესტირების* პროცესს.

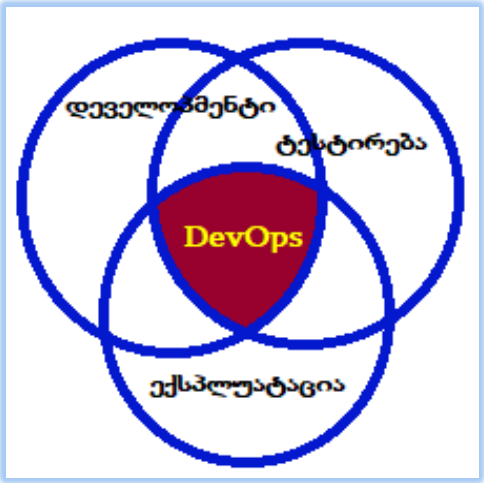
გამარტივებულია პროგრამისტთა გუნდის მუშაობა: სხვადასხვა წევრს აქვს პარალელურად მუშაობის შესაძლებლობა; View-ლოგიკა, ბიზნეს ლოგიკა და კონტროლერის ლოგიკა შეიძლება იწერებოდეს დამოუკიდებლად, სხვადასხვა დეველოპერის მიერ.

წინამდებარე პარაგრაფის საფუძველზე შეიძლება გავაკეთოთ დასკვნა, რომ MVC (Model–View–Controller – სამეულით) არის მონოლითური პროგრამული აპლიკაციის სტრუქტურული მიდგომა. *Model* შეიცავს სისტემის ძირითადი *ბიზნეს-ლოგიკას*, *View* – მხოლოდ იმ ნაწილებს, რომლებიც საჭიროა *მომხმარებლის ინტერფეისის* მუშაობისთვის, ხოლო *Controller* – მოთავსებული მათ შორის, *მართავს მოდელს* მომხმარებლის მოთხოვნილებების შესაბამისად.

ამ ტიპის სტრუქტურის მთავარი იდეაა ის, რომ ის ინარჩუნებს დამოუკიდებლობას (სუსტ კავშირს) კოდის ცალკეულ ნაწილებს შორის. თეორიულად, ეს აადვილებს ახალი ინტერფეისების დამატებას საჭოროების შემთხვევაში ან მოდელის შეცვლას ინტერფეისის შეცვლის გარეშე. პრაქტიკაში ეს მარტივად არ ხდება. MVC რამდენიმე წლის განმავლობაში იყო პოპულარული პროგრამული სისტემების მონოლითურ არქიტექტურასთან ერთად. ბოლო პერიოდში უფრო ვითარდება და მეტი ყურადღება ექცევა *მიკროსერვისების* გამოყენებას [130,131].

მიკროსერვისული არქიტექტურა (Microservice) არის პროგრამული სისტემების სერვისზე ორიენტირებული არქიტექტურის (SOA) ვარიანტი, მისი მიზნია მაქსიმალურად მცირე ზომის, სუსტად დაკავშირებული, ადვილად

მოდიფიცირებადი მოდულების – მიკროსერვისების ურთიერთქმედების ორგანიზება. იგი ვითარდება 2010-იანი წლებიდან, განსაკუთრებით Agile-დეველოპმენტის და DevOps-ის განვითარების შედეგად (ნახ.3.12) [132, 133].



ნახ.3.12. DevOps მეთოდოლოგია

3.7.2. DevOps მეთოდოლოგია და ინსტრუმენტული საშუალებები

Agile და DevOps მეთოდოლოგიები ასრულებს ურთიერთდამატებით როლებს. მაგალითად, პროგრამული სისტემების ავტომატიზებული კონსტრუირება და ტესტირება, უწყვეტი ინტეგრაცია და უწყვეტი მიწოდება [10, 134, 135]. 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.

DevOps მეთოდოლოგია აერთიანებს პროგრამული უზრუნველყოფის დეველოპმენტის (Dev) და ინფორმაციული ტექნოლოგიების ოპერაციებს (Ops) (ნახ.3.12). მისი მიზანია პროგრამული სისტემების შექმნის ციკლის დროის შემცირება და მაღალი ხარისხის პროგრამული უზრუნველყოფის უწყვეტი მიწოდება [133]. 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 [133]:

1. Coding – კოდის დეველოპმენტი და ანალიზი, საწყისი კოდის მენეჯმენტის (Source Code Management) ინსტრუმენტი, კოდების შერწყმა [136];

2. Building – უწყვეტი ინტეგრაციის (Continuous Integration) ინსტრუმენტი, კონსტრუირების სტატუსი [137];

3. Testing – უწყვეტი ტესტირების (Continuous Testoing) ინსტრუმენტი, რომელიც უზრუნველყოფს სწრაფ და დროულ უკუკავშირს ბიზნეს რისკების მიხედვით [138];

4. Packaging – არტეფაქტების რეპოზიტორია (მონაცეთა საცავი - data warehouse), აპლიკაციის წინასწარი განთავსება. საცავში ინახება პროგრამული პაკეტები და მათი მეტა-მონაცემები, ცვლილებათა ჟურნალი სისტემის მენეჯერი-სათვის [139];

5. Releasing – ცვლილებათა მენეჯმენტი, რელიზის (ვერ-სიის) გამოცემის დამტკიცება, აპლიკაციის ვერსიის ავტო-მატიზაცია Application-release automation (ARA) [140];

6. Configuring – ინფრასტრუქტურის კონფიგურაცია და მენეჯმენტი, ინფრასტრუქტურა, როგორც კოდის ინსტრუ-მენტი (Infrastructure as code – IaC) [141];

7. მონიტორინგი – აპლიკაციების მწარმოებლურობის (სწრაფქმედების) მონიტორინგი, მუშაობის გამოცდილება საბოლოო მომხმარებელთან (Application Performance Mana-gement – APM). APM ცდილობს კომპლექსური პროგრამების შესრულების პრობლემების გამოვლენას (იხ. პარაგრაფი 2.3 და 3.2.1 – პროფაილერი). პროგრამული სისტემის მწარმოე-ბლურობის ოპტიმიზაციის საკითხს ჩვენ კიდევ დავუბ-რუნდებით მომდევნო პარაგრაფში – React Hook-ის საფუძველზე.

8. და სხვ.

3.7.3. React-ის მახასიათებლები, კომპონენტები და მუშაობის პრინციპები

➢ **როგორ მუშაობს React ?** React-ის კომპონენტი – ესაა კოდის ფრაგმენტი, რომელიც ასახავს გვერდის ნაწილს. ყოველი კომპონენტი JavaScript-ის ფუნქციაა, რომელიც აბრუნებს კოდის ფრაგმენტს, რაც შეესაბამება ვებ-გვერდის ფრაგმენტს. React იყენებს JSX ენას, რომელიც HTML-ის მსგავსია, ოღონდაც იგი მუშაობს JavaScript-ის შიგნით (HTML არა) [77].

➢ **React Native** – არის JavaScript-ის ფრეიმვორკი საკუთარი მობილური აპლიკაციების შესაქმნელად. იგი იყენებს React პლატფორმას და იძლევა მრავალ ჩაშენებულ კომპონენტს და აპლიკაციის პროგრამულ ინტერფეისს (API - application programming interface). React-ით იგი ვებ-გვერდზე პირდაპირ ტრანსლირებას უკეთებს **DOM** ბრაუზერს. React Native-სთვის mark-up (მონიშვნა, ფორმატირება) ტრანსლირდება ხოსტის პლატფორმის შესაბამისად Android-ისთვის დამახასიათებელი სპეციფიკაციით.

React Native, ფაქტობრივად, მუშაობს JavaScript ფაილების ჩასაშენებლად აპლიკაციაში და ლოკალური შესრულებისათვის. *React Native API* – არის ნამდვილი მობილური აპლიკაცია. React Native არ ქმნის მობილურ ვებ-აპლიკაციებს, რადგანაც იგი იყენებს მომხმარებლის ინტერფეისის იმ ძირითად სამშენებლო ბლოკებს, რომელსაც იყენებს ჩვეულებრივი აპლიკაციები iOS-ის და Android-სთვის. ამ სამშენებლო ბლოკების შესაერთებლად აქ გამოიყენება JavaScript და React, ნაცვლად Swift, Kotlin ან Java ენებისა [142].

- **DOM და VDOM ობიექტები React-ში.**

Document Object Model (DOM) არის HTML და XML დოკუმენტების პროგრამირების ინტერფეისი. იგი ასახავს გვერდს ისე, რომ პროგრამებს შეუძლია შეცვალოს მათი სტრუქტურა, სტილი და შინაარსი. DOM ასახავს დოკუმენტს კვანძებისა და ობიექტების სახით. აქ შესაძლებელია პროგრამული ენების მიერთება გვერდთან.

ვებ-გვერდი არის დოკუმენტი, რომელიც შეიძლება ასახული იყოს ბრაუზერის ფანჯარაში ან როგორც HTML-ის წყარო. ორივე ერთიდაიგივეა. DOM არის ვებ-გვერდის ობიექტ-ორიენტირებული წარმოდგენა, რომლის ცვლილება შესაძლებელია სცენარების ენით, მაგალითად JavaScript-ით.

DOM არაა პროგრამირების ენა, მაგრამ მის გარეშე JavaScript-ს არ ექნებოდა არავითარი მოდელი ან ცნებები ვებ-გვერდის, HTML და XML დოკუმენტების და მათი შემადგენელი ელემენტების შესახებ. დოკუმენტის თითოეული ელემენტი, მთლიანი დოკუმენტი, სათაურები, ცხრილები დოკუმენტში, ტექსტები ცხრილის უჯრებში – არის დოკუმენტის ობიექტური მოდელის ნაწილი. ამიტომ მათთან მიმართვა და მანიპულაციების ჩატარება შესაძლებელია DOM-ის და JavaScript-ის საშუალებით.

Virtual Document Object Model (VDOM) – არის JavaScript-ის მსუბუქი ობიექტი, რომელიც რეალური DOM-ის უბრალო ასლია. ესაა კვანძების ხე, სია ელემენტების, მათი ატრიბუტების და შედგენილობისა, როგორც ობიექტებისა და თვისებების. React-ის რენდერის ფუნქცია აგებს ამ ხეს. DOM-ის

ცვლილება ყოველთვის იწვევს მისი შესაბამისი VDOM-ის ცვლილებას.

- **React Fiber და React360**

ფეისბუკმა 2017 წ. გამოუშვა React Fiber – ახალი front-end ალგორითმი React Framework ბიბლიოთეკისთვის. იგი მომავალში უნდა გახდეს React-ის ინფრასტრუქტურის ფუნქციების დამუშავების ინსტრუმენტი {126}. React 360 შემუშავდა ინფორმაციული საზოგადოების ფართო სპექტრისთვის და მისი გამოყენება ორიენტირებულია ვირტუალური რეალობის სისტემებისაკენ [143,144].

- **React-ის მახასიათებლები** [146]:

- **JSX** – ესაა JavaScript-ის გაფართოებული სინტაქსი. React-ში არაა აუცილებელი JSX გამოყენება (რეკომენდებულია);
- **Components** – React არის ყველაფერი კომპონენტების შესახებ. კომპონენტებით აზროვნება, განსაკუთრებით საყურადღებოა დიდი პროექტების დამუშავების დროს;
- **Unidirectional data flow and Flux** – მონაცემთა ერთმიმართულებიანი ნაკადი ხორციელდება React-ში, რაც აადვილებს აპლიკაციის ანალიზს. Flux არის შაბლონი (pattern), რომელიც გვეხმარება მონაცემთა ერთმიმართულებიანი სახით შესანახად;
- **License** – ლიცენზირება React-ისა ხდება Facebook Inc ფარგლებში. დოკუმენტაციისა კი - CC BY 4.0 -ის შესაბამისად.

❖ *React-ის უპირატესობები:*

✓ იყენებს ვირტუალურ DOM-ს, რომელიც JacaScript-ის ობიექტია. ეს ამაღლებს აპლიკაციის მწარმოებლურობას, რადგან VDOM მუშაობს უფრო სწრაფად, ვიდრე DOM;

✓ შეიძლება გამოყენებულ იქნას როგორც კლიენტის მხარეს (front-end), ასევე სერვერის მხარეს (back-end). აგრეთვე სხვა პლატფორმებთან;

✓ კომპონენტებისა და მონაცემთა შაბლონები (pattern) აუმჯობესებს კოდის კითხვადობას, რაც გვეხმარება დიდი აპლიკაციების მხარდაჭერაში.

❖ **React-ის შეზღუდვები:**

✓ ეხება მხოლოდ წარმოდგენის დონეს (View), ამიტომ საჭირო ხდება სხვა ტექნოლოგიების გამოყენებაც აპლიკაციის დამუშავების ინსტრუმენტების დასაკომპლექტებლად;

✓ იყენებს JSX-ის ჩადგმულ შაბლონებს, რაც შეიძლება არ იყოს მოხერხებული ზოგიერთი დეველოპერისათვის.

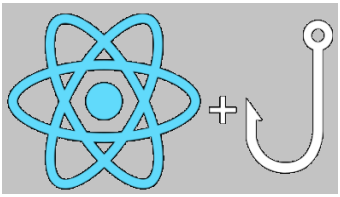
➢ **Redux_(JavaScript_library)**

არის JavaScript-ის ბიბლიოთეკა ღია საწყისი კოდით, რომელიც მართავს აპლიკაციის მდგომარეობას. იგი ხშირად გამოიყენება React- ან Angular-თან ერთად მომხმარებელთა ინტერფეისების ასაგებად [146]. იგი მსგავსია Flux Facebook არქიტექტურისა. შეიცავს ინსტრუმენტებს, რომლებიც განსაკუთრებით ამარტივებს საცავის მონცემების გადაცემას.

3.7.4. Hooks და Hooking ტექნოლოგია

პროგრამირებაში ტერმინი hooking („ჩაჭრა“, „перехват“) მოიცავს სხვადასხვა მეთოდს, რომლებიც გამოიყენება ოპერაციული სისტემის, აპლიკაციების ან სხვა პროგრამული კომპონენტების ქცევის შეცვლის ან დამატების მიზნით,

პროგრამული უზრუნველყოფის კომპონენტებს შორის გადაცემული ფუნქციების (შეტყობინებების ან მოვლენების) „დაჭერის" გზით. კოდი, რომელიც ასრულებს ასეთი დაჭერილი ფუნქციების (შეტყობინებების ან მოვლენების) დამუშავებას, hook-ს უწოდებენ („ანკესი“, „მახე“) (ნახ.3.13) [147, 148].



ნახ.3.13. ReactHooks: ლოგო

ამგვარად, hooking – ტექნოლოგიაა, რომელიც ცვლის ინფორმაციული სისტემის ამა თუ იმ კომპონენტის სტანდარტულ ყოფაქცევას.

Hook არის მექანიზმი, რომლის დახმარებითაც აპლიკაციას შეუძლია „დაიჭიროს“ ისეთი მოვლენები, როგორიცაა შეტყობინება, მაუსის მოქმედება, კლავიატურაზე დაკლიკვა [149]. ფუნქცია, რომელიც დაიჭერს განსაზღვრული ტიპის მოვლენას, უწოდებენ hook-პროცედურას. მას შეუძლია ზემოქმედება ყოველ მიღებულ მოვლენაზე, შემდეგ კი შეცვალოს ეს მოვლენა ან გააუქმოს.

Hooks-ის კონცეფციის გამოყენების მაგალითებია:

- შეტყობინებათა მონიტორინგი პროგრამული სისტემის გამართვის პროცესში;
- პროგრამულ სისტემაში მაკროსების ჩაწერის და რეპროდუქციის უზრუნველყოფა;
- F1-ღილაკის (help key) მხარდაჭერის უზრუნველყოფა;
- მაუსისა და კლავიატურის იმიტაცია;
- კომპიუტერზე ბაზირებული სწავლების (Computer Based Training – CBT) აპლიკაციის რეალიზაცია.

შენიშვნა: Hook-ები, როგორც წესი, ანელებს სისტემის მუშაობის სწრაფქმედებას, რადგანაც მატულობს თითოეული შეტყობინების დამუშავების მოცულობა. საჭიროა, რომ Hook-ი დაყენდეს მხოლოდ საჭიროების შემთხვევაში და მისი გამოყენების შემდეგ – სასწრაფოდ წაიშალოს.

არსებობს Hook-ების სხვადასხვა ტიპი, რომლებიც უზრუნველყოფს შეტყობინებებზე (ან მოვლენებზე) წვდომის და დამუშავების განსხვავებულ ასპექტებს. მაგალითად, WH_MOUSE ჰუკი აპლიკაციის მიერ გამოიყენება მაუსიდან შემოსული შეტყობინებათა ტრაფიკის მონიტორინგისათვის [149].

Hook-ების ყოველი ტიპისათვის სისტემა უზრუნველყოფს ჰუკების ცალკე მწკრივის (Hook Chains) მხარდაჭერას. ესაა მაჩვენებლების სია (pointers list) აპლიკაციით განსაზღვრულ, სპეციალურ ფუნქციებზე, რომლებსაც Hook Procedures-ს უწოდებენ.

როდესაც მოსულია შეტყობინება (ან მოხდა მოვლენა), რომელიც დაკავშირებულია განსაზღვრული ტიპის hook-თნ („მახესთან“), სისტემა გადასცემს შეტყობინებას ერთიმეორის

მიმდევრობით ყოველ პროცედურას, რომლებიც მითითებულია Hook Chains-ში. ჰუკ-პროცედურებს შეუძლია სხვადასხვა ფუნქციის შესრულება მათი ტიპების შესაბამისად. მაგალითად, ზოგ პროცედურას შეუძლია შეტყობინების მხოლოდ მონიტორინგი, ზოგს – შეტყობინების შეცვლა ან მათი შეჩერება მწკრივში ისე, რომ მათ ვერ მიაღწიოს შემდეგ ჰუკ-პროცედურამდე ან დანიშნულების ფანჯრამდე.

რომელიმე ტიპის ჰუკისათვის უპირატესობის მისანიჭებლად დეველოპერს შეუძლია პროცედურულ ჰუკს SetWindowsHookEx ფუნქციით განუსაზღვროს ადგილი ჰუკების მწკრივში. ჰუკ-პროცედურას, მაგალითად, ექნება ასეთი სინტაქსი:

```
LRESULT CALLBACK HookProc(
  int nCode,
  WPARAM wParam,
  LPARAM lParam
)
{
   // process event
   ...

   return CallNextHookEx(NULL, nCode, wParam, lParam);
}
```

HookProc არის ადგილი სახელის მისათითებლად, რომელსაც განსაზღვრავს აპლიკაცია.

nCode პარამეტრი არის hook-კოდი, რომელსაც იყენებს hook-პროცედურა, რათა განსაზღვროს შესასრულებელი ქმედება. Hook-კოდის მნიშვნელობა დამოკიდებულია ჰუკის ტიპზე. ყოველ ტიპს აქვს hook კოდების საკუთარი ერთობლიობა.

wParam და lParam პარამეტრების მნიშვნელობები დამოკიდებულია Hook-კოდზე, მაგრამ, ჩვეულებისამებრ, ისინი შეიცავს ინფორმაციას, იმ შეტყობინების შესახებ, რომელიც იქნა გაგზავნილი ან მიღებული.

SetWindowsHookEx ფუნქცია ყოველთვის აყენებს hook-პროცედურას hook-მწკრივის დასაწყისში. როდესაც ხდება მოვლენა, რომელსაც თვალყურს ადევნებს განსაზღვრული ტიპის ჰუკი, სისტემა გამოიძახებს hook-მწკრივს, რომელიც დაკავშირებულია ამ ჰუკთან. ყოველი ჰუკ-პროცედურა მწკრივში განსაზღვრავს - გადასცეს თუ არა მოვლენა შემდეგ პროცედურას. ამისათვის იგი გამოძახებით იყენებს CallNextHookEx ფუნქციას.

ქვემოთ მოცემულია HookTypes:

- WH_CALLWNDPROC და WH_CALLWNDPROCRET – შეტყობინებათა გაგზავნის მონიტორინგი;
- WH_CBT – computer-based training (CBT) applications;
- WH_DEBUG – განისაზღროს სისტემისათვის ჰუკ-პროცედურის გამოძახების ნებართვა;
- WH_FOREGROUNDIDLE – დაბალი პრიორიტეტის ჰუკ-პროცედურის გამოძახება;
- WH_GETMESSAGE – კლავიატურის ან მაუსის შეტ-ყობინებების მონიტორინგი;

- WH_JOURNALPLAYBACK – შეტყობინებათა დაყენება სისტემურ შეტყობინებათა რიგში;
- WH_JOURNALRECORD – შემავალი მოვლენების კონტროლი და ჩაწერა;
- WH_KEYBOARD_LL – კლავიატურიდან შეტანილი მოვლენების მონიტორინგი;
- WH_KEYBOARD – შეტყობინებათა ტრაფიკის მონიტორინგი (კლავიატურიდან შეტანილი);
- WH_MOUSE_LL – მაუსიდან მიღებული მოვლენების მონიტორინგი
- WH_MOUSE – მაუსიდან მიღებული შეტყობინებების თვალყურისდევნება;
- WH_MSGFILTER და WH_SYSMSGFILTER – შეტყობინებათა თვალყურისდევნება მენიუს და დიალოგური ფანჯრებისთვის;
- WH_SHELL - გარსის აპლიკაციის გააქტიურება.

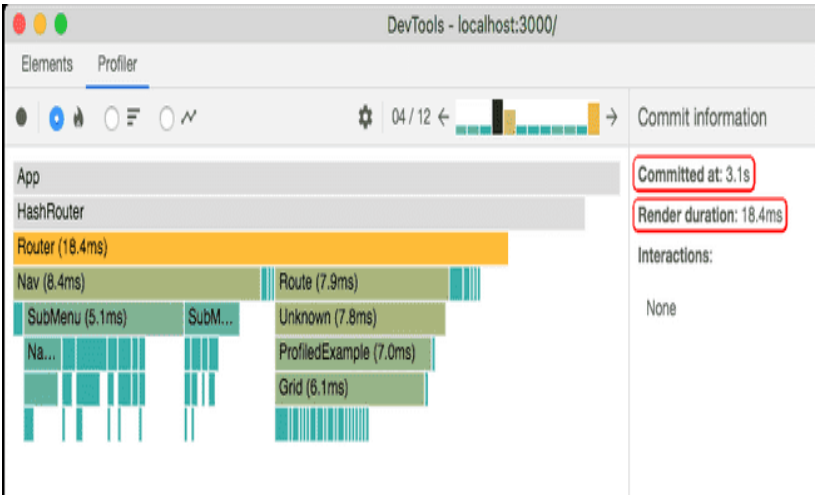
3.7.5. React Hooks Memoization მეთოდი

წინამდებარე პარაგრაფში წარმოდგენილია ჩვენ მიერ შესწავლილი, გამოკვლეული და ჩამოყალიბებული ამოცანის გადაწყვეის საკითხი – როგორიცაა React Hooks memoization პრობლემა. ამოცანა მიეკუთვნება front-end Development -ის სფეროს და მომხმარებელთა აპლიკაციების ფუნქციონირების სრულყოფის მიზნით ხორციელდება, კერძოდ, იგი ეხება სისტემის ოპტიმიზაციას, მისი მწარმოებლურობის ამაღლებას [77, 150].

ამასთანავე, წარმოდგენილია პრობლემის გადაწყვეტის გზები, კონკრეტული საილუსტრაციო მაგალითები და რეკომენდაციები ფრონტ-ენდ აპლიკაციების ასაგებად React Hook-ის გამოყენებით.

- ***ვისთვისაა ეს მეთოდი ?***

React საკმაოდ სწრაფია გამოყენების უმეტეს შემთხვევისთვის. თუ აპლიკაცია საკმარისად სწრაფია და არ არსებობს შესრულების რაიმე პრობლემა, მაშინ ეს მეთოდი არ იქნება საჭირო. სხვა შემთხვევაში, როცა შესრულების სწრაფქმედების პრობლემა სახეზეა, მისი მოგვარება ნამდვილად სასურველი და აუცილებელი საქმეა, ასე რომ, სანამ აპლიკაციის ოპტიმიზაციას დავიწყებთ, საჭიროა შედარებით კარგად გავიხსენოთ React Profiler (ნახ.3 14).



ნახ.3.14. React Profiler: Flame Chart

თუ გვაქვს აპლიკაციის იდენტიფიცირებული სცენარი, სადაც შესრულება ნელა ხდება, მაშინ დამახსოვრება (memoization) სავარაუდოდ, საუკეთესო ვარიანტია.

React.memo არის შესრულების ოპტიმიზაციის ინსტრუმენტი, უმაღლესი რიგის კომპონენტი. იგი მსგავსია React.PureComponent-ისა, მაგრამ ფუნქციების კომპონენტებისათვის, ნაცვლად კლასებისა. თუ ჩვენი ფუნქციონალური კომპონენტი იგივე შედეგს იძლევა იმავე პარამეტრებით, მაშინ React-ი დაიმახსოვრებს, გამოტოვებს კომპონენტის შესრულებას (ვიზუალიზაციას) და ხელმეორედ გამოიყენებს წინა (ბოლო) ჯერზე მიღებულ შესაბამის შედეგს.

სტანდარტულად იგი მხოლოდ ზედაპირულად შეადარებს რთულ ობიექტებს props-ობიექტში. თუ საჭიროა შედარების კონტროლის განხორციელებაც, მაშინ შესაძლებელია შედარების ფუნქციის წარმოდგენაც მეორე არგუმენტის სახით.

- ***მემორიზაციის გარეშე***

განვიხილოთ საილუსტრაციო მაგალითი, როდესაც ჩვენ არ გამოვიყენებთ დამახსოვრებას (memoization) და რატომ შეიძლება ამან გამოიწვიოს შესრულების პრობლემები.

```
function List({ items }) {
  log('renderList');
  return items.map((item, key) => (
    <div key={key}>item: {item.text}</div>
  ));
}
```

```
export default function App() {
  log('renderApp');
  const [count, setCount] = useState(0);
  const [items, setItems] = useState(getInitialItems(10));
  return (
    <div>
      <h1>{count}</h1>
      <button onClick={() => setCount(count + 1)}>
        inc
      </button>
      <List items={items} />
    </div>
  );
}
```

მაგალითი 1: ცოცხალი დემო (იხ.ორიგინალი [150])

ყოველთვის, როდესაც დაწკაპუნებულია inc, მაშინ რეგისტრირდება როგორც renderApp, ისე renderList, მიუხედავად იმისა, შეიცვალა თუა არა List-ში რამე. თუ ხე საკმარისად დიდია, ის მარტივად შეიძლება გახდეს ვიწრო ადგილი (bottleneck – „ბოთლის ყელი“). საჭიროა რენდერების (შესრულებათა) რაოდენობის შემცირება.

- ***მარტივი მემორიზაცია***

```
const List = React.memo(({ items }) => {
  log('renderList');
  return items.map((item, key) => (
    <div key={key}>item: {item.text}</div>
  ));
});
```

```
export default function App() {
  log('renderApp');
  const [count, setCount] = useState(0);
  const [items, setItems] = useState(getInitialItems(10));
      return                                                    (
    <div>
      <h1>{count}</h1>
      <button      onClick={()      =>      setCount(count      +      1)}>
       inc
      </button>
      <List                    items={items}                    />
    </div>
  );
}
```

მაგალითი 2: ცოცხალი დემო (იხ.ორიგინალი [150]).

ამ მაგალითში მემორიზაცია სწორად მუშაობს და ამცირებს რენდერების რაოდენობას. აწყობის (დამონტაჟების) დროს რეგისტრირდება renderApp და renderList, მაგრამ როდესაც ამოქმედდება inc, მაშინ მხოლოდ renderApp დარეგისტრირდება.

- ***მემორიზაცია და გამოხმაურება***

ჩავატაროთ მცირე მოდიფიკაცია, დავამატოთ inc ღილაკი სიის ყველა ელემენტს. საჭიროა სიფრთხილე, უკუ-გამოძახების (გამოხმაურების) გადაცემამ დამახსოვრებულ კომპონენტში შეიძლება გამოიწვიოს უმნიშვნელო შეცდომები.

```
function App() {
log('renderApp');
const [count, setCount] = useState(0);
```

```
const [items, setItems] = useState(getInitialItems(10));
return (
<div>
<div style={{ display: 'flex' }}>
<h1>{count}</h1>
<button onClick={() => setCount(count + 1)}>
inc
</button>
</div>
<List
items={items}
inc={() => setCount(count + 1)}
/>
</div>
);
}
```

მაგალითი 3: ცოცხალი დემო (იხ.ორიგინალი [150]).

ამ მაგალითში მემორიზაცია ვერ ხერხდება. იმის გამო, რომ ჩვენ ვიყენებთ inline lambda-ს (ჩადგმული ფუნქცია), თითოეული რენდერისთვის იქმნება ახალი მიმთითებელი (reference), რაც React.memo-ს ხდის უსარგებლოს. საჭიროა თავად ფუნქციის დამახსოვრების ერხის (გზის) განსაზღვრა, სანამ მოხდება კომპონენტის დამახსოვრება.

- ***უკუ-გამოძახების გამოყენება***

საბედნიეროდ, React-ს აქვს ორი ჩაშენებული ჰუკი (hooks) ამ მიზნით: useMemo და useCallback. პირველი სასარგებლოა ძვირადღირებული გამოთვლებისათვის, ხოლო

მეორე – უკუ-გამოძახებების გადასაცემად, რომელიც საჭიროა ოპტიმიზირებული შვილობილი კომპონენტებისათვის.

```
function App() {
log('renderApp');

const [count, setCount] = useState(0);
const [items, setItems] = useState(getInitialItems(10));

const inc = useCallback(() => setCount(count + 1));

return (
<div>
<div style={{ display: 'flex' }}>
<h1>{count}</h1>
<button onClick={inc}>inc</button>
</div>
<List items={items} inc={inc} />
</div>
);
}
```

მაგალითი 4: ცოცხალი დემო (იხ.ორიგინალი [150]).

ამ მაგალითში, მემორიზაცია კვლავ ვერ ხერხდება. inc-ის ყოველ დაკლიკვაზე გამოიძახება renderList. UseCallback–ის სტანდარტული ქცევა მდგომარეობს ახალი მნიშვნელობის გამოთვლაში, ფუნქციის ყოველი ახალი ეგზემპლარის გადაცემისას. ვინაიდან ჩადგმული ფუნქციები ქმნის ახალ ეგზემპლარს ყოველი რენდერინგის დროს, აქ Callback-ის გამოიყენება სტანდარტული კონფიგურაციით უსარგებლოა.

- ***useCallback შეტანით***

```
const inc = useCallback(() => setCount(count + 1), [count]);
```

მაგალითი 5: ცოცხალი დემო (იხ.ორიგინალი [150]).

useCallback იღებს მეორე არგუმენტს, შესატან მონაცემთა მასივს. მხოლოდ ამ მონაცემთა ცვლილების შემთხვევაში Callback-ი დააბრუნებს ახალ მნიშვნელობას. ამ მაგალითში, UseCallback დააბრუნებს ახალ მიმთითებელს (reference) ყოველ ჯერზე, როცა მთვლელი იცვლება. ვინაიდან თითოეული რენდერის დროს რაოდენობა იცვლება, ამოიტომ useCallback დააბრუნებს ახალ მნიშვნელობას მისი ყოველი შესრულებისას. არც ამ კოდის შენახვა (მემორიზაცია) ხდება.

- ***useCallback ცარიელი მასივის შეტანით***

```
const inc = useCallback(() => setCount(count + 1), []);
```

მაგალითი 6: ცოცხალი დემო (იხ.ორიგინალი [150]).

useCallback-ს შეუძლია მიიღოს ცარიელი მასივი შესატანი მონაცემების სახით, რომელიც გამოიძახებს შინაგან ჩადგმულ ფუნქციას (lambda) მხოლოდ ერთხელ და დაიმახსოვრებს მიმთითებელს მომავალი გამოძახებებისათვის. ეს კოდი იმახსოვრებს, ერთი renderApp გამოიძახება ნებისმიერი ღილაკის დაჭერით, მთავარი inc ღილაკი იმუშავებს სწორად, ხოლო შიგა inc ღილაკები შეწყვეტს სწორად მუშაობას.

მთვლელი გაიზრდება 0-დან 1-მდე და ამის შემდეგ ის შეჩერდება. ლამბდა (ჩადგმული ფუნქცია) იქმნება ერთხელ, მაგრამ გამოიძახება მრავალჯერ. რადგან lambda-ს შექმნისას არის 0, ის იქცევა ზუსტად ისე, როგორც ეს ქვემოთ მოყვანილია კოდში:

```
const inc = useCallback(() => setCount(1), []);
```

ჩვენი პრობლემის ძირეული მიზეზი ისაა, რომ ჩვენ ვცდილობთ ერთდროულად წავიკითხოთ (და ჩავწეროთ) მდგომარეობ-იდან (აში). ამისათვის საჭიროა API, რომელიც ამ მიზნისათვისაა დამუშავებული. React გვთავაზობს პრობლემის გადაჭრის ორი გზას:

- ***useState ფუნქციონალის განახლებით***

```
const inc = useCallback(() => setCount(c => c + 1), []);
```

მაგალითი 7: ცოცხალი დემო *(იხ.ორიგინალი [150]).*

useState-ით დაბრუნებულმა setter-ებმა შეიძლება მიიღოს ფუნქცია არგუმენტის სახით, სადაც შესაძლებელია მოცემული მდგომარეობის წინა-მნიშვნელობის წაკითხვა. ამ მაგალითში მემორიზაცია მუშაობს სწორად, ყოველგვარი შეცდომების გარეშე.

<u>შენიშვნა:</u> ორი საკვანძო სიტყვა (JavaScript-ოჯახიდან) განსაზღვრავს წვდომის getter და setter ფუნქციებს fullName თვისებისათვის. როცა წვდომა განხორციელებულია, მაშინ დაბრუნებული მნიშვნელობა გამოიყენება getter-იდან. როდესაც მნიშვნელობა არის მითითებული, გამოიძახება setter და გადაეცემა ეს მნიშვნელობა.

- ***useReducer***

```
const [count, dispatch] = useReducer(c => c + 1, 0);
```

მაგალითი 8: ცოცხალი დემო (იხ.ორიგინალი [150]).

ამ შემთხვევაში useReducer მემორიზაცია მუშაობს ზუსტად ისე, როგორც UseState. იმის გამო, რომ გაგზავნას გარანტირებულად ექნება ერთნაირი მითითებები რენდერინგის დროს, მაშინ Callback-ის გამოყენება აღარ არის საჭირო. ეს კი კოდს ნაკლებად მიდრეკილს ხდის მემორიზაციასთან დაკავშირებული შეცდომებისადმი.

- ***useReducer vs useState***

useReducer უფრო შესაფერისია იმ ობიექტთა მდგომარეობის მართვისათვის, რომლებიც შეიცავს მრავალ ქვემნიშვნელობებს ან როდესაც მომდევნო მდგომარეობა დამოკიდებულია წინაზე. UseReducer-ის გამოყენების ჩვეულებრივი ნიმუშია useContext, რათა თავიდან იქნას აცილებული უკუგამოძახებათა ცხადი გადაცემები კომპონენტების დიდ ხეზე .

ძირითადი წესი, რომელსაც ჩვენ ვუწევთ რეკომენდაციას ისაა, რომ გამოყენებულ იქნას UseState იმ მონაცემებისათვის, რომლებიც არ ტოვებს კომპონენტს, მაგრამ თუ მშობლებსა და შვილებს შორის საჭიროა მონაცემთა არატრივიალური ორმხრივი გაცვლა, მაშინ უკეთესი არჩევანია UseReducer.

3.7.6. React Hooks-ის აისბერგი

კლასთა კომპონენტებისაგან განსხვავებით React Hooks-ები, დაბალი დონის „სამშენებლო ბლოკებია", აპლიკაციების ასაგებად და ოპტიმიზაციისათვის მინიმალური შაბლონური კოდებით (boilerplate). იგი კომპიუტერულ პროგრამირებაში გამოიყენება და კოდის ის ნაწილია, რომელიც პროგრამის ბევრ ადგილას თავსდება უცვლელად [152].

პროგრამირების ენებში, რომლებიც ითვლება მრავალ-სიტყვიანად (verbose), დეველოპერეს უხდება დიდი მოცულობის კოდის წერა შედარებით უმნიშვნელო ზომის ფუნქციონალის შესასრულებლად [153]. ასეთ კოდს უწოდებენ შაბლონურს (boilerplate).

შაბლონური კოდების გამოყენების მოთხოვნილება შეიძლება შემცირდეს მაღალი დონის მექანიზმების საშუალებით, როგორიცაა, მაგალითად, *მეტაპროგრამირება* (აქ კომპიუტერი თვითონ წერს ავტომატურად შაბლონურ კოდს ან სვამს მას პროგრამის კომპლიაციის დროს).

შეთანხმება კონფიგურაციის შესახებ (რომელიც ავტომატურად უზრუნველყოფს კარგ მნიშვნელობებს, რათა შემცირდეს პროგრამისათვის დეტალების მითითების აუცილებლობა ყოველ პროექტში) და *მოდელურ-ორიენტირებული დაპროექტება* (რომელშიც გამოიყენება მოდელები და მოდელი->კოდის გენერატორები, რაც გამორიცხავს ხელოვნური კოდის შაბლონის აუცილებლობას).

წინამდებარე პარაგრაფში განვიხილავთ აღნიშნულ პრობლემაა და მისი გადაწყვეტის ვერსიას, რომელიც ს. დოლიძის მიერ იქნა შემუშავებული [151].

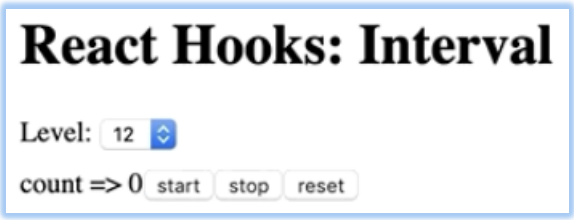
ამგვარად, აღნიშნულ კვლევით ნაშრომში, რომელიც 12 ნაწილისაგან შედგება, ილუსტრირებულია ზოგადი პრობლემები და მათი გადაწყვეტის ხერხები. აგრეთცე მოცემულია React Hooks Radar და React Hooks Checklist მცირე რეკომენდაციები მოკლე ცნობარის სახით (ნახ.3.15).

ნახ.3.15. The Iceberg of React Hooks [151]

- ***შემთხვევის შესწავლა (Case Study): ინტერვალის დანერგვა (იმლემენტაცია)***

მიზანია მთვლელის (counter) რეალიზაცია, რომელიც იწყება 0-დან და იზრდება ყოველ 500 მწამში.

გათვალისწინებულ უნდა იქნას მართვის სამი ღილაკი: *დაწყება (start), გაჩერება (stop)* და *გასუფთავება (clear)* (ნახ.3.16).



ნახ.3.16

❖ **Level 0: Hello World**

```
export default function Level00() {
  console.log('renderLevel00');
  const [count, setCount] = useState(0);
  return (
    <div>
      count => {count}
      <button onClick={() => setCount(count + 1)}>+</button>
      <button onClick={() => setCount(count - 1)}>-</button>
    </div>
  );
}
```

ეს არის მარტივი, სწორად აგებული მთვლელი, რომელიც მომხმარებლის ზემოქმედებით ღილაკებზე შესაბამისად მატულობს ან მცირდება.

❖ **Level 1: setInterval**

```
export default function Level01() {
  console.log('renderLevel01');

  const [count, setCount] = useState(0);
   setInterval(() => {
      setCount(count + 1);
   }, 500);
   return <div>count => {count}</div>;
}
```

ამ კოდის მიზანია მთვლელის მნიშვნელობის გაზრდა ყოველ 500 მწ-ში. კოდს აქვს რესურსების დიდი ხარჯი, იგი არასწორადაა რეალიზაებული. კოდი ადვილად შლის ბრაუზერის ჩანართს. ვინაიდან Level01 ფუნქცია გამოიძახება ყოველთვის, როდესაც ხდება რენდერინგი (შესრულება,

ვიზუალიზაცია), ეს კომპონენტი ქმნის ახალ ინტერვალს ყოველ ჯერზე, როდესაც იგი გაიშვება.

მუტაციები, ხელმოწერები, ტაიმერები, ჟურნალის წარმოება და სხვა თანმხლები ეფექტები არ არის დაშვებული ფუნქციის კომპონენტის ძირითად ტანში (რომელსაც უწოდებენ React-ის რენდერინგის ფაზას). ასეთი პროცესი იწვევს მომხმარებლის ინტერფეისში (UI) დამაბნეველ შეცდომებს და შეუსაბამობას.

⚯ Hooks API Reference: useEffect

❖ **Level 2: useEffect**

```
export default function Level02() {
  console.log('renderLevel02');

  const [count, setCount] = useState(0);

  useEffect(() => {
    setInterval(() => {
      setCount(count + 1);
    }, 500);
  });

  return <div>Level 2: count => {count}</div>;
}
```

გვერდითი მოვლენების (ეფექტების) უმეტესობა ხდება useEffect-ის შიგნით. ამ კოდს ასევე აქვს რესურსების დიდი ხარჯი და არასწორადაა რეალიზებილი. UseEffect-ის ქცევა (სტანდარტულად) უნდა განხორციელდეს ყოველი შესრულების (რენდერის) შემდეგ, ამიტომაც მთვლელის

მნიშვნელობის ცვლილების ყოველ ჯერზე შეიქმნება ახალი ინტერვალი.

➿ Hooks API Reference: useEffect, Timing of Effects.

❖ **Level 3: run only once**

```
export default function Level03() {
    console.log('renderLevel03');

    const [count, setCount] = useState(0);

    useEffect(() => {
        setInterval(() => {
            setCount(count + 1);
         }, 300);
     }, []);
  return <div>count => {count}</div>;
}
```

Giving *[]*, როგორც მეორე არგუმენტი, useEffect-ისთვის გამოიძახებს ფუნქციას ერთხელ, მონტაჟის შემდეგ. მიუხედავად იმისა, რომ setInterval მხოლოდ ერთხელ გამოიძახება, ეს კოდი მაინც არასწორადაა რეალიზებული.

მთვლელის მნიშვნელობა გაიზრდება 0-დან 1-მდე და ასე დარჩება. ისრის ფუნქცია შეიქმნება ერთხელ და როდესაც ეს მოხდება, მთვლელი იქნება 0.

ამ კოდს აქვს რესურსების უმნიშვნელო ხარჯი. კომპონენტის დემონტაჟის შემდეგაც კი, setCount კვლავ გამოიძახება.

⚮ Hooks API Reference: useEffect, მოვლენის პირობითი ამუშავება

Level 4: cleanup

```
useEffect(() => {
    const interval = setInterval(() => {
          setCount(count + 1);
     }, 300);
     return () => clearInterval(interval);
}, []);
```

რესურსების ხარჯის შესამცირებლად, ყველაფერი უნდა ალაგდეს, როდესაც ანკესის (ჰუკის) სიცოცხლის ციკლი დასრულდება. ამ შემთხვევაში დასაბრუნებელი ფუნქცია გამოიძახება კომპონენტის გაუქმების შემდეგ.

ამ კოდს არ აქვს რესურსების ხარჯი, მაგრამ არასწორედ ხორციელდება, ისევე, როგორც წინა.

⚮ Hooks API Reference: მოვლენის გასუფთავება

❖ **Level 5: use count as dependency**

```
useEffect(() => {
      const interval = setInterval(() => {
          setCount(count + 1);
      }, 500);
      return () => clearInterval(interval);
}, [count]);
```

დამოკიდებულებათა მასივის წარმოდგენა useEffect-ისთვის შეცვლის მის სასიცოცხლო ციკლს. ამ მაგალითში

UseEffect გამოიძახება ერთხელ, მისი ჩაყენების შემდეგ და ყოველთვის იცვლება მთვლელი. გაწმენდის (Cleanup) ფუნქცია გამოიძახება ყოველ ჯერზე მთვლელის მნიშვნელობის ცვლილების გამო, წინა რესურსის გასაუქმებლად.

ეს კოდი მუშაობს სწორად, შეცდომების გარეშე, მაგრამ იგი ოდნავ დაბნეულობას იწვევს. კერძოდ, setInterval იქმნება და იშლება ყოველ 500 მწ-ში. ყოველი setInterval-ი გამიძახება მხოლოდ ერთხელ.

⚯ Hooks API Reference: useEffect, მოვლენის პირობითი ამოქმედება

❖ Level 6: setTimeout

```
useEffect(() => {
    const timeout = setTimeout(() => {
        setCount(count + 1);
    }, 500);
    return () => clearTimeout(timeout);
}, [count]);
```

ეს კოდი და ზემოთ მოცემული კოდი სწორად მუშაობს. ვინაიდან useEffect-ი გამოიძახება მთვლელის ყოველ ცვლილებისას, setTimeout-ის გამოყენებასაც იგივე ეფექტი აქვს, როგორც setInterval-ის გამოძახებას. ეს მაგალითი არაეფექტურია,რადგან ახალი setTimeout იქმნება ყოველ ჯერზე, როდესაც ხდება რენდერინგი. React-ს აქვს უკეთესი ხერხი პრობლემის გადასაჭრელად.

❖ **Level 7: functional updates for useState**

```
useEffect(() => {
    const interval = setInterval(() => {
        setCount(c => c + 1);
    }, 500);
    return () => clearInterval(interval);
}, []);
```

წინა მაგალითში გამოიყენებოდა useEffect მთვლელის ყოველ ცვლილებისას. ეს აუცილებელი იყო მიმდინარე მნიშვნელობის განახლებისათვის.

useState უზრუნველყოფს API-ის წინა მდგომარეობის განახლებისათვის, მიმდინარე მნიშვნელობის გარეშე. ამისათვის საჭიროა lambda ფუნქციის მიწოდება setState-სთვის.

ეს კოდი მუშაობს სწორად და უფრო ეფექტურად. კომპონენტის სასიცოცხლო ციკლის განმავლობაში გამოიყენება ერთი setInterval-ი. clearInterval გამოიძახება მხოლოდ ერთხელ კომპონენტის გაუქმების შემდეგ.

🔗 Hooks API Reference: useState, ფუნქციონალის განახლება

❖ **Level 8: local variable**

```
export default function Level08() {
  console.log('renderLevel08');
  const [count, setCount] = useState(0);
  let interval = null;

  const start = () => {
    interval = setInterval(() => {
      setCount(c => c + 1);
```

```
        }, 500);
    };

    const stop = () => {
        clearInterval(interval);
    };

    return (
      <div>
          count => {count}
          <button onClick={start}>start</button>
          <button onClick={stop}>stop</button>
      </div>
    );
}
```

ჩვენ დავამატეთ დაწყების (start) და გაჩერების (stop) ღილაკები. ეს კოდი არასწორადაა შესრულებული, გაჩერების ღილაკი არ მუშაობს. ახალი მიმთითებელი იქმნება ყოველი რენდერის (სესრულების) დროს, ამიტომ გაჩერებას ექნება მიმთითებელი null-ზე.

➿ Hooks API Reference: არსებობს რაიმე ეგზემპლარის ცვლადის მსგავსი?

❖ **Level 9: useRef**

```
export default function Level09() {
   console.log('renderLevel09');
   const [count, setCount] = useState(0);

   const intervalRef = useRef(null);
```

```
    const start = () => {
        intervalRef.current = setInterval(() => {
              setCount(c => c + 1);
        }, 500);
     };

    const stop = () => {
        clearInterval(intervalRef.current);
     };

     return (
        <div>
            count => {count}
            <button onClick={start}>start</button>
            <button onClick={stop}>stop</button>
        </div>
      );
}
```

useRef არის მახე (go-to-hook) თუ საჭიროა ცვალებადი ცვლადი. ლოკალური ცვლადებისაგან განსხვავებით, React უზრუნველყოფს, რომ ყოველი რენდერის (შესრულების) დროს ერთიდაიმავე მიმთითებელზე მოხდეს დაბრუნება.

ეს კოდი, სავარაუდოდ, სწორია, მაგრამ აქვს მცირე შეცდომა. თუ დაწყება (start) გამოიძახება მრავალჯერ,, მაშინ setInterval-იც გამიძახება იმდენჯერვე, რაც გამოიწვევს რესურსის ხარჯს.

Hooks API Reference: useRef

❖ Level 10: useCallback

```
export default function Level10() {
  console.log('renderLevel10');

  const [count, setCount] = useState(0);
  const intervalRef = useRef(null);

  const start = () => {
        if (intervalRef.current !== null) {
              return;
        }

        intervalRef.current = setInterval(() => {
              setCount(c => c + 1);
        }, 500);
    };

    const stop = ( ) => {
         if (intervalRef.current === null) {
     return;
    }

    clearInterval(intervalRef.current);
    intervalRef.current = null;
  };
  return (
    <div>
        count => {count}
        <button onClick={start}>start</button>
        <button onClick={stop}>stop</button>
    </div>
  );
}
```

რესურსების ხარჯის (გაჟონვის) თავიდან ასაცილებლად, ჩვენ უბრალოდ უგულებელვყოფთ გამოძახებებს, თუ ინტერვალი უკვე გაშვებულია. მიუხედავად იმისა, რომ clearInterval (null) გამოძახება არ იწვევს შეცდომას, მაინც კარგი პრაქტიკაა რესურსის წაშლა (dispose) მხოლოდ ერთხელ.

ამ კოდს არ აქვს რესურსების გაჟონვა, სწორად არის შესრულებული, მაგრამ შეიძლება ჰქონდეს შესრულების პრობლემა.

მემორიზაცია არის React-ში შესრულების *ოპტიმიზაციის* მთავარი ინსტრუმენტი. React.memo ასრულებს ზედაპირულ შედარებას და თუ მიმთითებლები (references) იგივეა, ხდება რენდერის გამოტოვება (skipped).

თუ დაწყება (start) და შეჩერება (stop) გადაეცემა მემორიზაციის კომპონენტს, მაშინ მთელი ოპტიმიზაცია შესაძლოა ჩავარდნილიყო, რადგან ახალი მიმთითებელი ბრუნდება ყოველი რენდერის (შესრულების) შემდეგ.

🔗 React Hooks: Memoization

❖ Level 11: useCallback

```
export default function Level11() {
  console.log('renderLevel11');

  const [count, setCount] = useState(0);
  const intervalRef = useRef(null);

  const start = useCallback(() => {
    if (intervalRef.current !== null) {
      return;
    }
```

```
    intervalRef.current = setInterval(() => {
          setCount(c => c + 1);
       }, 500);
    }, []);

  const stop = useCallback(() => {
       if (intervalRef.current === null) {
         return;
       }

       clearInterval(intervalRef.current);
       intervalRef.current = null;
     }, []);

 return (
    <div>
        count => {count}
        <button onClick={start}>start</button>
        <button onClick={stop}>stop</button>
    </div>
   );
 }
```

იმისთვის, რომ React.memo-მ სწორად შეასრულოს თავისი სამუშაო, საჭიროა მხოლოდ ის, რომ შენახულ იყოს ფუნქციები, Callback Hook-ის გამოიყენეთ. ამგვარად, ერთიდაიგივე მიმთითებელი იქნება წარმოდგენილი ყოველი რენდერის (შესრულების) შემდეგ.

ამ კოდს არ აქვს რესურსების გაჟონვა, სწორადაა რეალიზებული, არ აქვს შესრულების მწარმოებლურობის პრობლემა, მაგრამ კოდი საკმაოდ რთულია, თუნდაც უბრალო მთვლელისთვის.

⚯ Hooks API Reference: useCallback

❖ Level 12: custom hook - მომხმარებლის მახე

```
function useCounter(initialValue, ms) {
  const [count, setCount] = useState(initialValue);
  const intervalRef = useRef(null);

  const start = useCallback(() => {
    if (intervalRef.current !== null) {
      return;
    }
    intervalRef.current = setInterval(() => {
      setCount(c => c + 1);
    }, ms);
  }, []);

  const stop = useCallback(() => {
    if (intervalRef.current === null) {
      return;
    }

    clearInterval(intervalRef.current);
    intervalRef.current = null;
  }, []);

  const reset = useCallback(() => {
    setCount(0);
  }, []);

  return { count, start, stop, reset };
}
```

კოდის გასამარტივებლად, საჭიროა არსებული სირთულის ინკაფსულირება მომხმარებლის მახის (custom hook) *useCounter*-ის შიგნით, შემდეგ კი წარმოდგენილ იყოს შესაბამისი სუფთა api: *{count, start, stop, reset}*.

```
export default function Level12() {
   console.log('renderLevel12');
   const { count, start, stop, reset } = useCounter(0, 500);
   return (
      <div>
          count => {count}
          <button onClick={start}>start</button>
          <button onClick={stop}>stop</button>
          <button onClick={reset}>reset</button>
        </div>
    );
}
```

Hooks API Reference: Using a Custom Hook

- **React Hooks Radar**

ყველა React Hook- ს თანაბარია, მაგრამ ზოგიერთ hook-ები, სხვებთან შედარებით, *უფრო თანაბარია*, ვიდრე სხვები.

✓ Green – მწვანე

მწვანე ჰუკები თანამედროვე React აპლიკაციების ძირითადი სამშენებლო ბლოკებია. გამოყენებისას ისინი უსაფრთხოა თითქმის ყველგან,

1. useReducer
2. useState
3. useContext

○ Yellow – ყვითელი

ყვითელი ჰუკები უზრუნველყოფს შესრულების მწარმოებლურობის სასარგებლო ოპტიმიზაციას მემორი-ზაციის გამოყენებით. სასიცოცხლო ციკლისა და მონაცემთა შეტანის მართვა უნდა მოხდეს სიფრთხილით.

1. useCallback
2. useMemo

● Red – წითელი

წითელი ჰუკები ურთიერთქმედებს ცვალებად სამყა-როსთან, გვერდითი ეფექტების გამოყენებით. ისინი ყველაზე მძლავრია და განსაკუთრებული სიფრთხილით უნდა იქნას გამოყენებული. რეკომენდებულია მომხმარებელთა ჰუკების გამოყენება ყველა არატრივიალური შემთხვევისათვის.

1. useRef
2. useEffect
3. useLayoutEffect

- **React Hooks Checklist (წესები):**

1. საჭიროა ჰუკების (Hooks) გამოყენების წესების დაცვა;

2. უნდა გამოირიცხოს რაიმე გვერდითი მოვლენების გაკეთება რენდერის მთავარ ფუნქციაში;

3. უნდა გაუქმდეს /წაიშალოს/, განადგურდეს ყველა გამოყენებული რესურსი;

4. უმჯობესია useReducer ან ფუნქციონალური განახლებები useState-სათვის, რათა თავიდან იქნას აცილებული ერთიდაიმავე მნიშვნელობის წაკითხვა და ჩაწერა მახეში (in hook);

5. არ უნდა გამოვიყენოთ ცვალებადი ცვლადები რენდერ-ფუნქციის შიგნით, უკეთესია UseRef-ის გამოყენება;

6. თუ რასაც ვინახავთ useRef-ში აქვს უფრო მცირე სასიცოცხლო ციკლი, ვიდრე თავად კომპონენტს, არ უნდა დაგვავიწყდეს რესურსის განადგურებისას მნიშვნელობის წაშლა.

7. საჭიროა სიფრთხილე უსასრულო რეკურსიისა და რესურსების უკმარობისას;

8. დამახსოვრებულ იქნას (მემორიზაცია) ფუნქციები და ობიექტები, როცა ეს საჭიროა შესრულების მწარმოებლურობის ამაღლების მიზნით;

9. სწორად უნდა ჩაიწეროს შესატანი დამოკიდებულებები (undefined => every render, [a, b] => when a or b change, [] => only once);

10. უნდა გამოვიყენოთ მომხმარებელთა ჰუკები (custom hooks) არატრივიალური შემთხვევებისთვის.

ამგვარად, Web-აპლიკაციების დეველოპმენტის სფეროში ერთ-ერთი მნიშვნელოვანი მიმართულებაა მასშტაბირებადი ვებ-აპლიკაციების დამუშავება. back-end და front-end ტექნოლოგიები მუდმივად განიცდის განახლებას და სრულყოფას. მნიშვნელოვანია აქ ოპტიმიზაციის საკითხებიც.

რა თქმა უნდა, აქტუალურია: HTML, CSS, PHP, Angular და BEM ტექნოლოგიები და მეთოდოლოგიები. სხვა სისტემების მიმოხილვასთან ერთად ჩვენ წარმოვადგინეთ ReactJS-ის შესაძლებლობები და აპლიკაციების სწრაფქმედების ოპტიმიზაციის მემორიზაციის მეთოდი (React Hooks).

React-ის საფუძველზე აგებულმა აპლიკაციებმა ბაზარზე საკმაო პოპულარობა მოიპოვა ვებ-დეველოპერებში. ისინი დაცული და მრავალი ფუნქციით აღჭურვილი საშუალებებია. React Hooks-ის ინსტრუმენტის გამოყენება მასშტაბირებადი და ეფექტური ვებ-აპლიკაციების ასაგებად მეტად მნიშვნელოვანია. იგი გვეხმარება სუსტი ადგილების გამოვლენისა და მწარმოებლურობის ამაღლების (ოპტიმიზაციის) პრობლემების გადაწყვეტაში.

IV თავი
Web-სერვისების ავტომატური ტესტირების თანამედროვე საინფორმაციო ტექნოლოგიები

4.1. Selenium IDE სამუშაო გარემოს გაცნობა

Selenium-IDE (Integrated Development Environment) არის პროგრამული პროდუქტი, რომლის საშუალებითაც შესაძლებელია ტესტ-ქეისების შექმნა ავტომატურ რეჟიმში [154-157]. იგი ადვილად გამოყენებადი Firefox ბრაუზერის პლაგინია.

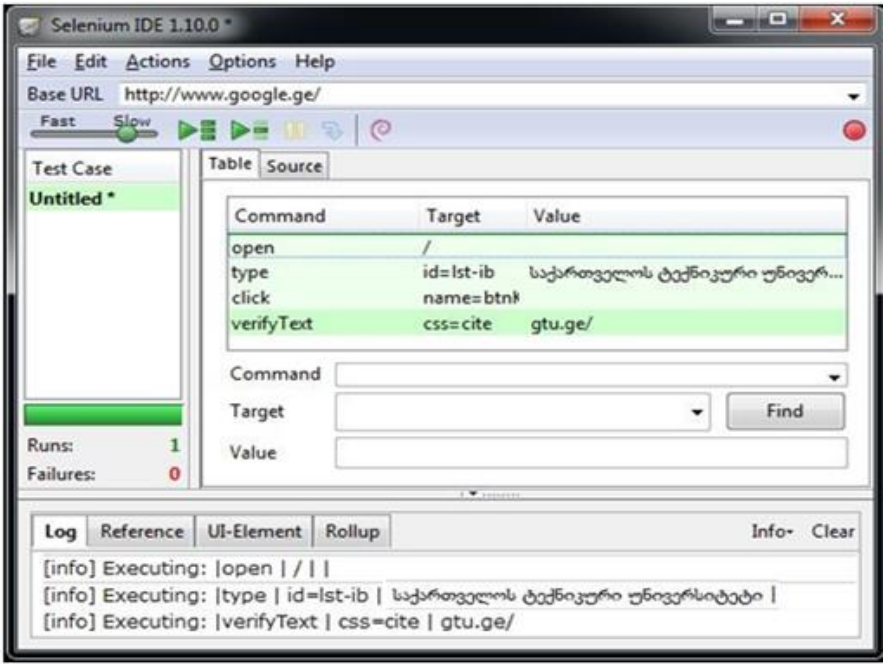
ზოგადად ითვლება, რომ Selenium IDE ყველაზე ეფექტური გზაა ტესტ-ქეისების შესაქმნელად. იგი შეიცავს კონტექსტურ მენიუს, რომლის საშუალებით შეგვიძლია მოვნიშნოთ ვებგვერდის UI (User Interface) ელემენტი და შემდგომ ავირჩიოთ შესაბამისი Selenium ბრძანება. Selenium IDE არის არამხოლოდ ტესტ-სკრიპტების წერისას დახარჯული დროის მოგების საშუალება, არამედ იგი საუკეთესო არჩევანია Selenium სკრიპტ-სინტაქსის შესასწავლად [158].

Selenium IDE თავსებადია მხოლოდ Firefox ბრაუზერზე, არ არსებობს მისი ალტერნატივა, რომელიც სცენარებს ჩაიწერს IE-ზე (Internet Explorer) ან სხვა ბრაუზერზე. მიუხედავად ამისა Selenium IDE-ის და Firefox-ის საშუალებით შექმნილი ტესტ-სკრიპტები შესაძლებელია შესრულებაზე გავუშვათ ნებისმიერ ინტერნეტ ბრაუზერზე (IE, FF, Chrome, Opera,..) Selenium RC-ის საშუალებით [154].

როგორც აღვნიშნეთ, Selenium IDE არის Firefox ბრაუზერის პლაგინი, რომლის ინსტალაციისათვის საჭიროა:

- გავხსნათ ლინკი: http://seleniumhq.org/download/;
- გადმოვწეროთ და დავაინსტალიროთ Selenium IDE როგორც Firefox პლაგინი;
- გადავტვირთოთ Firefox ბრაუზერი;
- Tools მენიუდან გავუშვათ Selenium IDE.

4.1 ნახაზზე ნაჩვენებია Selenium IDE-ს სამუშაო გარემო.



ნახ.4.1. Selenium IDE სამუშაო გარემო

❖ *მენიუ:*

➢ **File** მენიუს აქვს ტესტ-ქეისების და ტესტ-სუიტების ოფციები. მათი გამოყენებით შესაძლებელია ახალი ტესტ-ქეისის დამატება, გახსნა, შენახვა და ექსპორტი სხვადასხვა

ენებზე. ასევე შესაძლებელია მიმდინარე ტესტ-ქეისის გახსნა. ყველა ჩამოთვლილი ფუნქცია ვრცელდება ტესტ-სუიტებისთვისაც;

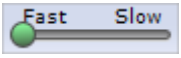
- **Edit** მენიუს საშუალებით შესაძლებელია კოპირება, გადაწერა, წაშლა, ყველაფრის მონიშვნა, რათა დავარედაქტიროთ Selenium ბრძანებები მოცემულ ტესტ-ქეისში;

- **Options** მენიუ განკუთვნილია Setting–ის ცვლილებებისათვის. აქედან შესაძლებელია განისაზღვროს ბრძანების ლოდინის დრო, ტესტ-ქეისების ენა და სხვა;
- **Help** მენიუ არის სტანდარტული Firefox help მენიუ, რომელიც შეიცავს Selenium IDE – ის ოფიციალურ დოკუმენტაციას.

❖ **პროგრამული პანელი:**

პროგრამულ პანელზე განლაგებულია ღილაკები, რომლებიც უზრუნველყოფს ტესტ-ქეისების ჩაწერას, გაშვებას, კონტროლს, დებაგს და ა.შ. (ნახ.4.2).

ნახ.4.2. Selenium IDE-ს პროგრამული პანელი



Speed Control (სიჩქარის კონტროლი): აკონტროლებს ტესტ-ქეისის შესრულებაზე გაშვების სიჩქარეს;


Run All (ყველას გაშვება): შესრულებაზე უშვებს მოცემულ ტესტ-სუიტს, შემადგენელი ტესტ-ქეისებით;

 Run (გაშვება): შესრულებაზე უშვებს მონიშნულ ტესტს. როცა მოცემულია მხოლოდ ერთი მარტივი ტესტი, ამ შემთხვევაში ღილაკს და "Run All" ღილაკს ერთიდაიგივე ფუნქცია აქვს;

 Pause / Resume (გაჩერება, განახლება): უზრუნველყოფს გაშვებული ტესტ-ქეისის გაჩერება-განახლებას;

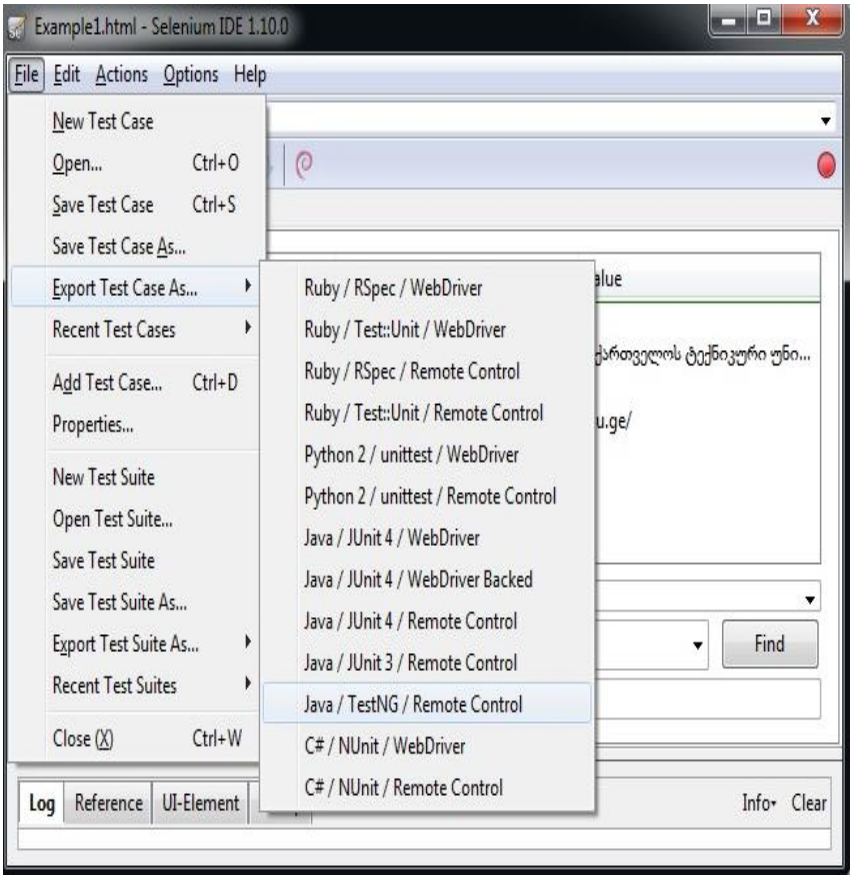
 Step (ბიჯი): მისი საშუალებით შესაძლებელია ტესტ-ქეისის გაშვება ბიჯებით, ანუ ყოველ ბიჯზე სრულდება ერთი Selenium ბრძანება. იგი გამოიყენება დებაგის დროს;

 Record (ჩამწერი): იწერს მომხმარებლის მიერ ინტერნეტ–ბრაუზერში გაკეთებულ მოქმედებებს;

Selenium IDE-ს საშუალებით შესაძლებელია ჩაწერილი HTML სკრიპტი დავაექსპორტიროთ რომელიმე თანამედროვე პროგრამირების ენაზე. 4.3 ნახაზზე ასახულია Java-ზე ექსპორტი, ხოლო 4.4-ზე გენერირებული Java კოდი [154].

მას შემდეგ რაც დავაინსტალირეთ Selenium IDE, მისი საშუალებით შეგვიძლია გავაანალიზოთ ტესტირების პროცესი.

ზოგადად არსებობს რამდენიმე წესი, რომელიც უნდა გავითვალისწინოთ ტესტის შექმნისას. ეს წესები ვრცელდება ყველა ავტომატურ ტესტზე, რომლებიც განკუთვნილია სამომხმარებლო ინტერფეისებისთვის.



ნახ.4.3. ტესტ-სკრიპტის Java-ზე ექსპორტი

```java
package com.example.tests;

import com.thoughtworks.selenium.*;
import org.testng.annotations.*;
import static org.testng.Assert.*;
import java.util.regex.Pattern;

public class example1 extends SeleneseTestNgHelper {
    @Test public void testExample1() throws Exception {
        selenium.open("/");
        selenium.type("id=lst-ib", "საქართველოს ტექნიკური უნივერსიტეტი");
        selenium.click("name=btnK");
        verifyEquals(selenium.getText("css=cite"), "gtu.ge/");
    }
}
```

ნახ.4.4. ექსპორტის შედეგად მიღებული Java ფაილი

ავტომატური ტესტირების წესები:

- ტესტს ყოველთვის განსაზღვრული უნდა ჰქონდეს *საწყისი წერტილი.* ჩვენ შემთხვევაში ეს ნიშნავს გვერდის გახსნას, საწყისი URL - ის მითითებას, რომ დაიწყოს პროცესი;
- ტესტი არ უნდა იყოს *დამოკიდებული* სხვა ტესტის გაშვების შედეგზე. თუ მოცემული ტესტი ჩავარდა (გავიდა შეცდომაზე), ამან არ უნდა გამოიწვიოს მომდევნო ტესტის ჩავარდნა;
- ტესტმა ერთჯერ უნდა დატესტოს ერთი შემთხვევა;
- ტესტი უნდა გაიწმინდოს თვითონვე.

რომელიმე ამ წესთაგანი შესაძლებელია დარღვეულ იქნას, რადგან მათი დარღვევა შეიძლება ნიშნავდეს, რომ თქვენ იპოვნეთ გამოსავალი, მაგრამ დიდი რაოდენობის ტესტების შემთვხვევაში შეიძლება მივიღოთ პირიქით: პატარა ტესტის ჩავარდნამ გამოიწვიოს სცენარის უდიდესი ნაწილის გაწითლება.

ამ წესების შესაბამისად შევქმნათ ტესტი Selenium IDE–ს გამოყენებით. ტესტის ჩასაწერად გავუშვათ Mozilla Firefox და მისი Tools მენიუდან ავირჩიოთ Selenium IDE. შევნიშნოთ რომ ჩაწერის ღილაკი არის ავტომატურად მონიშნული [154].

ტესტის ჩაწერისთვის საჭიროა შევასრულოთ შემდეგი ქმედებები:

1) შევცვალოთ **Base URL** შემდეგი მისამართით:

http://book.theautomatedtester.co.uk/

2) დავაჭიროთ Radio button-ზე;

3) შევცვალოთ **Select**–ის მნიშვნელობა;

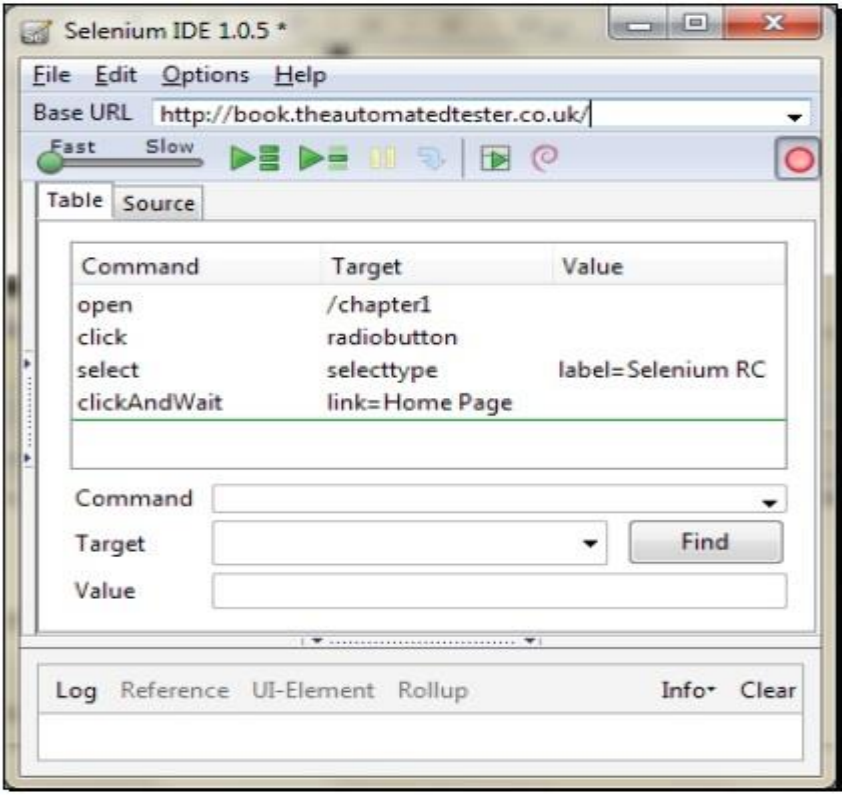
4) გადავიდეთ Home Page ლინკზე;

5) ჩაწერილი ტესტი უნდა გამოიყურებოდეს 4.5 ნახაზის მსგავსად;

6) დავაჭიროთ შესრულებაზე გაშვების ღილაკს:  .

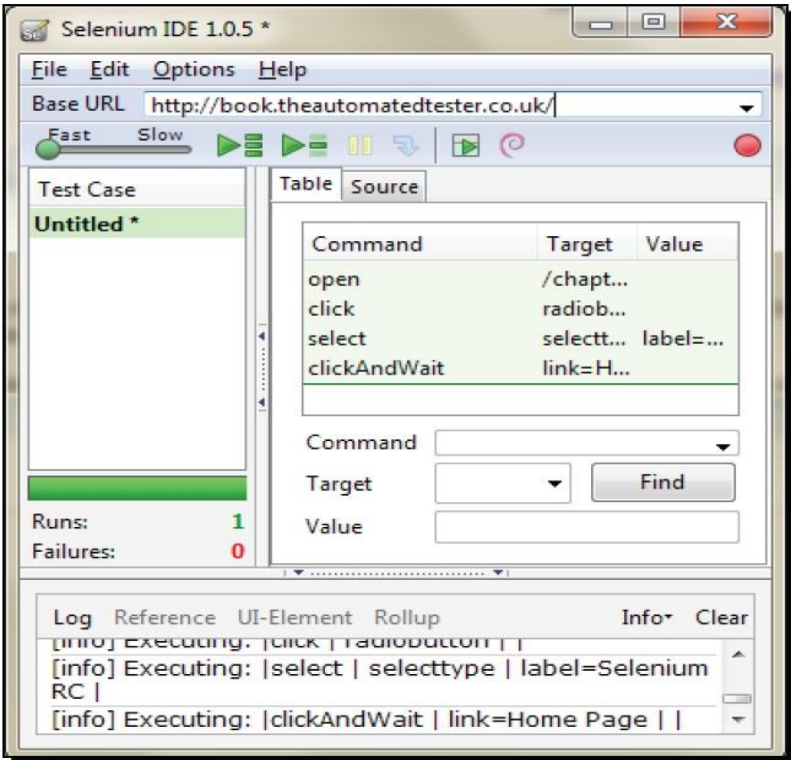
7) როდესაც ტესტი დასრულდება, იგი უნდა გამოიყურებოდეს შემდეგი სქრინშოტის ანალოგურად (ნახ.4.6).

ამრიგად, ჩვენ წარმატებით ჩავწერეთ პირველი ტესტი და გავუშვით შესრულებაზე. განხილულ ტესტში საწყის წერტილად შეიძლება განვიხილოთ **open** ბრძანება. მას აქვს მინიჭებული საწყისი URL-ი, საიდანაც იგი გადადის /chapter1 ლინკზე, და იწყება ჩაწერის პროცესი [42].



ნახ.4.5. Selenium IDE ჩაწერილი ტესტი

როდესაც ტესტი დაასრულებს მუშაობას, მივიღებთ შესრულებული ქმედებების მიმდევრობას, რომელთაც აქვს მწვანე ფონი. იგი გვიჩვენებს, რომ მოქმედებები დასრულდა წარმატებით. მარცხენა მხარეს მოცემულია **Runs: 1**, ანუ გვაქვს ერთი წარმატებული ტესტი. თუ ტესტი ჩავარდებოდა, მაშინ **Failures** იქნებოდა 1-ის ტოლი.



ნახ.4.6. Selenium IDE ტესტის შედეგი

განხილულ მაგალითებში ჩვენ ჩავწერეთ პროცესები, რომლებსაც ახორციელებს მომხმარებელი. შევამოწმეთ ღილაკების და ლინკების მუშაობა.

სამწუხაროდ, ჩვენ არ განგვიხილავს შემთხვევა, რომ სხვა რაიმე ელემენტი არსებობს Web-გვერდზე, დამალული ან ცხადი სახით [42].

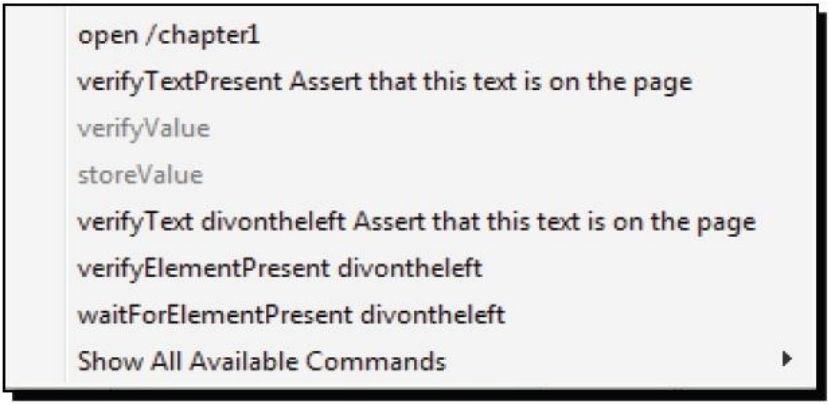
4.2. Selenium IDE: ვალიდაცია

Selenium-ს გააჩნია ორი ტიპის მექანიზმი Web-გვერდზე არსებული ელემენტების შემოწმებისათვის. პირველი მექანიზმია assert: იგი ამოწმებს არის თუ არა ელემენტი მოცემულ გვერდზე. ტესტმა თუ ელემენტი ვერ იპოვნა, ამ შემთხვევაში ტესტი გაჩერდება და დასრულდება.

მეორე არის verify: ისიც აგრეთვე ანალოგიურად ამოწმებს არის თუ არა ელემენტი მოცემულ გვერდზე, მაგრამ თუ ელემენტი არ არსებობს, მაშინ ტესტი აგრძელებს მუშაობას.

ილუსტრაციისთვის გამოვიყენოთ განხილული ტესტი და დავამატოთ assert ან verify ბრძანებები. ამისთვის საჭიროა Selenium IDE-ს კონტექსტური მენიუს გამოყენება [154].

გვერდზე არსებულ ელემენტთან მივიტანოთ მაუსის კურსორი და დავაჭიროთ მარჯვენა ღილაკს, გამოჩნდება შემდეგი ფორმის კონტექსტური მენიუ (ნახ.4.7).



ნახ.4.7. Selenium IDE კონტექსტური მენიუ

შევცვალოთ განხილული ტესტი: შევამოწმოთ სხვა ელემენტების არსებობა განხილულ გვერდზე.

1) გავხსნათ Selenium IDE ჩაწერის რეჟიმში;
2) მივუთითოთ Base URL –

http://book.theautomatedtester.co.uk/

3) გადავიდეთ Chapter1-ზე;
4) დავაჭიროთ Radio button-ს;
5) შევცვალოთ Select–ის მნიშვნელობა Selenium Grid-ით;
6) დავამოწმოთ Assert that this text is on the page -ს არსებობა;
7) დავადასტუროთ გვერდზე Verify-ღილაკის არსებობა. გამოვიყენოთ კონტექსტური მენიუ;
8) ყველა პუნქტის შესრულების შემდეგ, Selenium IDE უნდა გამოიყურებოდეს შემდეგნაირად (ნახ.4.8).

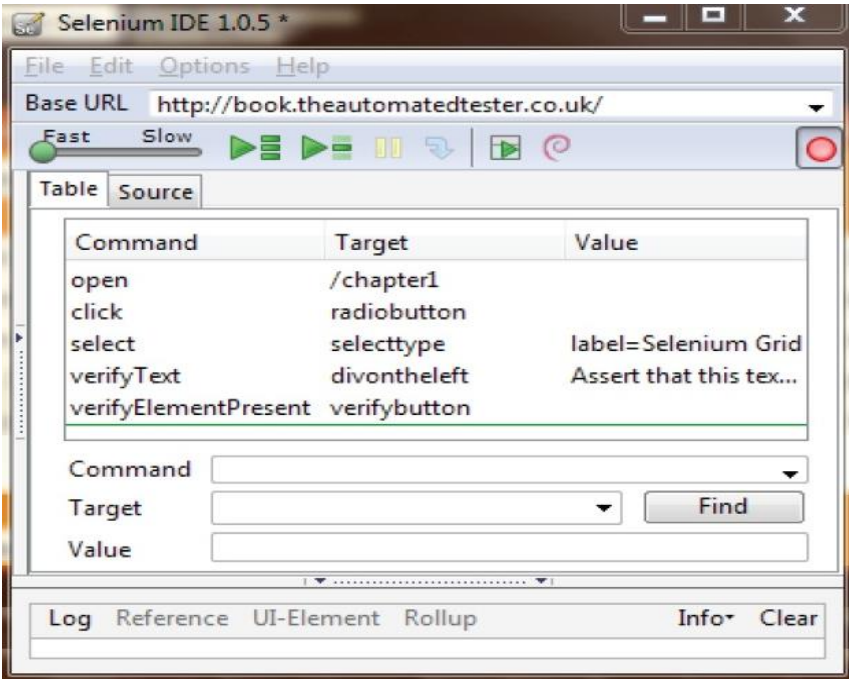
თუ გავუშვებთ ტესტს, დავინახავთ, რომ მოხდება ტექსტის და ღილაკის არსებობის შემოწმება.

გავითვალისწინოთ, რომ ყველა verify ბრძანებას აქვს მუქი მწვანე ფერი.

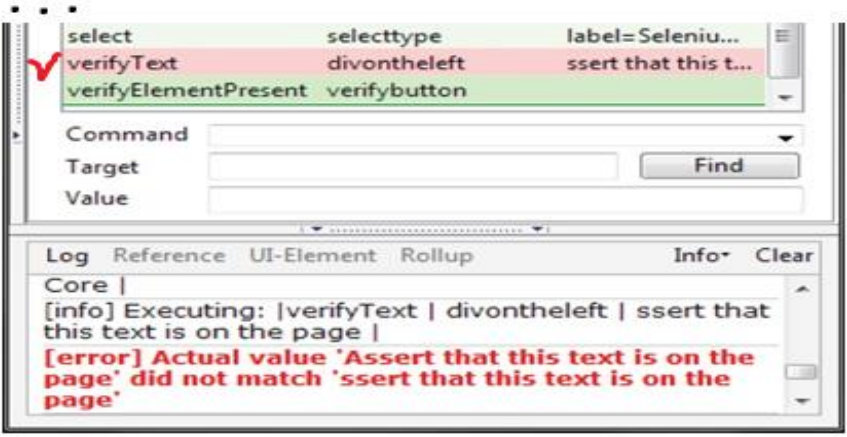
რა მოხდებოდა იმ შემთხვევაში თუ verify ბრძანება ვერ იპოვნიდა იმას, რაც აღვწერეთ ?

Selenium IDE დააფიქსირებდა შეცდომას, რომ აღწერილი ტექსტი არ არსებობს, მაგრამ იგი გააგრძელებდა ტესტის შესრულებას.

ეს შემთხვევა მოცემულია ნახ.4.9 ნახაზზე [155]:



ნახ.4.8. Selenium IDE ჩაწერილი ტესტი



ნახ.4.9. Selenium IDE გაშვებული ჩავარდნილი ტესტი

verify ბრძანების ნაცვლად თუ გამოვიყენებთ assert ბრძანებას, ამ შემთხვევაში ტესტი დაასრულებს მუშაობას იქ, სადაც დაფიქსირდება შეცდომა. საბოლოოდ ჩვენ ვნახეთ, რომ შეგვიძლია გამოვიყენოთ assert და verify ბრძანებები, რათა შევამოწმოთ არსებობს თუ არა განსაზღვრული ელემენტი მოცემულ გვერდზე.

ავტომატურად, Selenium IDE არ იწერს მსგავსი ტიპის ბრძანებებს, ამიტომ ამ სახის ბრძანებები უნდა მივუთითოთ ხელით [154]. საბოლოოდ შეგვიძლია დავასკვნათ, რომ თუ ჩვენ გამოვიყენებთ assert ბრძანებას და ის იპოვნის შეცდომას, ამ შემთხვევაში ტესტი გაჩერდება, verify ბრძანების დროს კი ტესტი აგრძელებს მუშაობას. ეს არის მათი ერთადერთი განმასხვავებელი თვისება. მიუხედავად ამისა ორივე ტიპის ბრძანებებს აქვთ თავისი გამოყენების სფეროები.

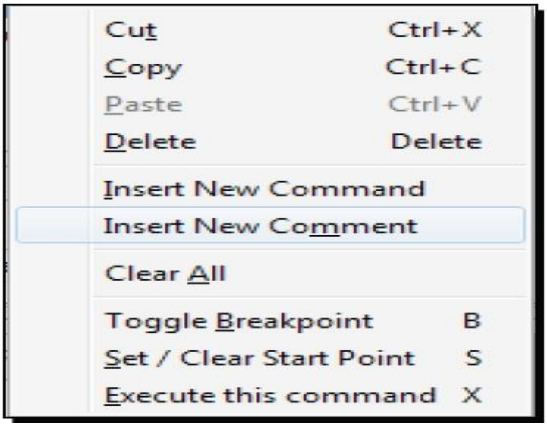
ქვემოთ მოცემულია სხვადასხვა verify და assert ბრძანებები:

- **verifications**
  - verifyElementPresent – ელემენტის არსებობის შემოწმება;
  - verifyElementNotPresent – ელემენტის არარსებობის შემოწმება;
  - verifyText – ტექსტის შემოწმება;
  - verifyAttribute – ატრიბუტის შემოწმება;
  - verifyChecked – მონიშვნის შემოწმება (ჩექბოქსი, რადიობოქსი);
  - verifyAlert – ალერტის შემოწმება;
  - verifyTitle – სათაურის შემოწმება;
- **assertions**
  - assertElementPresent - ელემენტის არსებობის შემოწმება;
  - assertElementNotPresent – ელემენტის არარსებობის შემოწმება;
  - assertText – ტექსტის შემოწმება;

- assertAttribute – ატრიბუტის შემოწმება;
- assertChecked – მონიშვნის შემოწმება (ჩექბოქსი,რადიობოქსი);
- assertAlert – ალერტის შემოწმება;
- assertTitle – სათაურის შემოწმება.

როგორც ჩვენთვის ცნობილია, იმისათვის, რომ გვქონდეს ადვილად აღქმადი პროგრამული კოდი, საჭიროა გამოვიყენოთ კომენტარები, რომლებიც დაგვეხმარება პროგრამული კოდის გარჩევისას. განხილული ფაქტის მსგავსად, კარგი იქნება თუ ჩვენს ავტომატურ ტესტებს დავურთავთ კომენტარებს, რომლებიც მომავალში გამოყენებულ იქნება სხვა ტესტერის მიერ [154].

იმისთვის, რომ დავამატოთ კომენტარი განხილულ ტესტში, საჭიროა: Selenium IDE - ს კონტექსტური მენიუდან ავირჩიოთ Insert New Comment ბრძანება, შედეგად მივიღებთ ცარიელი ადგილს ბრძანებებს შორის (ნახ.4.10).



ნახ.4.10. Selenium IDE კონტექსტური მენიუ

განხილულ ნახაზზე მოცემულია კომენტარი, რომელიც შექმნილია Selenium IDE-ს დახმარებით. ტესტის კომენტარი ყოველთვის გამოჩნდება მუქი წითელი ფერის ტექსტით.

თანამედროვე ინფორმაციულ ტექნოლოგიებში გავრცელებული Web აპლიკაციები, სამწუხაროდ, ბრაუზერის მხოლოდ ერთ ფანჯარაში არ არსებობს. მაგალითისთვის განვიხილოთ რეპორტების საიტი. თითოეულ რეპორტს აქვს თავისი საკუთარი ფანჯარა, სადაც მომხმარებელი გადაადგილდება ერთი ფანჯრიდან მეორეზე და ა.შ.

მრავალფანჯრიანი Web გვერდის ტესტირების პროცესში შეიძლება წავაწყდეთ სირთულეებს. ეს არის ის დეტალური საკითხები, რომლებსაც უნდა მივაქციოთ ყურადღება, რათა უპრობლემოდ გადავიდეთ სხვადასხვა ფანჯრებზე.

მრავალფანჯრიანი აპლიკაციების მართვა არის ერთერთი რთული საკითხი Selenium ავტომატური ტესტირების დროს. ამ დროს ბრაუზერის მიერ გახსნილ ყოველ შვილობილ ფანჯარაზე უნდა გადავიდეს Selenium-ი იმდენჯერ, რამდენჯერაც წარმოიქმნება შვილობილი პროცესი.

მრავალფანჯრიანი Web გვერდის საილუსტრაციოდ განვიხილოთ მაგალითი, სადაც გვერდის ელემენტზე მაუსით დაკლიკვა წარმოქმნის ახალ ფანჯარას. თუ ჩართულია ბრაუზერის pop-up blocker, გავთიშოთ იგი, რათა მოხდეს ახალი pop-up ფანჯრის ჩამოტვირთვა [42].

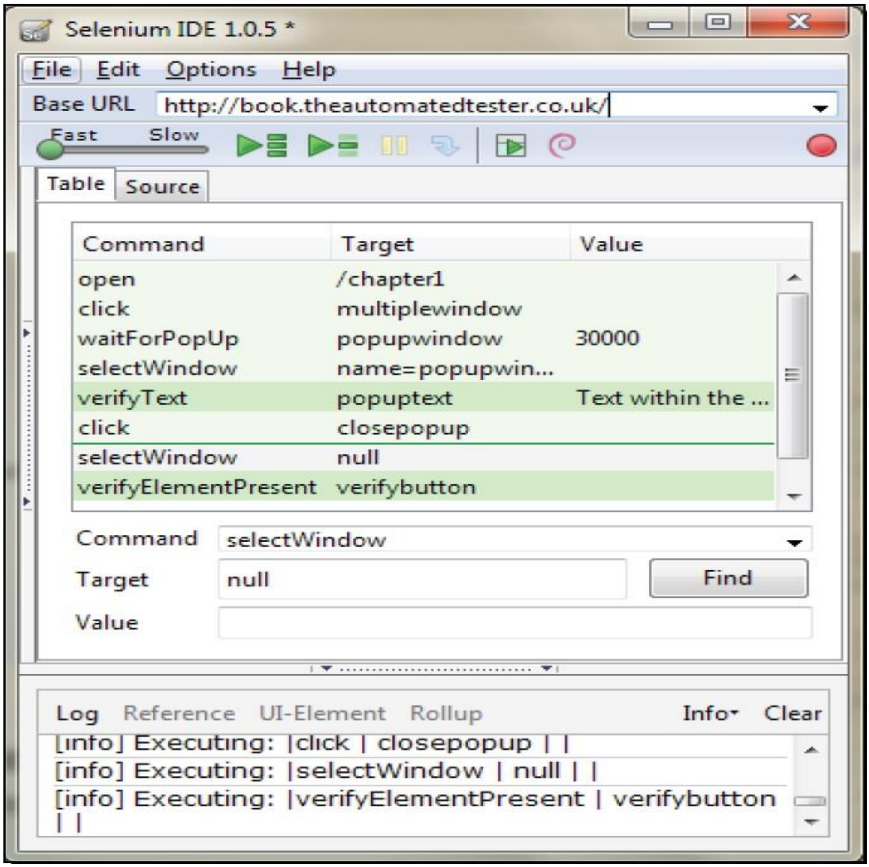
1) ავამუშავოთ Selenium IDE, გადავიდეთ Chapter1 გვერდზე;

2) დავკლიკოთ ელემენტი, რომლის ტექსტია **Click this link to launch another**, შედეგად წარმოიქმნება პატარა ფანჯარა;

3) ფანჯარის ჩამოიტვირთვის შემდეგ დავამატოთ verify ბრძანება ელემენტისთვის წარწერით – Text within the pop up window;

4) დავაჭიროთ **Close the window** ლინკს;

5) დავადასტუროთ, რომ დავბრუნდით საწყის გვერდზე (ნახ.4.11).



ნახ.4.11. Selenium IDE მრავალფანჯრიანი სისტემა

შევამჩნევთ, რომ განხილულ ტესტში Selenium აჭერს ლინკს, რომელიც ხსნის ახალ ფანჯარას და Selenium ავტომატურად სვამს **waitForPopUp** ბრძანებას. მაშასადამე, ტესტი ხვდება, რომ იგი უნდა დაელოდოს web სერვერს, რათა მიიღოს პასუხი მოთხოვნაზე და საბოლოოდ ბრაუზერმა გახსნას ახალი გვერდი. ნებისმიერი ბრძანება, რომელიც ითხოვს ახალი გვერდის ჩამოტვირთვას web სერვერიდან, იქნება ***waitFor*** ტიპის ბრძანება.

განვიხილოთ ბრძანება **selectWindow**. იგი ატყობინებს Selenium-ს, რომ მას სჭირდება გადავიდეს სხვა ფანჯარაზე, რომლის სახელია **popupwindow** და მომდევნო ბრძანებები შეასრულოს აღნიშნულ ფანჯარაზე მანამ, სანამ გადავმისამართდებით კიდევ სხვა ფანჯარაზე და ა.შ.

როდესაც ტესტი დაასრულებს მუშაობას pop-up ფანჯარაზე, შემდეგ საჭიროა დავბრუნდეთ მშობელ ფანჯარაზე. ამისთვის საჭიროა განვსაზღვროთ ფანჯარა როგორც **null**. ეს უზრუნველყოფს ტესტის უკან, მშობლიურ ფანჯარაზე დაბრუნებას.

4.3. Selenium ტესტები Ajax აპლიკაციებისთვის

Ajax (Asynchronous JavaScript and XML) არის Web ტექნოლოგია, რომელიც უზრუნველყოფს სწრაფი, დინამიკური გვერდების შექმნას. Ajax ტექნოლოგიის დროს Web აპლიკაციას შეუძლია გააგზავნოს და მიიღოს მონაცემები

სერვერიდან ასინქრონულად, ანუ განაახოლს Web გვერდის რაღაც ნაწილი მთლიანი გვერდის განახლების გარეშე.

Ajax ტექნოლოგიით დაწერილი აპლიკაციების ავტომატური ტესტირებისას, შესაძლებელია ტესტის ჩავარდნის მიზეზი გახდეს ელემენტის არარსებობა მიმდინარე გვერდზე – როდესაც ბრაუზერი განახლების რეჟიმშია და ელოდება გაგზავნილი მოთხოვნის პასუხს სერვერიდან. შესაძლებელია აპლიკაციის მუშაობის სიჩქარე დავარდნილი იყოს სხვადასხვა ფაქტორების გამო: ტექნიკური მოწყობილობის სიმძლავრე, ქსელის პრობლემა, გადატვირთული რეჟიმი და სხვ.

არსებული პრობლემის გადაჭრა შესაძლებელია waitFor ტიპის ბრძანებების მეშვეობით. ასეთი ტიპის ბრძანებებს Selenium გადაჰყავს ლოდინის რეჟიმში. waitFor ბრძანებები გამოიყენება იმ შემთხვევაში, როდესაც საჭიროა Selenium გადავიდეს ლოდინის რეჟიმში.

აღნიშნული ბრძანებების დროს Selenium-ის ლოდინის დრო არის მაქსიმუმ 30 წამი „გამოუცხადებლად", მაგრამ შეგვიძლია აღნიშნული დროის ცვლილება setTimeout ბრძანების საშუალებით. მაშასადამე, setTimeout ბრძანება ზოგადად, განსაზღვრავს waitFor ტიპის ბრძანებების მაქსილამურ ლოდინის დროს.

არსებობს Selenium-ის სხვადასხვა ბრძანებები, რომლებიც გამოიყენება web ელემენტების ლოდინისთვის. ქვემოთ ჩამოთვლილია waitFor ტიპის ბრძანებები [154]:

- WaitForAlertPresent – დაელოდე ალერტის წარმოქმნას;
- waitForAlertNotPresent –დაელოდე ალერტის არწარმოქმნას;
- waitForElementPresent – დაელოდე ელემენტის გამოჩენას;

- waitForElementNotPresent – დაელოდე ელემენტის არგამოჩენას;
- waitForTextPresent – დაელოდე ტექსტის გამოჩენას;
- waitForTextNotPresent – დაელოდე ტექსტის არგამოჩენას;
- waitForPageToLoad – დაელოდე გვერდის ჩამოტვირთვას;
- waitForFrameToLoad – დაელოდე ფრეიმის ჩამოტვირთვას.

ზოგიერთი ზემოთ ჩამოთვლილი ბრძანება შეიძლება გაეშვას სხვა ბრძანების შესრულებისას ავტომატურად. მაგალითად, განვიხილოთ clickAndWait ბრძანება. მისი შესრულებისას ხდება ორი ბრძანების გაშვება: იგი ჯერ აკეთებს ჩვეულებრივ click–ს და შემდგომ ასრულებს waitForPageToLoad ბრძანებას. ასევე ჩვენთვის ნაცნობი open ბრძანება, რომელიც სრულდება მაშინ, როდესაც გვერდი მთლიანად ჩამოიტვირთება.

ზოგჯერ საჭიროა web გვერდის ელემენტების შენახვა ტესტში, რათა მოგვიანებით გამოვიყენოთ ისინი. ეს ნიშნავს იმას, რომ ტესტში შევინახოთ web გვერდზე არსებული მონაცემი შემდგომი გამოყენების მიზნით. Selenium ტექნოლოგიით მონაცემების შენახვა ხდება store ტიპის ბრძანებით. ტესტებში ცვლადების გამოყენების ფორმატია:

storedVars['variableName'];

ჩვენ შევქმენით რამდენიმე ტესტი, წარმატებით გავუშვით შესრულებაზე, ვნახეთ თუ როგორ მუშაობს Selenium ტესტი AJAX აპლიკაციებზე, მაგრამ სამწუხაროდ ცოტა რთულია ისეთი ტესტის შექმნა, რომელიც გაეშვება ბოლომდე. ამიტომ, ავტომატური ტესტების შექმნისას

საჭიროა შექმნილი ტესტის დებაგი, რომ ვნახოთ დაწერილი ტესტ-სრიპტის შეცდომები [154].

თუ ტესტი გაშვებულია **Pause** ღილაკის საშუალებით, შესაძლებელია გავაჩეროთ მისი შესრულება. როდესაც ტესტის შესრულება გაჩერდება, **Step** ღილაკის საშუალებით შესაძლებელია მომდევნო ბრძანებების მიმდევრობით თითო-თითოდ შესრულება.

თუ მიმდინარე web გვერდზე გვაქვს ელემენტის პოვნის პრობლემა, შეგვიძლია ჩავწეროთ ელემენტის დასახელება Target ველში, დავაწვეთ Find ღილაკს და web გვერდის ელემენტი წამიერად განათდება. თუ მიმდინარე გვერდზე არცერთი ელემენტი არ განათდა, მაშინ მსგავსი სახელის ელემენტი არ არსებულა web გვერდზე. საბოლოოდ, ჩვენ განვიხილეთ ტესტ–სკრიპტების ჩაწერა Selenium IDE-ის საშუალებით და ჩაწერილი ტესტები წარმატებით გავუშვით შესრულებაზე.

სამწუხაროდ, თანამედროვე Web ტექნოლოგიებში არსებობს ისეთი მოქმედებები, რომელთაც Selenium ვერ ასრულებს. როგორც ვიცით, Selenium შექმნეს JavaScript-ზე, ამის გამო იგი ცდილობს განახორციელოს ისეთი მოქმედებები, რაც შესაძლებელია JavaScript-ით. სხვა სიტყვებით რომ ვთქვათ, იგი შემოიფარგლება JavaScript ტექნოლოგიის შესაძლებლობებით.

ქვემოთ ჩამოთვლილია Selenium IDE-ს არსებული „სუსტი წერტილები":

- **Serverlight და Flex / Flash** აპლიკაციების მოქმედებები შეუძლებელია ჩაიწეროს Selenium IDE-ით. აღნიშნული ორივე

ტექნოლოგია მოქმედებს თავის საკუთარ ელემეტებზე და არ იმართება DOM ტექნოლოგიით. DOM (Document Object Model) არის პლატფორმისგან, პროგრამული ენისგან დამოუკიდებელი მოდელი, რომელიც უზრუნველყოფს ობიექტების წარმოდგენას HTML, XHTML, XML დოკუმენტებში. იგი HTML დოკუმენტია ხისებური სტრუქტურით;

- **HTML5** ტექნოლოგია არ არის Selenium IDE-ს მიერ ბოლომდე მხარდაჭერილი. აღნიშნული პრობლემის კარგი მაგალითია HTML5 ელემენტი, რომელსაც აქვს ატრიბუტი contentEditable=true. თუ განვიხილავთ type ბრძანებას, რათა ჩავწეროთ რამე ტექსტი მოცემულ ელემენტზე, დავინახავთ, რომ ტესტი წარმატებით დაასრულებს მუშაობას, მაგრამ UI (user interface) ელემენტში ტექსტი არ ჩაიწერება [42,154];
- Selenium IDE არ მუშაობს **Canvas** ელემენტებთან, მაშასადამე შეუძლებელია 2D ფიგურების მართვა;

Selenium IDE–ის არ შეუძლია ფაილების ატვირთვა, რადგან Selenium-ით შეუძლებელია Windows ფანჯრების მართვა.

4.4. ელემენტის ლოკატორები და მათი კლასიფიკაცია

ავტომატური ტესტების წარმატებით მუშაობა დამოკიდებულია GUI (Graphical Use Interface) ელემენტების იდენტიფიკაციაზე, მათი განლაგების განსაზღვრაზე, რათა ტესტის შესრულებისას მათზე განხორციელდეს სხვადასხვა ოპერაცია. Selenium-ს გააჩნია სხვადასხვა ინსტრუმენტი რომლებიც განსაზღვრავს ელემენტებზე წვდომის განსხვავებულ გზებს: Name, ID, Link, CSS და Xpath.

Web დეველოპმენტის პროექტში რეკომენდებულია GUI ელემენტებისთვის აღიწეროს ისეთი ატრიბუტები, როგორიცაა: ID, Name, Class. ეს ატრიბუტები იძლევა იმის შესაძლებლობას, რომ აპლიკაცია გახდეს უფრო ტესტირებადი და სტანდარტული გზებით მოხდეს ელემენტებზე წვდომა.

მიუხედავად ამისა პრაქტიკაში გვხვდება ისეთი შემთხვევები, სადაც შეუძლებელია მოცემული ატრიბუტების განსაზღვრა, ან ელემენტს უბრალოდ არ აქვს აღწერილი ID, Name. მსგავს შემთხვევებში იყენებენ CSS და XPath ლოკატორებს [154].

Web გვერდზე არსებული ელემენტის ლოკატორი არის მისი მისამართი, რომლითაც მივმართავთ ელემენტს: ID, name, link, XPath, CSS. ლოკატორები გამოიყენება ტესტის წერისას, რათა მოვახდინოთ საჭირო ელემენტის იდენტიფიკაცია და განლაგების განსაზღვრა. Selenium ტესტში ლოკატორის საშუალებით მივმართავთ ელემენტს მასზე გარკვეული ურთიერთქმედების მიზნით. ჩვენ უკვე გავუშვით წარმატებული ტესტები, რომლებიც იყენებდა ლოკატორებს. ავტომატური ტესტირებისას, იმ HTML ელემენტებს, რომლებთანაც ვურთიერთქმედებთ ავტომატურად, უმჯობესია მინიჭებული ჰქონდეს ID და Name [42].

შემდგომ ჩვენ განვიხილავთ:

- ელემენტების განლაგებას ID-ით;
- ელემენტების განლაგებას Name-ით;
- ელემენტების განლაგებას Link-ით;
- ელემენტების განლაგებას XPath-ით;
- ელემენტების განლაგებას CSS-ით.

- *CSS (Cascading Style Sheet)* არის ტექსტის სტილური გაფორმების ენა, რომლის საშუალებით განსაზღვრავენ Web გვერდზე HTML ელემენტების წარმოდგენის სტილს. CSS-ის საშუალებით HTML დოკუმენტი შეგვიძლია წარმოვადგინოთ სხვადასხვა სტილით. იგი საშუალებას იძლევა დავწეროთ ტექსტის ფორმატირების, ელემენტების წარმოდგენის სტილი ცალკე .css ფაილში. ზოგადად, Web გვერდზე ელემენტების წარმოდგენის სტილი შენახულია გარე .css ფაილებში. ტექსტის გარე გაფორმება საშუალებას გვაძლევს მარტივად ვცვალოთ მთლიანი საიტის სტილი, ერთი მარტივი ფაილის ცვლილებით.
- *XPath* არის XML ტეგების ენა, რომელიც გამოიყენება კვანძების მოსანიშნად XML-ში. სხვა სიტყვებით რომ ვთქვათ, იგი არის ენა, რომელიც გამოიყენება XML დოკუმენტიდან ინფორმაციის ამოსაღებად. მას გააჩნია თავისი სინტაქსი, რომლის საშუალებითაც მოძრაობს XML ხისებური სტრუქტურის დოკუმენტებში. Xpath-ს, მსგავსად XML-ისა, აქვს ხისებური სტრუქტურა, რათა იმოძრაოს XML ხის კვანძებზე.

განვიხილოთ მარტივი XML დოკუმენტი (ნახ.4.12):

```
<PostAdr residential="true">
  <name title="Mr.">
    <first>Aaron</first>
    <last>Bartell</last>
  </name>
  <street>123 Center Rd</street>
  <cty>Mankato</cty>
  <state>MN</state>
  <zip>56001</zip>
  <phone>123-123-1234</phone>
  <phone>321-321-4321</phone>
</PostAdr>
```

ნახ.4.12. XML დოკუმენტი

აქ მოცემულ XML დოკუმენტში first ელემენტის შესაბამისი Xpath-ი იქნება: **/PostArd/name/first**. როგორც ვხედავთ XPath-ის აგებისას, მშობელი კვანძიდან შვილობილ კვანძზე გადასვლა ხდება "/" სიმბოლოთი. თუ XML დოკუმენტში გვაქვს იდენტური ტეგები (მაგალითად, phone), მაშინ იდენტიფიკაცია ხდება ტეგების ინდექსირების საშუალებით. ინდექსირება იწყება 1-დან.

მოცემულ მაგალითში 321-321-4321 ტელეფონის ნომრის შესაბამისი XPath არის: **/PostArd/phone** [154].

განვიხილოთ რამდენიმე პროგრამული ინსტრუმენტი, ინტერნეტ ბრაუზერის პლაგინები. ისინი დაგვეხმარება იმის გაგებაში, თუ როგორაა განსაზღვრული ელემენტები და მათი ატრიბუტები, DOM (Document Object Model) სტრუქტურა, CSS სტილის ატრიბუტები და სხვ.

კომპიუტერზე დავაყენოთ ისეთი პროგრამული საშუალებები, რომლებიც დაგვეხმარება ლოკატორის აგებაში:

- **Firebug:** https://addons.mozilla.org/en-US/firefox/addon/firebug/

  - Firebug ფაქტობრივად, არის Web დეველოპერების პროგრამული ინსტრუმენტი. იგი საშუალებას იძლევა ვიპოვნოთ Web გვერდის ელემენტები Find ფუნქციის გამოყენებით. Firebug არის Firefox ბრაუზერის პლაგინი;

  - Firebug-ის საშუალებით შესაძლებელია HTML დოკუმენტის ხისებური სტრუქტურით წარმოდგენა;

  - მას აქვს JavaScript-ის REPL (Read Eval Print Loop), რომელიც საშუალებას გვაძლევს გავუშვათ და შევამოწმოთ JavaScript კოდი.

- **Firefinder:** https://addons.mozilla.org/en-US/firefox/addon/firefinder-for-firebug/

პროგრამული ისტრუმენტი, Web გვერდზე XPath და CSS ტესტირებისთვის. იგი Firebug-ის დანამატია.

- **IE Developer.** იგი Firebug-ის ანალოგია, რომელიც ინტეგრირებულია Internet Explorer-ში და მისი გაშვება შესაძლებელია F12 ღილაკით.
- **Google Chrome Developer**

არის Firebug-ის ანალოგი, Google Chrome ბრაუზერის ინსტრუმენტი, რომლის საშუალებითაც შესაძლებელია ელემენტების პოვნა web გვერდზე, მათი Xpath-ის განსაზღვრა და ა.შ. [42].

ჩამოთვლილი ინსტრუმენტები გვეხმარება ელემენტის ლოკატორის დეტალების განსაზღვრასა და ატრიბუტების პოვნაში. ისინი Web გვერდზე არსებულ ინფორმაციას წარმოადგენს იერარქიულ ხეში. მათი საშუალებით შესაძლებელია ელემენტის ნებისმიერი ტიპის ლოკატორის განსაზღვრა ტესტ-სკრიპტებში გამოყენების მიზნით.

როგორც უკვე განვიხილეთ, თუ მიმდინარე web გვერდზე გვაქვს ელემენტის პოვნის პრობლემა, Selenium IDE-ს Target ველში შეგვიძლია ჩავწეროთ ელემენტის დასახელება (ელემენტის ლოკატორი), დავაჭიროთ Find ღილაკს და web გვერდის ელემენტი წამიერად განათდება.

განვიხილოთ მაგალითი თუ როგორ შეიძლება ელემენტის ლოკატორის პოვნა მოცემულ გვერდზე Firebug-ის საშუალებით. ცხადია Firefox-ზე დაინსტალირებული უნდა გვქონდეს Firebug. განვიხილოთ Firebug-ის გამოყენების მაგალითი:

1) Firefox-ში გაუშვით Firebug – დავკლიკოთ მარჯვენა ზედა კუთხეში მოთავსებული „ხოჭოს" პიკტოგრამა;

2) გამოსულ ფანჯარაზე დავაჭიროთ შემდეგ ღილაკს:  ;

3) მივიტანოთ მაუსის კურსორი აქტიური გვერდის რომელიმე ელემენტთან;

4) გადავატაროთ მაუსი სხვადასხვა ელემენტებს.

როგორც ვხედავთ Firebug ანათებს მაუსით მითითებული ყველა ელემენტის მონაცემებს, სადაც შეგვიძლია ვნახოთ არჩეული ელემენტის სხვადასხვა ატრიბუტი: ID, Name, class, value, text და სხვ. ელემენტის ატრიბუტების გამოყენებით შესაძლებელია ლოკატორის აგება ტესტ-სკრიპტებისთვის, რათა მათი გამოყენებით მოხდეს ელემენტების იდენტიფიკაცია [42].

ტესტ–სკრიპტში ლოკატორად ID ატრიბუტის გამოყენება არის ყველაზე კარგი არჩევანი, რათა განისაზღვროს ელემენტის ადგილმდებარეობა მიმდინარე web გვერდზე. რეკომენდებულია, რომ დეველოპერებმა უზრუნველყონ ელემენტის id ატრიბუტის გაწერა, რომელიც იქნება უნიკალური ყველა ელემენტისთვის. ასეთი id ატრიბუტის არსებობა უზრუნველყოფს მის ზუსტ და საიმედო განლაგებას.

DOM-ის დამუშავების დროს ბრაუზერი იყენებს id-ის, როგორც პრივილეგირებულ გზას, რომ მოხდეს ელემენტის მიკვლევა, ამიტომ იგი არის ყველაზე სწრაფი ლოკატორი. თუ web ელემენტს გადავაადგილებთ მიმდინარე გვერდზე და ლოკატორად განსაზღვრულია id, ტესტი მაინც იმუშავებს.

ტესტი იმუშავებს იმიტომ, რომ განხილულ ელემენტს ვწვდებით არა მისი ადგილმდებარეობის მიხედვით, არამედ id-ით.

ელემენტის ლოკატორად id ატრიბუტის გამოყენება არის *ყველაზე კარგი სტრატეგია*, მაგრამ ხშირ შემთხვევაში შეუძლებელია id–ის გამოყენება შემდეგი მიზეზების გამო:

- გვერდზე მოცემულ ყველა ელემენტს არ აქვს id ატრიბუტი;
- id ატრიბუტის მნიშვნელობა გენერირდება დინამიკურად.

აღნიშნულ შემთხვევებში იყენებენ name ლოკატორებს. განსხვავებით id-სგან, name ატრიბუტი შეიძლება არ იყოს უნიკალური. მიმდინარე გვერდზე შესაძლებელია არსებობდეს რამდენიმე ერთნაირი name ატრიბუტის მქონე ელემენტი. მსგავს სიტუაციაში Selenium-ი მიმართავს პირველ ნაპოვნ ელემენტს, რომელიც შეიძლება არ იყოს ტესტის სამიზნე ელემენტი. ეს გამოიწვევს ტესტის ჩავარდნას.

შევნიშნოთ, რომ არ არის აუცილებელი ელემენტს გააჩნდეს name ატრიბუტი, ისევე როგორც id.

id ლოკატორის ანალოგიურად, თუ მოცემულ ელემენტს გადავაადგილებთ მიმდინარე web გვერდზე, ტესტი მაინც იმუშავებს. ტესტი იმუშავებს იმიტომ, რომ გვერდის ელემენტს მივმართავთ არა მისი მდებარეობის მიხედვით, არამედ name ატრიბუტით.

როგორც უკვე აღვნიშნეთ, მოცემულ გვერდზე შეიძლება არსებობდეს ორი ერთიდაიმავე name-ს მქონე ელემენტი. მოცემულ სიტუაციაში ვიყენებთ ე.წ. ფილტრს,

რომლითაც ვახდენთ საჭირო ელემენტის იდენტიფიკაციას: Target ველში ვწერთ ისეთ ლოკატორს, რომელიც განსხვავებულია მეორე მსგავსი სახელის მქონე ელემენტის ლოკატორისგან. მაგალითად [42]:

name=verifybutton value=chocolate;

Selenium ტექნოლოგიით Web გვერდზე არსებულ ლინკზე მიმართვა ხდება ლინკის ტექსტის საშუალებით. თუ დავაკვირდებით ზემოთ განხილულ მაგალითებს, დავინახავთ, რომ ლინკის ლოკატორს აქვს შემდეგი ფორმატი: link=LinkText, მაგალითად, LinkText=Chapter2.

არსებობს ლინკები, რომელთა ტექსტი გენერირდება დინამიკურად. ასეთ შემთხვევაში ლინკის ლოკატორს აგებენ ნაწილობრივი ტექსტით, კერძოდ, ლინკის ტექსტის სტატიკური ნაწილით. თუ დინამიკურად გენერირებად ლინკის არ გააჩნია სტატიკური ტექსტი, მაშინ ლოკატორად იყენებენ სხვა ალტერნატიულ საშუალებებს: id, name, CSS, Xpath [42].

როგორც უკვე აღვნიშნეთ, XPath არის XML ტეგების ენა, რომელიც გამოიყენება კვანძების მოსანიშნად XML-ში. ყველა თანამედროვე ინტერნეტ ბრაუზერს აქვს Xpath-ის მხარდაჭერა, რადგან DOM-ში HTML გვერდი წარმოდგება როგორც XHTML დოკუმენტი. XPath ენა ბაზირებულია XML ხისებურ სტრუქტურაზე და გააჩნია შესაძლებლობა იმოძრაოს ხის გარშემო, მონიშნოს კვანძები სხვადასხვა კრიტერიუმების საშუალებით.

ელემენტის XPath ლოკატორი ძალიან მოქნილია და წარმოადგენს ერთ-ერთ ბოლო შემოთავაზებულ ლოკატორ–

სტრატეგიას, თავისი ნელი შესრულებით. მისი ნელი მუშაობა განპირობებულია იმით, რომ ელემენტის ძებნისას DOM იერარქიულ ხეში შვილ კვანძზე გადასვლა ხდება მშობელი კვანძის გამოყენებით და ეს პროცესი გრძელდება მანამ, სანამ არ მივაღწევთ საძიებო კვანძს [159,160].

შესაძლებელია xpath ლოკატორის ოპტიმიზაცია მის კვანძებში ელემენტის web ატრიბუტების მითითებით.

Selenium ელემენტის xpath-ს აქვს ფორმატი: /xpath. მაშასადამე Selenium-ში xpath ლოკატორი მიიღება ჩვეულებრივ xpath-ს წინ დამატებული '/' სიმბოლო. ანუ Firebug-ის მიერ გენერირებულ XPath-ს წინ უნდა დაემატოს ერთი დახრილი ხაზი '/'.

Web ტექნოლოგიებში არის შემთხვევები, როდესაც შეუძლებელია ელემენტს მივანიჭოთ სტატიკური ატრიბუტები, მაგალითად, თუ მის დასახელებას ვაგებთ მონაცემთა ბაზიდან მიღებული მონაცემით, ატრიბუტების მნიშვნელობები გაწერილია დინამიკურად და იცვლება გვერდის განახლებასთან ერთად და სხვ.

არის შემთხვევები, სადაც ელემენტის დასახელების მხოლოდ ნაწილი არის დინამიკური. XPath ტექნოლოგიას შეუძლია მიმართოს გვერდის ელემენტს ატრიბუტის ნაწილობრივი მნიშვნელობით. განვიხილოთ XPath ფუნქციები, რომლებიც უზრუნველყოფს ელემენტებზე წვდომას ნაწილობრივი მნიშვნელობებით: starts-with(), ends-with() და contains().

- starts-with() ფუნქცია აიდენტიფიცირებს ელემენტს საწყისი ლოკალური მნიშვნელობით. მაგალითად, თუ

ელემენტის ID არის ctrl_12, სადაც ctrl არის ლოკალური და 12 დინამიკური მნიშვნელობა, აღნიშნული ფუნქცია განსაზღვრავს ელემენტის ადგილმდებარეობას ctrl მნიშვნელობით შემდეგნაირად:

```
//input[starts-with(@id, 'ctrl_')];
```

- ends-with() ფუნქცია აიდენტიფიცირებს ელემენტს ბოლო ლოკალური მნიშვნელობით. მაგალითად, თუ ელემენტის ID არის a_1_userName, სადაც userName არის ლოკალური, მოცემული ფუნქცია განსაზღვრავს ელემენტის ადგილმდებარეობას _ userName მნიშვნელობით:

```
//input[ends-with(@id, '_userName')];
```

contains() ფუნქცია აიდენტიფიცირებს ელემენტს ატრიბუტის დასახელების შემცველი მნიშვნელობით, ქვე–ტექსტით. მაგალითად, თუ ელემენტის ID არის

```
panel_login_userName_textfield,
```

მოცემული ფუნქცია განსაზღვრავს ელემენტს userName ქვე–ტექსტით:

```
//input[contains(@id, 'userName')];
```

საბოლოოდ შევნიშნოთ რომ Xpath-ს აქვს მრავალი წესი, ფუნქცია, სინტაქსი, რომ მიმართოს გვერდის ნებისმიერ ელემენტს. ერთიდაიგივე ელემენტის xpath შეიძლება აიგოს მრავალნაირად, თითოეულ კვანძში ატრიბუტების დამატებით, კვანძების ინდექსირებით, ქვე–ტექსტებით და სხვ.

სხვა ლოკატორებთან შედარებით Selenium-ის XPath ლოკატორის ერთადერთ ნაკლად შეიძლება განვიხილოთ

მისი ნელი შესრულება. ეს ნაკლი განსაკუთრებით შესამჩნევია IE-ბრაუზერზე (Internet Explorer).

CSS ლოკატორით ელემენტების ადგილმდებარეობის განსაზღვრა ხდება ელემენტის სტილით. ანალოგიურად Xpath-ისა, DOM იერარქიულ ხეში CSS ლოკატორის აგება ხდება ხის კვანძების გამოყენებით. მშობელი კვანძიდან შვილობილ კვანძზე გადავდივართ '>' სიმბოლოთი.

ერთიდაიმავე ელემენტის CSS ლოკატორი შეიძლება აიგოს მრავალნაირად, მაგალითად:

```
css=input[id='but1'];
css=input#but1;
css=input [id='but1'][value='Button with ID'];
```

CSS ლოკატორებს აქვს ნაკლი, რაც ვლინდება მათ ელემენტის სტრუქტურისადმი დამოკიდებულებაში. თუ სტრუქტურა შეიცვლება, მსგავსად xpath-სა, ლოკატორი ვეღარ იპოვნის ელემენტს და ტესტი ჩავარდება [42].

4.5. Selenium ტესტებში JavaScript-ის გამოყენება

JavaScript (JS) არის მსოფლიოში ერთ-ერთი ყველაზე პოპულარული ენა [161]. JavaScript არის სკრიპტების ენა, რომლის საშუალებით შესაძლებელია დინამიკური სკრიპტების დაწერა HTML კოდში. მას აღიქვავს ყველა თანამედროვე ინტერნეტ-ბრაუზერი. JavaScript და Java არის ორი ერთმანეთისგან აბსოლიტურად განსხვავებული ენა თავისი დიზაინით და კონცეფციით [162]. მათ აქვთ ერთმანეთისგან მთლიანად განსხვავებული სემანტიკა. JS

სინტაქსი უფრო ახლოს დგას C ენასთან. JAVA არის უფრო კომპლექსური პროგრამირების ენა, რომელიც Sun-ის მიერაა შექმნილი. ორიგინალური JavaScript–ი შექმნილია Netscape-ს მიერ.

JavaScript-ში, ისევე როგორც უმრავლეს სკრიპტულ ენებში, ცვლადის ტიპს განსაზღვრავს მისი მნიშვნელობა. მაგალითად, ცვლადი x შეიძლება განისაზღვროს როგორც number მასზე რამე რიცხვის მინიჭებით, შემდეგ იგივე ცვლადი გახდეს string ტიპის, მასზე ტექსტური მნიშვნელობის მინიჭებით და ა.შ. [154].

ქვემოთ განვიხილავთ თუ როგორ შეიძლება გამოვიყენოთ JavaScript ტესტ-სკრიპტებში. Selenium ტესტში რომ განვსაზღვროთ რაიმე JavaScript კოდი, ამისთვის უნდა გამოვიყენოთ შემდეგი აღწერის ბლოკი.

JavaScript { javascript operators; **}**

სისტემურ ფრჩხილებში { } თავსდება JavaScript ოპერატორი, ან ოპერატორთა მიმდევრობა, ერთმანეთისგან წერტილ-მძიმით გამოყოფილი. Selenium ტესტებში JS ოპერატორები სრულდება ჩვეულებრივი მიმდევრობით და შედეგი ბრუნდება ბოლო ოპერატორის საშუალებით, მაგალითად:

```
var  a, b ;
a = 2 ; b = 5 ;
a + b ;
```

განხილული ფრაგმენტი Selenium-ს შედეგად დაუბრუნებს 7-ს.

Selenium სისტემა დაწერილია JavaScript ტექნოლოგიით. ამის გამო შესაძლებელია Web გვერდზე მანიპულირება JS სტანდარტული ფუნქციებით. ეს გულისხმობს იმას, რომ თუ Selenium ინსტრუმენტი არ გვთავაზობს საჭირო ფუნქციას, შესაძლებელია ტესტებში წმინდა JavaScript-ის გამოყენება.

Selenium ტესტ-სკრიპტებში JS ფუნქციების გამოყენებას მიმართავენ მაშინ, როდესაც დასატესტირებელია რთულად ტესტირებადი სცენარები. თუ ტექნოლოგია არ არის თავსებადი JavaScript-თან, მაშინ Selenium უძლურია მოცემული სცენარის რეალიზაციისთვის, ამ შემთხვევაში მიმართავენ უფრო მძლავრი ინსტრუმენტების გამოყენებას, როგორებიცაა: CodedUI და Test Complete [163].

4.6. Selenium ბრძანებების გაფართოება და დამატებითი ფუნქციები

განვიხილოთ თუ როგორ შეიძლება Selenium ბრძანებათა სისტემის გაფართოება, რადგან Web ტესტირებისას არის შემთხვევები, სადაც Selenium სტანდარტული ბრძანებები არ არის საკმარისი.

Selenium ტექნოლოგია დეველოპერებს აძლევს შესაძლებლობას, რომ განსაზღვრონ და შექმნან ახალი ფუნქციები – მსგავს სამ სვეტიან ფორმატში. იმის გამო რომ Selenium სისტემა დაწერილია JavaScript-ზე, ამიტომ დამატებითი ფუნქციების რეალიზება შესაძლებელია მხოლოდ JavaScript-ენის საშუალებით.

წარმოვიდგინოთ შემთხვევა, როდესაც ვიყენებთ ერთიდაიმავე JavaScript კოდის ფრაგმენტს სხვადასხვა ტესტში:

type | locator | javascript{ ... }

ვთქვათ სცენარის შესრულებისას გვინდა მიმდინარე თარიღის გამოთვლა და საიტის რამდენიმე გვერდზე ჩაწერა "dd/mm/yyyy" ფორმატით.

დასმულ ამოცანას გადავჭრით JavaScript ოპერატორების გამოყენებით, ტესტ–სკრიპტში JS პროგრამული ფრაგმენტის რეალიზებით. თუ მოგვიანებით აღმოვაჩენთ, რომ JS ფრაგმენტის რეალიზაციისას დაშვებულია შეცდომა, მაშინ საჭირო იქნება ყველა იმ ტესტის ცვლილება, სადაც გამოყენებულია მოცემული JS ფრაგმენტი. ეს მიდგომა, როგორც პროგრამული დეველოპმენტიდან არის ცნობილი, არის მცდარი, რადგან ამ დროს მიმდინარეობს ერთიდაიგივე შესწორების მრავალჯერადი გამეორება [154].

აღნიშნული პრობლემის გადასაჭრელად Selenium გვთავაზობს ფუნქციების აღწერას, რომელებსაც გამოვიძახებთ ნებისმიერ ტესტ-სკრიპტში. წარმოდგენილი პრაქტიკული მაგალითისთვის აღვწერთ typeCurrentDate ფუნქციას, რეალიზაციას გავაკეთებთ JavaScript ოპერატორებით და ტესტ-სკრიპტში გამოვიძახებთ მისი სახელის (typeCurrentDate) საშუალებით. აღწერილი სტრატეგია გაცილებით მოქნილია, რადგან დეფექტის პოვნის შემთხვევაში საჭირო იქნება მხოლოდ აღწერილი ფუნქციის ერთჯერადი გასწორება და არა სხვადასხვა ადგილას ერთიდაიმავე შესწორებების შეტანა [154].

მომხმარებლის მიერ რეალიზებულ ფუნქციებს უწოდებენ Selenium გაფართოებად ბრძანებებს. ისინი ინახება ცალკე ფაილში. გაფართოებადი ბრძანებების გამოყენების დროს

საჭიროა მათი ფაილის მისამართი მივაბათ Selenium IDE-ს ან Selenium RC-ს. აღნიშნული ფაილები წარმოადგენს JavaScript ფაილებს .js გაფართოებით. js ფაილში იწერება მომხმარებლის მიერ განსაზღვრული ფუნქცია, რეალიზებული JavaScript სინტაქსით.

Selenium გაფართოებადი ბრძანებები იწერება პროტოტიპული ენით. იმისთვის რომ განვსაზღვროთ დამატებითი ფუნქცია, მისი აღწერის ბლოკი უნდა იყოს შემდეგი სახის:

```
Selenium.prototype.doFunctionName = function()
{
   . . .
}
```

„do" იწერება ფუნქციის დასახელების წინ, რომელიც Selenium-ს მიუთითებს, რომ ეს ფუნქცია შეიძლება გამოყენებულ იქნას როგორც ჩვეულებრივი Selenium ბრძანება.

დავუშვათ, რომ გვინდა გავტესტოთ აპლიკაციის რაიმე ფუნქცია, რომელიც მოითხოვს შემთხვევით აღებული რიცხვების ჩაწერას „ტექსტბოქსში". ზოგადად, შემთხვევით აღებულ რიცხვებს, ე.წ Random-ით მიღებულ რიცხვით მნიშვნელობებს დიდი გამოყენება აქვს ტესტირებისას.

მაგალითად, კალკულატორის ტესტირების დროს, პროგრამაში შესატანი მნიშვნელობების განსაზღვრის დროს და ა.შ. მოცემული ამოცანა რომ გავამარტივოთ, ამისათვის საჭიროა შევქმნათ Selenium-ის დამატებითი ფუნქცია (მაგალითად, storeRandom) და მისი მუშაობის შედეგი შევინახოთ Selenium-ია რაიმე ცვლადში [42].

აღნიშნული დამატებითი ბრძანების რეალიზაციისას დაგვჭირდება ფუნქციის არგუმენტების გამოყენება.

ფუნქციის დასახელებაში მოცემულ ცვლადებს ეწოდება ფუნქციის არგუმენტები. ფუნქციის განსაზღვრისას მის დასახელებაში მოცემული პირველი არტგუმენტი ასოცირდება „Target" ველთან, ხოლო მეორე არგუმენტი „Value" ველთან.

განვიხილოთ აღწერილი ამოცანის რეალიზება დამატებითი ფუნქციის საშუალებით:

1) ავამუშავოთ ტექსტური რედაქტორი და გავხსნათ უკვე შექმნილი user-extensions.js ფაილი;

2) შევქმნათ ახალი ფუნქცია სახელწოდებით storeRandom, რომელიც უნდა გამოიყურებოდეს შემდეგნაირად:

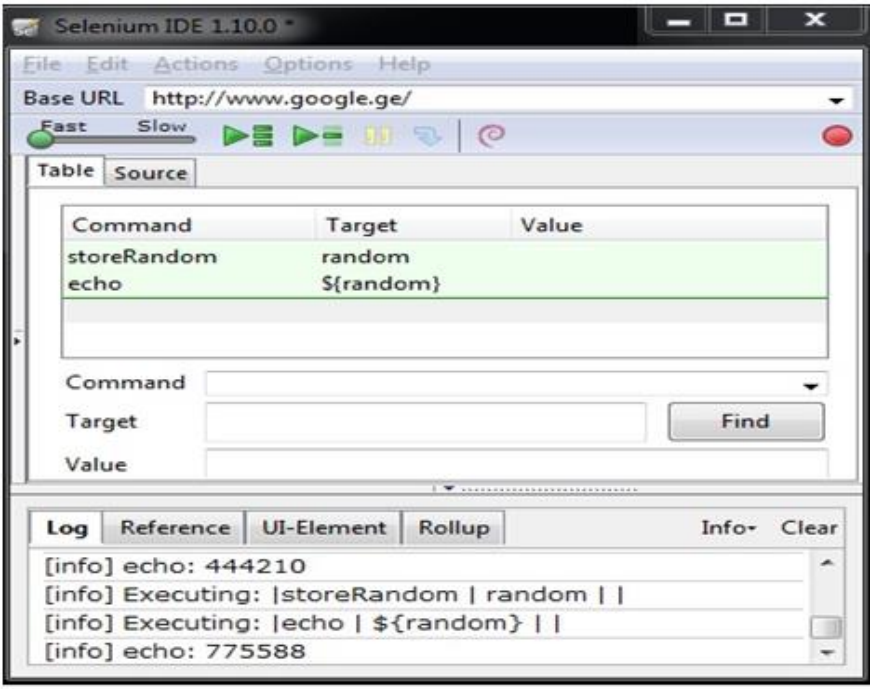
```
Selenium.prototype.doStoreRandom = function(variableName)
{
  random = Math.floor(Math.random() * 1000000);
  storedVars[variableName] = random;
}
```

1) შევინახოთ ფაილი და გადავტვირთოთ Selenium IDE;

2) Selenium IDE-ში შევქმნათ ბრძანება storeRandom და გადავცეთ რაიმე ცვლადი, რომელსაც მიენიჭება ჩვენ მიერ განსაზღვრული ფუნქციის მუშაობის შედეგი;

3) შევქმნათ echo ბრძანება, რათა დავინახოთ ფუნქციის მუშაობის შედეგი.

4.13 ნახაზზე მოცემულია დამატებითი ფუნქციის მაგალითი Selenium სისტემაში.



ნახ.4.13. Selenium IDE: დამატებითი ფუნქციის მაგალითი

განხილულ მაგალითში ვნახეთ თუ როგორ შეიძლება აღვწეროთ დამატებითი Selenium ფუნქცია, რომელიც მუშაობის შედეგს დააბრუნებს არგუმენტად გადაცემულ ცვლადში. აღწერილი ფუნქცია Selenium ცვლადზე წვდომისთვის იყენებს storedVars ოპერატორს. თუ დავაკვირდებით ამ ნახაზს, დავინახავთ რომ დამატებთი ბრძანების მუშაობის შედეგი იწერება Log-ში echo ბრძანების საშუალებით [154]. შევნიშნოთ, რომ, თუ დამატებით ფუნქციის აღწერისას javascript კოდში დავუშვით შეცდომა, Selenium IDE-ს გადატვირთვის დროს ამოვარდება შეცდომის აღწერის შეტყობინება.

4.7. Selenium Remote Control (RC) სერვერი

ახლა განვიხილოთ Selenium ტექნოლოგიის სერვერი – *Selenium Remote Control,* მისი დაყენების და გაშვების ინსტრუქცია, არგუმენტები, ზოგადი მახასიათებლები და ა.შ.

Selenium Remote Control (შემოკლებით RC) არის ერთ–ერთი ყველაზე პოპულარული, გამოყენებადი Selenium პროდუქტი, რომელსაც დიდი პრაქტიკული გამოყენება აქვს Web აპლიკაციების ავტომატური ტესტირებისას.

Selenium RC დეველოპერებს საშუალებას აძლევს ტესტე-ბი დაწერონ მათთვის სასურველ დაპროგრამების ენაზე. აქ განვიხილავთ Windows ოპერაციულ სისტემაზე Selenium RC-ს ინსტალაციის ინსტრუქციას, Selenium IDE-ს მიერ შექმნილი სცენარის გაშვებას RC სერვერზე და სხვ. [154-160].

Selenium Remote Control არის დაწერილი Java ტექნოლოგიით, არის jar გაფართოების ფაილი. მაშასადამე, რადგან Java არის *პლატფორმისგან დამოუკიდებელი* პროგრამირების ენა, ამიტომ RC-ს გაშვება შესაძლებელია ნებისმიერ თანამედროვე პლატფორმაზე: Mac, Linux, Windows, Solaris და სხვა [42,154].

Selenium IDE მუშაობს მხოლოდ Firefox (FF) ბრაუზერზე. IDE წარმოადგენს FF ბრაუზერის პლაგინს, რომლის საშუალებით შესაძლებელია ტესტ-სკრიპტების ჩაწერა ავტომატურ რეჟიმში. ჩვენ, როგორც Web დეველოპერებმა, ან ტესტერებმა ვიცით, რომ Web აპლიკაციის მომხმარებლები არ იყენებენ მხოლოდ ერთ ბრაუზერს. ისინი Firefox-ის გარდა შეიძლება იყენებდნენ Internet Explorer-ს, Google chrome-ს, Safari–ს და ა. შ.

აღნიშნულიდან გამომდინარე შეგვიძლია ვთქვათ, რომ Selenium IDE ინსტრუმენტის ძირითად ნაკლოვანებას წარმოადგენს მისი *ბრაუზერზე დამოკიდებულება* (მუშაობს მხოლოდ Firefox-ზე).

Selenium RC ტესტირების ინსტრუმენტის შექმნის მიზანი იყო ტესტების გაშვება სხვადასხვა ბრაუზერზე. ფაქტობრივად, იგი შექმნეს როგორც proxy შუალედური სერვერი აპლიკაციასა და ტესტ-სცენარს შორის. RC შეიცავს Selenium Core-ს, რომელიც უშვებს ბრაუზერს, კითხულობს და ასრულებს **Selense** ბრძანებებს, ინახავს ტესტირების საშედეგო ფაილს (რეპორტს) და სცენარის დასრულებისას „კლავს" ბრაუზერის სესიას.

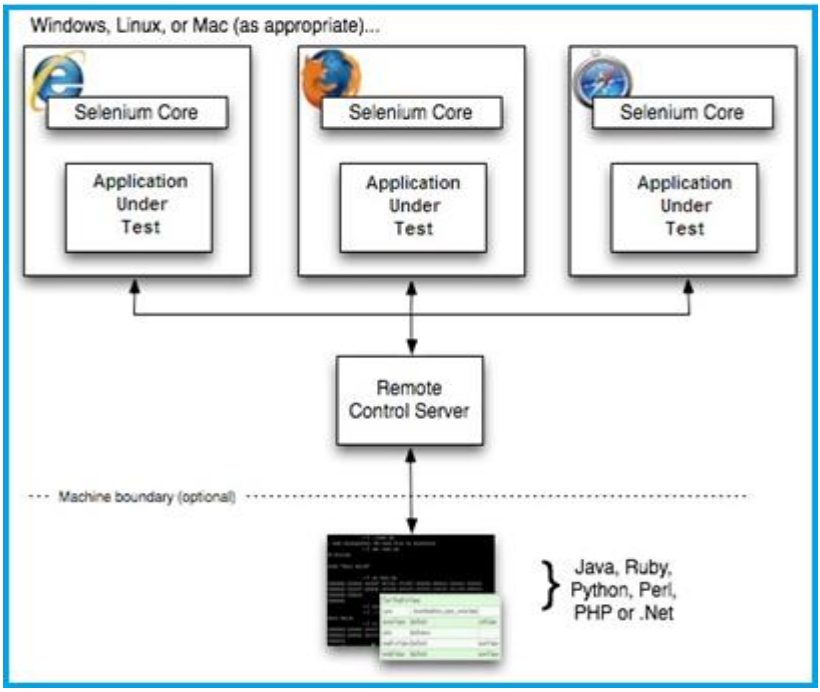
Selenium RC შეიცავს თანამედროვე პროგრამული ენების ბიბლიოთეკას, რომლის გამოყენებით შესაძლებელია ტესტების წერა ისეთ პოპულარულ ენებზე როგორიცაა: **Java, C#, Ruby, Python, Perl, PHP** და სხვა.

ახლო წარსულში დეველოპერები ტესტებს ქმნიდნენ ძირითადად Java და C# ობიექტორიენტირებულ ენებზე. რაც გამოწვეული იყო იმით, რომ აპლიკაციები იქმნებოდა ერთერთ მათგანზე.

ბოლო დროს მიმდინარეობს დინამიკური ენების მძლავრი პოპულარიზაცია, დეველოპერები გადადიან მსგავსი ტიპის ენების გამოყენებაზე. Ruby და Python ყველაზე პოპულარული თანამედროვე ენებია, რომლებზეც მიმდინარეობს პროგრამისტების მიგრაცია [155].

პროგრამული ენის გამოყენება HTML ფორმატის ტესტ–სკრიპტების ნაცვლად უზრუნველყოფს აზრიან, გასაგებ,

პროგრამულად მოქნილ, კარგად სტრუქტურირებად ტესტ-სკრიპტის შექმნას. დაპროგრამების ენით დაწერილ ტესტებში შესაძლებელია ციკლების, პირობითი ოპერატორების გამოყენება, რაც ამარტივებს სკრიპტის შექმნას და გაცილებიდ კითხვადს ხდის ტესტს [154-160]. მოცემულ ნახაზზე წარმოდგენილია Selenium RC სერვერის არქიტექ-ტურის დიაგრამა (ნახ.4.14) [154].



ნახ.4.14. Selenium RC არქიტექტურის დიაგრამა

განვიხილოთ Windows ოპერაციულ სისტემაზე Selenium RC-ს ინსტალაციის ინსტრუქცია. შევნიშნოთ, რომ RC სერვერის გასაშვებად Windows სისტემაზე საჭიროა ჯავას JDK პლატფორმის ინსტალაცია, რადგან RC დაწერილია Java-ზე

და წარმოდგენილია jar ფაილად. თუ კომპიუტერზე არ აყენია Java, წარმოდგენილი ინსტრუქციის შესრულებამდე გადმოწერეთ და დააყენეთ ბოლო ვერსიის Java JDK პლატფორმა.

1) ჩამოწერეთ Selenium RC მისამართიდან:

http://docs.seleniumhq.org/download/

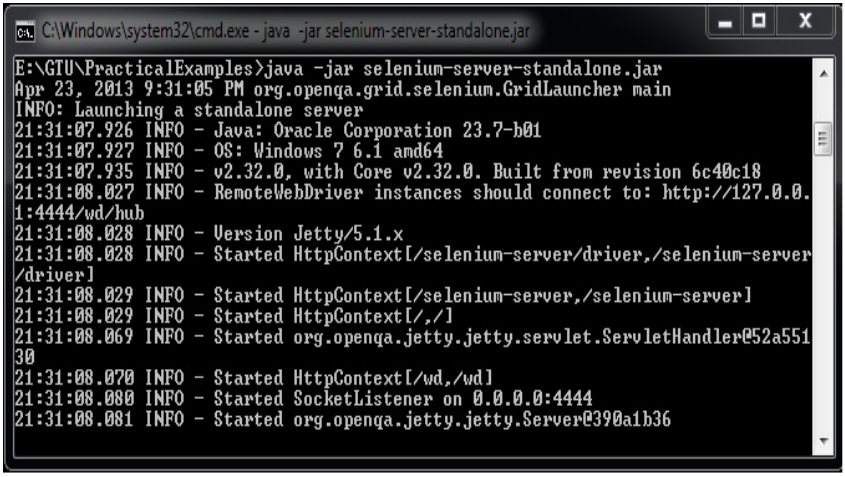
2) ჩამოწერილი selenium-server-standalone.jar ფაილი შეინახეთ ხისტ დისკოზე თქვენთვის სასურველ დირექტორიაში;

3) გაუშვით Windows ბრძანებათა ველი - cmd და კონსოლიდან მიმართეთ იმ დირექტორიას, სადაც შეინახეთ Selenium RC-ს jar ფაილი;

4) cmd-ში ჩაწერეთ და გაუშვით ბრძანება:

java -jar selenium-server-standalone.jar;

5) მე-4 ბიჯზე გამოყენებული ბრძანებით ამუშავდება RC სერვერი (ნახ.4.15).



ნახ.4.15. cmd: Selenium RC-ს გაშვება

მაშასადამე, გადმოვწერეთ Selenium RC სერვერი და გავუშვით Windows ბრძანებათა ველიდან. იგი არის JAR გაფართოების ფაილი, რომლის გასაშვებად გამოიყენება ჩვეულებრივი jar ფაილის გამშვები ბრძანება: **java -jar FileName.jar;**

განხილული მასალიდან ვიცით, რომ Selenium IDE არის Firefox ბრაუზერის პლაგინი. მისი საშუალებით აპლიკაციების ტესტირება შესაძლებელია მხოლოდ Firefox-ზე.

თანამედროვე ტექნოლოგიებში Web-აპლიკაციებთან ურთიერთობისთვის მომხმარებლები იყენებენ სხვადასხვა ინტერნეტ ბრაუზერებს. ბრაუზერისა და ოპერაციული სისტემის კომბინაცია შესაძლოა ნიშნავდეს, რომ პროგრამისტმა ან ტესტერმა გაუშვას ტესტი 9-ჯერ ან მეტჯერ, რათა დარწმუნდეს, რომ აპლიკაცია მუშაობს ყველა პოპულარულ ბრაუზერისა და ოპერაციული სისტემის კომბინაციაზე [154].

დავუშვათ, რომ საჭიროა ტესტ-სცენარების გაშვება კომპიუტერზე, სადაც არ არის დაინსტალირებული Selenium IDE, ან საერთოდ არ გვინდა, რომ Firefox გამოვიყენოთ მატესტირებელ ბრაუზერად. მოცემული ამოცანის გადასაწყვეტად საჭიროა Selenium Remote Control სერვერის გამოყენება.

Selenium RC-ზე ტესტ-სუიტის (სცენარის) გასაშვებად უნდა გამოვიყენოთ –**htmlsuite** არგუმენტი. მისი საშუალებით Selenium-ს ვატყობინებთ, რომ გახსნას html ტიპის ტესტ-სუიტი. გაშვების დროს ასევე საჭიროა მივუთითოთ ტესტ-სუიტის ადგილმდებარეობა დისკოზე, საშედეგო ფაილის დასახელება და ჩაწერის ადგილი. ასევე საჭიროა საწყისი URL

მისამართისა და ბრაუზერის დასახელების განსაზღვრა. საბოლოოდ, Selenium RC სერვერის გამშვები ბრძანება cmd-ში შეიძლება წარმოვადგინოთ შემდეგი ფრაგმენტით:

```
java -jar selenium-server-standalone.jar -htmlsuite *firefox
http://book.theautomatedtester.co.uk
c:\path\to\testuite.html c:\path\to\resultName.html
```

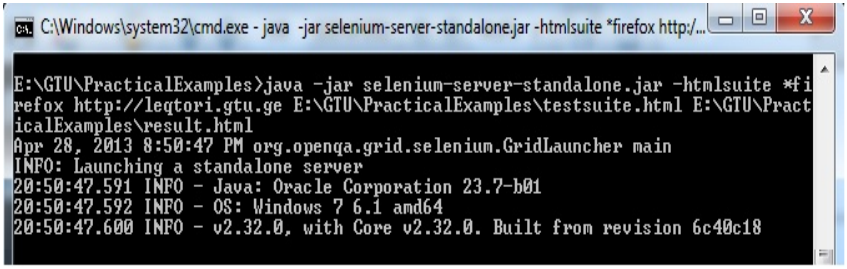
განვიხილოთ Selenium IDE-ს საშუალებით შექმნილი რომელიმე სცენარის Selenium RC-ზე გაშვების პრაქტიკული მაგალითი:

1) გაუშვით Windows ბრძანებათა ველი (cmd) და კონსოლიდან მიმართეთ იმ დირექტორიას, სადაც შეინახეთ Selenium RC-ს jar ფაილი;

2) cmd–ში ჩაწერეთ შემდეგი ბრძანება:

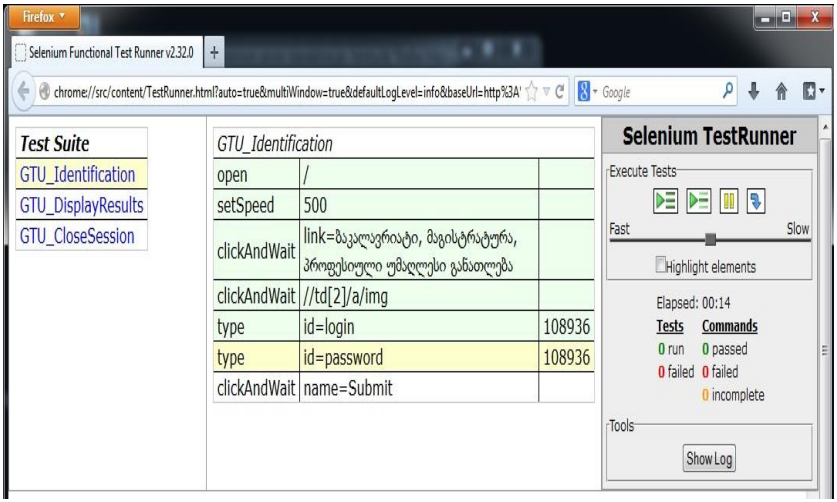
```
java -jar selenium-server-standalone.jar -htmlsuite *firefox
http://leqtori.gtu.ge E:\GTU\PracticalExamples\testsuite.html
E:\GTU\PracticalExamples\result.html
```

მოცემული ბრძანების შესრულების შედეგად გაეშვება Selenium RC სერვერი (ნახ.4.16) [154].



ნახ.4.16. cmd: Selenium RC-ზე სცენარის გაშვება

მოცემული ტესტი დაიწყებს გაშვებას 2 ფანჯარაში. პირველი ფანჯარა არის Selenium Core ფრეიმვორკი, რომლის მარცხენა მხარეს მოთავსებულია ჩვენი ტესტ-სუიტი (ტესტ-სკრიპტების ჩამონათვალი), ფანჯრის ცენტრალურ ნაწილში წარმოდგენილია ტესტ-სკრიპტში მოცემული ბიჯები მიმდევრობით, ხოლო მის მარჯვენა მხარეს მოცემულია ტესტის მუშაობის შედეგი (ნახ. 4.17).

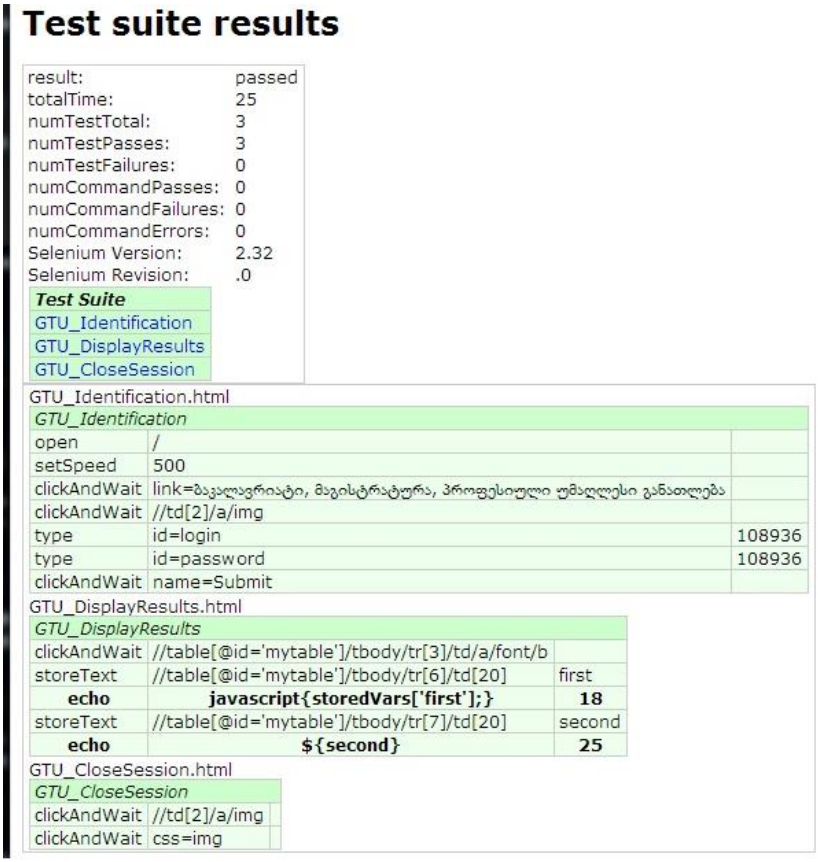


ნახ.4.17. Selenium Core: ტესტ-სკრიპტის შესრულების პროცესი

მეორე ფანჯარა ჩვეულებრივი Firefox ბრაუზერია, რომელშიც ავტომატურად სრულდება სცენარში გაწერილი მოქმედებები. განხილულ მაგალითში Selenium IDE ტესტები გავუშვით Firefox ბრაუზერზე. cmd-ში გამოყენებული ბრძანებით განვსაზღვრეთ ბრაუზერი (Firefox) და გადავეცით საწყისი URL მისამართი (http://leqtori.gtu.ge). ასევე გადავეცით საშედეგო ფაილის ადგილმდებარეობა, თუ სად

და რა დასახელებით უნდა შეინახოს საშედეგო (რეპორტ) ფაილი.

როდესაც ტესტი დაასრულებს მუშაობას, Selenium RC დააგენერირებს HTML ფორმატის საშედეგო ფაილს, რომელშიც ასახული იქნება თუ რომელმა სკრიპტმა გაიარა ტესტი და რომელი სკრიპტი „ჩავარდა“. 4.18 ნახაზზე წარმოდგენილია Selenium RC-ს მიერ გენერირებული საშედეგო ფაილი.



Test suite results

result:	passed
totalTime:	25
numTestTotal:	3
numTestPasses:	3
numTestFailures:	0
numCommandPasses:	0
numCommandFailures:	0
numCommandErrors:	0
Selenium Version:	2.32
Selenium Revision:	.0

Test Suite
GTU_Identification
GTU_DisplayResults
GTU_CloseSession

GTU_Identification.html

GTU_Identification		
open	/	
setSpeed	500	
clickAndWait	link=ბაკალავრიატი, მაგისტრატურა, პროფესიული უმაღლესი განათლება	
clickAndWait	//td[2]/a/img	
type	id=login	108936
type	id=password	108936
clickAndWait	name=Submit	

GTU_DisplayResults.html

GTU_DisplayResults		
clickAndWait	//table[@id='mytable']/tbody/tr[3]/td/a/font/b	
storeText	//table[@id='mytable']/tbody/tr[6]/td[20]	first
echo	**javascript{storedVars['first'];}**	**18**
storeText	//table[@id='mytable']/tbody/tr[7]/td[20]	second
echo	**${second}**	**25**

GTU_CloseSession.html

GTU_CloseSession	
clickAndWait	//td[2]/a/img
clickAndWait	css=img

ნახ.4.18. Selenium RC: ტესტის რეპორტ ფაილი

შევნიშნოთ, რომ თუ რომელიმე სკრიპტი „ჩავარდა“, ანუ ვერ გაიარა ტესტი, იგი საშედეგო ფაილში მოიცემა წითელი ფერით.

Internet Explorer არის ერთ-ერთი ყველაზე პოპულარული ბრაუზერი მომხმარებლებთა შორის. იგი მოყვება Microsoft ოპერაციულ სისტემებს. მიმდინარე მომენტისთვის Windows ოპერაციულ სისტემებს იყენებს მსოფლიოს კომპიუტერების 90%. აღნიშნულიდან გამომდინარე საჭიროა ჩვენ, როგორც ტესტერებმა ან პროგრამისტებმა უზრუნველვყოთ პროგრამული კოდის მუშაობა ბოლო ვერსიის IE ბრაუზერზე მაინც. თუმცა არსებობენ ადამიანები, რომლებიც ჯერ კიდევ იყენებენ ძველი ვერსიის ბრაუზერს – IE5.

თანამედროვე ინტერნეტ სამყაროში ერთ-ერთი გავრცელებადი და პოპულარული ბრაუზერია Google Chrome. ლოგიკურია პრაქტიკაში გაჩნდეს რომელიმე Web აპლიკაციის ტესტირების მოთხოვნა ისეთ პოპულარულ ბრაუზერებზე, როგორიცაა Google Chrome და Internet Explorer [42,154-160, 164].

Selenium RC სერვერის საშუალებით შესაძლებელია Google Chrome-ს გაშვება Firefox-ის ანალოგიურად, თუ გამშვებ ბრძანებაში *firefox-ის ნაცვლად დავწერთ *googlechrome-ს.

მაშასადამე, გამშვები ბრძანების მარტივი ცვლილებით შესაძლებელია სცენარების სხვადასხვა ბრაუზერზე გაშვება. Chrome ბრაუზერზე სცენარის გაშვება განხორციელდება ტესტის ცვლილების გარეშე.

ეს არის Selenium ტექნოლოგიის ერთ-ერთი მძლავრი მახასიათებელი, რის გამოც იგი ითვლება *ერთ-ერთ ფავორიტ უფასო მატესტირებელ ტექნოლოგიად !*

ზუსტად ანალოგიურად, Selenium RC სერვერის საშუალებით შესაძლებელია IE-ის გაშვება Google Chrome-ის ანალოგიურად, თუ გამშვებ ბრძანებაში *googlechrome-ის ნაცვლად დავწერთ *iexplore-ს.

მაშასადამე შეგვიძლია ვთქვათ, რომ Selenium RC სერვერის გამოყენებით ტესტები გავუშვით ისეთ თანამედროვე ბრაუზერებზე როგორიცაა: FF, IE და Google Chrome [154]. განხილული ინტერნეტ ბრაუზერების გარდა არსებობს სხვა ბრაუზერებიც, მაგალითად, Opera, Konquerer, Safari და სხვა. RC-ზე რომ გავუშვათ სხვა ბრაუზერები, უნდა ვიხელმძღვანელოთ შემდეგი სიით:

- ***firefox**
- ***mock**
- ***firefoxproxy**
- ***pifirefox**
- ***chrome**
- ***iexplore**
- ***firefox3**
- ***safariproxy**
- ***googlechrome**
- ***konqueror**
- ***firefox2**
- ***safari**
- ***piiexplore**
- ***firefoxchrome**
- ***opera**
- ***iehta**
- ***custom**

თუ რომელიმე ინტერნეტ ბრაუზერი არ არის მოცემული სიაში, მაშინ მისი გაშვება შესაძლებელია *custom ბრძანებით და ბრაუზერის გამშვები გზის მითითებით [154].

განვიხილოთ Selenium RC-ს კიდევ ერთი პოპულარული არგუმენტი **–port**. რადგან Selenium RC მოქმედებს როგორც

proxy სერვერი ტესტ-სცენარსა და აპლიკაციას შორის, ამიტომ მას ესაჭიროება რაღაც პორტი, რომ დაელოდოს ბრძანებებს.

ამისთვის იგი „გაჩუმების“ პრინციპით (default) იყენებს 4444 პორტს. იმ შემთხვევაში თუ მოცემული პორტი დაკავებულია, ან სერვისი ჩარჩენილია, შესაძლებელია სხვა პორტის გამოყენება შემდეგი ფრაგმენტით: **-port <port number>,** რაც საშუალებას გვაძლევს გავუშვათ RC სერვერი სხვ პორტზე პრობლემების გარეშე [42]. გარდა ჩამოთვლილი არგუმენტებისა, არსებობს RC-ს კიდევ მრავალი სხვა არგუმენტი:

- **-setTimeOut <timeout value>** მოცემული არგუმენტით განისაზღვრება Selenium RC-ს მუშაობის მაქსიმალური დრო მილისეკუნდებში (1 წმ = 1000 მს). გაჩუმების პრინციპით RC–ის მაქსიმალური მუშაობის დრო არის 30 წუთი;
- **-singleWindow** ეს არგუმენტი უზრუნველყოფს Selenium RC-ს გაშვებას ერთ ფანჯარაში. მოცემული RC-ს გაშვების ერთფანჯრიანი რეჟიმი გაცილებით სწრაფია, ამიტომ ძირითადად მას იყენებენ აპლიკაციების IE-ზე ტესტირებისას. IE-ზე ტესტირება გაცილებით ნელია ვიდრე FF–ზე.

ზოგადად Selenium RC სერვერის არგუმენტთა სიის ნახვა შესაძლებელია შემდეგი cmd ბრძანებით: **java -jar selenium-server-standalone.jar -h**

Windows ოპერაციულ სისტემაზე Selenium RC სერვერის გაშვება შესაძლებელია batch გამშვები ფაილის განსაზღვრითაც. მსგავსი ტიპის ფაილებს აქვს .bat გაფართოება. bat ფაილის განსასაზღვრად იყენებენ ნებისმიერ ტექსტურ რედაქტორს და მასში წერენ ჩვეულებრივ RC-ს გამშვებ

ბრძანებას (cmd ბრძანება). სწორად განსაზღვრული bat ფაილის გაშვება შესაძლებელია მაუსის ორი კლიკის საშუალებით, რომლის შედეგად გაეშვება Selenium RC სერვერი.

ამგვარად, ჩვენ განვიხილეთ Selenium RC სერვერი, რომელიც არის დაწერილი Java ტექნოლოგიით, ე.ი არის *პლატფორმისგან დამოუკიდებელი.*

RC-ს საშუალებით შესაძლებელია ტესტ-სკრიპტების წერა ნებისმიერ თანამედროვე პროგრამირების ენაზე, მაშასადამე იგი არის *დაპროგრამების ენისგან დამოუკიდებელი.*

ასევე RC სერვერით შესაძლებელია ნებისმიერი ბრაუზერის გაშვება, ანუ იგი ასევე არის *ბრაუზერისგან დამოუკიდებელიც.*

ჩამოთვლილი უპირატესობები განაპირობებს მის პოპულარობას, თუმცა მასაც გააჩნია ტექნოლოგიური ნაკლოვანებები, რომელთა თავიდან აცილების მიზნით შეიქმნა RC-ს შემდგომი განვითარება Selenium WebDriver [42].

4.8. Selenium WebDriver ინსტრუმენტი

თანამედროვე ინფორმაციულ ტექნოლოგიებში მიმდინარეობს Web აპლიკაციების ყოველღიური განვითარება. იქმნება მრავალი სამომხმარებლო Web აპლიკაცია, რაც განპირობებულია Web სისტემების პოპულარობით.

Web ტექნოლოგიების განვითარების პარალელურად ვითარდება ავტომატური მატესტირებელი ტექნოლოგიებიც. იქმნება ავტომატური მატესტირებელი ფრეიმვორკები,

რომლებიც უზრუნველყოფს საიტების მუშაობის შემოწმებას, ტესტირებას.

Selenium IDE და Selenium RC წლების განმავლობაში გამოიყენება ბრაუზერის მართვის ავტომატიზაციისთვის. როდესაც Selenium შეიქმნა „Jason Huggins"-ის მიერ, წყვეტდა პრობლემებს, რომლებიც მიიღებოდა ბრაუზერში მომხმარებლის ურთიერთქმედებით. ის კარგი ავტომატური ტესტირების ფრეიმვორკია, მაგრამ შემოიფარგლება JavaScript–ით.

მობილურ მოწყობილობებზე და HTML5 ტექნოლოგიაზე გადასვლის შემდეგ, ცხადი გახდა რომ Selenium RC სერვერი ვეღარ ასრულებდა თავის მოთხოვნას: ბრაუზერის ავტომატიზაცია, მომხმარებლის მოქმედებების შესრულება.

Simon Stewart-ს სურდა შეექმნა განსხვავებული ტექნოლოგია, რომელიც გადაწყვეტდა აღნიშნულ პრობლემებს. კომპანია ThoughtWorks-ში ყოფნისას მან დაიწყო WebDriver პროექტზე მუშაობა, რომლის პირველი რეალიზაცია განხორციელდა 2007 წელს [160, 165].

WebDriver-ის რეალიზაციის განხორციელების შემდეგ ცხადი გახდა მნიშვნელოვანი სხვაობა მას და Selenium RC-ს შორის, რაც გამოიხატებოდა მათ API-ში (Application Programming Interface). Selenium RC-ს ჰქონდა ლექსიკონზე ბაზირებული API, ხოლო WebDriver-ს კი – უფრო ობიექტზე ორიენტირებული API.

თავდაპირველად WebDriver გვთავაზობდა მხოლოდ Java–ს მხარდაჭერას, ხოლო RC გვთავაზობდა მრავალი ენის მხარდაჭერას. მათ შორის ასევე იყო მნიშვნელოვანი ტექნოლოგიური სხვაობა: Selenium Core (რაზეც RC არის

დაფუძნებული) რეალიზებულია JavaScript-ით, რომელიც ეშვება ბრაუზერში, ხოლო WebDriver ბრაუზერს მართავს ოპერაციული სისტემა [42, 154-160, 164].

2009 წლის აგვისტოში გამოაცხადეს ამ ორი პროექტის გაერთიანება, რომელსაც უწოდეს Selenium WebDriver (იგივე Selenium-2). ამჟამად WebDriver-ს გააჩნია შემდეგი ენების მხარდაჭერა: Java, C#, Python და Ruby. იგი გვთავაზობს Google Chrome, Firefox, Internet Explorer, Opera, Android და iPhone ბრაუზერების მხარდაჭერას.

WebDriver მუშაობს ოპერაციული სისტემის დონეზე, რაც იმას ნიშნავს, რომ იგი ინტერნეტ ბრაუზერს ბრძანებებს უგზავნის ოპერაციული სისტემიდან. ეს ფრეიმვორკი არ შემოიფარგლება JavaScript-ით, როგორც ეს იყო Selenium-თან მიმართებაში. მისი საშუალებით შესაძლებელია მივწვდეთ ფაილურ სისტემას, მაშასადამე შესაძლებელია ფაილის ატვირთვის ტესტის რეალიზება. ტესტების გაშვებისას WebDriver არ მოითხოვს პროქსი სერვერის დაყენებას, როგორც ეს იყო Selenium RC-ს დროს.

WebDriver-ის უარყოფით მხარედ უნდა აღინიშნოს ის, რომ იგი მოითხოვს ახალ რეალიზაციას ბრაუზერის ყოველი ახალი ვერსიის გამოსვლის პარალელურად. ეს არის გამოწვეული იმით, რომ ყოველ ბრაუზერს გააჩნია თავის გამშვები წერტილი ოპერაციულ სისტემაში [42].

მაშასადამე, Selenium WebDriver არის Web აპლიკაციების ავტომატური ტესტირების ფრეიმვორკი, რომელსაც შეუძლია გაუშვას ტესტები სხვადასხვა ბრაუზერზე (ნახ.4.19) და სხვადასხვა მოწყობილობაზე.

ნახ.4.19. WebDriver–ის ბრაუზერებთან თავსებადობა

WebDriver აგრეთვე საშუალებას გვაძლევს შევქმნათ ტესტები თანამედროვე პროგრამირების ენის გამოყენებით, მაშასადამე:

- შესაძლებელია პირობითი ოპერატორების გამოყენება: if-then-else ან switch-case;
- შესაძლებელია ციკლების გამოყენება: for, while, do-while

WebDriver-ის მიერ მხარდაჭერილი პროგრამირების ენებია: Java, .Net, PHP, Python, Perl და Ruby. ჩამოთვლილთაგან

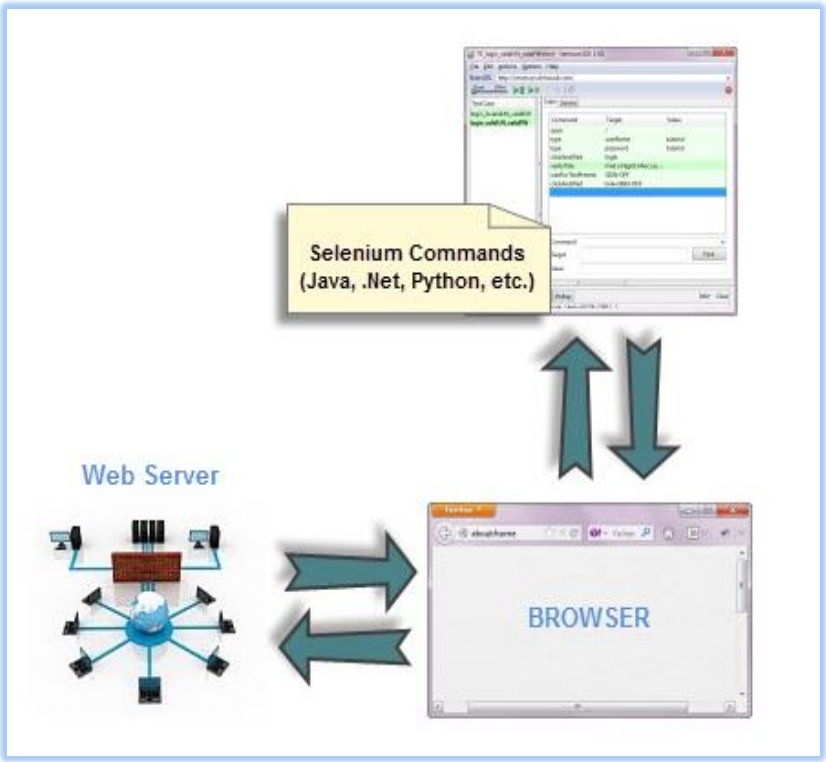
არ არის საჭირო ყველა ენის ცოდნა. საკმარისია ჩამოთვლილ-თაგან ვიცოდეთ ერთი რომელიმე ენა მაინც. ჩვენ განვიხილავთ Java პროგრამირების ენას და Eclipse პროგრამირების გარემოს.

თუ დავაკვირდებით **WebDriver**-ს და **Selenum RC**-ს, ორივეს აქვს სხვადასხვა ბრაუზერის და პროგრამირების ენის მხარდაჭერა. მაშასადამე, რით განსხვადება ისინი ?

1) ***არქიტექტურა:*** WebDriver–ის არქიტექტურა Selenium RC-ს არქიტექტურასთან შედარებით გაცილებით მარტივია. იგი ბრაუზერის მართვას ახორციელებს ოპერაციული სისტემის საშუალებით. მის გასაშვებად საკმარისია პროგრამირების რომელიმე გარემო და ინტერნეტ ბრაუზერი. WebDriver-ის არქიტექტურის დიაგრამა ნაჩვენებია 4.20 ნახაზზე [42].

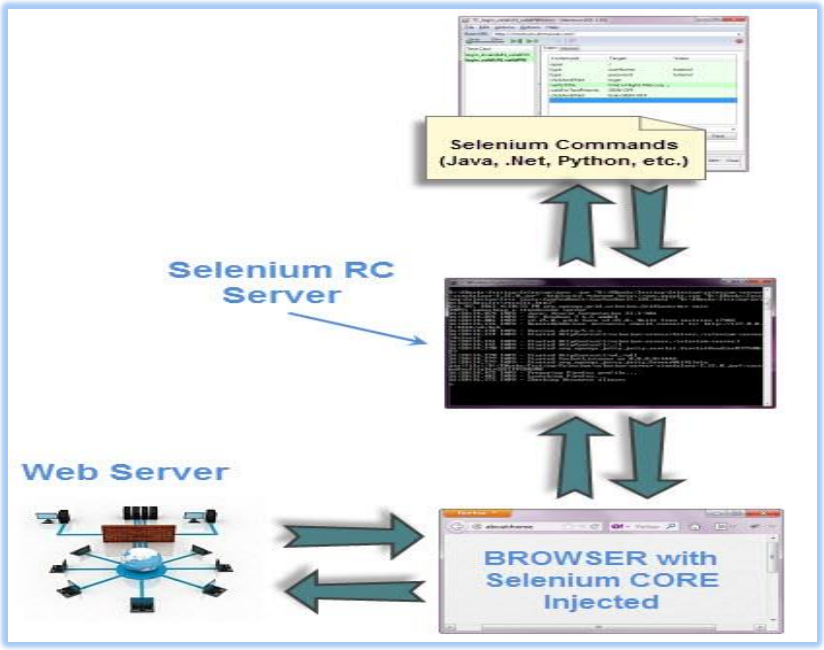
Selenium RC არის უფრო რთული არქიტექტურის:

- თავდაპირველად, ტესტირების დაწყებისთვის საჭიროა ცალკე აპლიკაციის გაშვება, რომელსაც ეწოდება Selenium RC სერვერი;
- Selenium RC სერვერი მოქმედებს როგორც „შუაკაცი" Selenium ბრძანებებსა და ბრაუზერს შორის;
- როდესაც დაიწყება ტესტირება, Selenium RC სერვერს ბრაუზერში „შეჰყავს" Selenium Core;
- ინსტრუქციების მიღების შემდეგ Selenium Core უშვებს მათ როგორც Javascript ბრძანებებს.



ნახ.4.20. WebDriver–ის არქიტექტურის დიაგრამა

Selenium RC სერვერის არქიტექტურის დიაგრამა შეიძლება წარმოვადგინოთ 4.21 ნახაზით.



ნახ.4.21. Selenium RC–ის არქიტექტურის დიაგრამა

2) **სიჩქარე:** WebDriver არის გაცილებით სწრაფი ვიდრე Selenium RC, რადგან იგი პირდაპირ „ელაპარაკება" ბრაუზერს.

Selenium RC არის უფრო ნელი, ტესტების შესრულებისას იგი იყენებს დამატებით Javascript პროგრამას, რომელსაც ეწოდება Selenium Core. Selenium Core არის ინსტრუმენტი, რომელიც მართავს ბრაუზერს [42].

3) **ურთიერთქმედება:** WebDriver გვერდის ელემენტებთან ურთიერთქმედებს უფრო რეალური გზით. მაგალითად, თუ სატესტირებელ გვერდზე გვაქვს disabled „ტექსტ-ბოქსი",

WebDriver-ს, ადამიანის მსგავსად არ შეუძლია მასში ტექსტის ჩაწერა (ნახ.4.22).

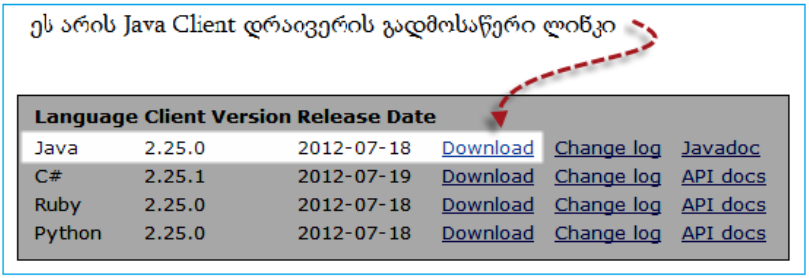
ნახ.4.22. ურთიერთქმედება-1

Selenium Core, ისევე როგორც JavaScript კოდი წვდება disabled ელემენტებს (ნახ.4.23). Selenium ტესტერები წარსულში გამოთქვამდნენ უკმაყოფილებას, რადგან Selenium Core-ს შეეძლო მნიშვნელობის შეყვანა disabled ელემენტში [42,154].

ნახ.4.23. ურთიერთქმედება-2

განვიხილოთ WebDriver–ის ინსტალაცია, მოვახდინოთ მისი კონფიგურაცია Eclipse ჯავა პროგრამირების გარემოში.

ვიგულისხმოთ, რომ უკვე დაინსტალირებული გვაქვს Java პლატფორმა (JDK) და Eclipse. Selenium-ის ოფიციალური საიტიდან გადმოვიწეროთ Java Client დრაივერი (ნახ.4.24).



ნახ.4.24. Java Client დრაივერის ლინკი

გადმოწერის შედეგად მივიღებთ ZIP ფაილს, სახელწოდებით:

selenium-<version number>.zip.

სიმარტივისთვის ამოვაარქივოთ ZIP ფაილი C დისკზე, რის შედეგადაც მივიღებთ შემდეგ დირექტორიას:

C:\selenium-<version number>\.

ეს დირექტორია შეიცავს JAR ფაილებს, რომლებსაც დავაიმპორტირებთ Eclipse-ში.

- **WebDriver-ის კონფიგურაცია Eclipse ინსტრუმენტში:**

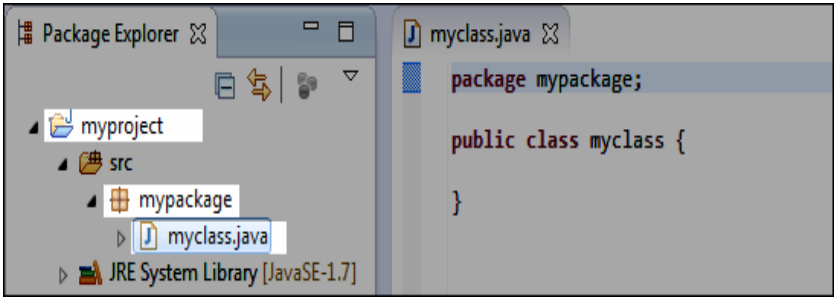
1) გაუშვით პროგრამა Eclipse;

2) შექმენით ახალი პროექტი File -> New -> Java Project. პროექტს სახელად დაარქვით "myproject";

3) ახლად შექმნილ პროექტზე მაუსის მარჯვენა კლიკის საშუალებით გადადით New->Package. პეკიჯს დაარქვით სახელი "mypackage";

4) mypackage-ში შექმენით ახალი Java კლასი მასზე მაუსის მარჯვენა კლიკის საშუალებით და აირჩიეთ New -> Class.

კლასის სახელი იყოს "myclass". თქვენი Eclipse ინსტრუმენტი დაახლოებით უნდა გამოიყურებოდეს 4.25 ნახაზის მსგავსად.



ნახ.4.25. Eclipse გარემო

5) myproject-ზე მაუსის მარჯვენა კლიკით - Properties-ზე;

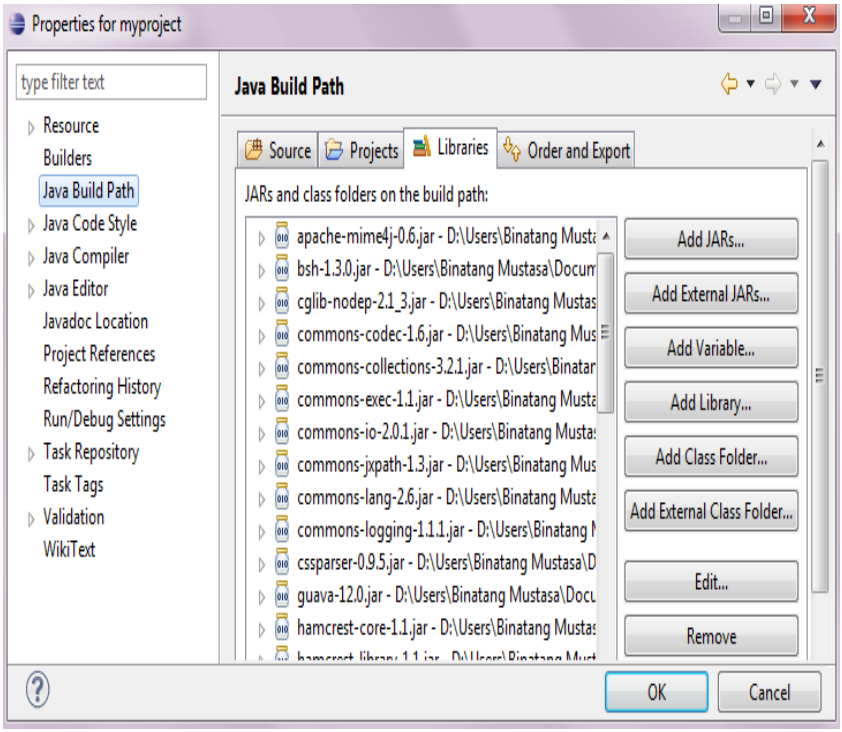
6) დიალოგურ ფანჯარაზე მოვნიშნოთ „Java Build Path";

7) გადავიდეთ Libraries ტაბზე და შემდეგ დავაჭიროთ „Add External JARs...";

8) მივმართოთ C:\selenium-<version number>\ დირექტორიას (სადაც შევინახეთ ამოარქივებული Java Client დრაივერი);

9) მოცემული დირექტორიიდან დავამატოთ ყველა JAR ფაილი. ჩვენი Properties დიალოგური ფანჯარა უნდა გამოიყურებოდეს შემდეგნაირად (ნახ.4.26);

10) დასასრულს დავაჭიროთ OK ღილაკს.



ნახ.4.26. პროექტის თვისებები

ამრიგად, აღწერილი ინსტრუქციით ჩვენს Eclipse პროექტში დავაიმპორტირეთ Selenium ბიბლიოთეკები.

WebDriver-ს აქვს ე.წ. „უჩინარი" ბრაუზერი, სახელწოდებით **HtmlUnit,** რომელიც ყველაზე სწრაფი ბრაუზერია.

იგი არის Java-ზე შექმნილი ბრაუზერი გრაფიკული ინტერფეისის გარეშე. სწორედ ინტერფეისის არარსებობა განაპირობებს ტესტების მაღალი სიჩქარით შესრულებას: Selenium აღარ ელოდება გვერდის ჩამოტვირთვას, ლინკზე გადასვლას, ელემენტის გამოჩენას და სხვ.

HTMLUnit და **Firefox** არის ორი ბრაუზერი, რომელთა ავტომატიზაციას WebDriver ახდენს პირდაპირ, რაც იმას ნიშნავს, რომ არ არის საჭირო ცალკეული დამატებითი კომპონენტების ინსტალაცია. სხვა ბრაუზერებისთვის საჭიროა დამატებითი კომპონენტები, რომელთაც ეწოდება Driver Server [158].

Driver Server განსხვავებულია ყოველი ბრაუზერისთვის. მაგალითად, Internet Explorer-ს აქვს თავის driver server, რომელსაც ვერ გამოვიყენებთ სხვა ბრაუზერისთვის. ქვემოდ მოცემულია ბრაუზერების სია და მათი შესაბამისი driver server:

- Chrome ChromeDriver
- Opera OperaDriver
- Safari SafariDriver
- Internet Explorer Internet Explorer Driver Server

შევქმნათ პირველი WebDriver სკრიპტი. გამოვიყენოთ ჩვენს მიერ შექმნილი კლასი „myclass" და დავწეროთ სკრიპტი რომელიც უზრუნველყოფს:

1. Google საწყისი გვერდის გახსნას;
2. მისი სახელწოდების შემოწმებას;
3. შემოწმების შედეგის ბეჭდას;
4. გაშვებული ბრაუზერის დახურვას.

ქვემოთ მოცემულია აღწერილი სცენარის შესაბამისი ლისტინგი WebDriver ჯავა კოდისთვის:

```
package mypackage;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
```

```
public class myclass {
   public static void main(String[] args) {
// declaration and instantiation of objects/variables
                WebDriver driver = new FirefoxDriver();
                String baseUrl = "https://www.google.com/";
                String expectedTitle = "Google";
                String actualTitle = "";
                // launch Firefox and direct it to the Base URL
                driver.get(baseUrl);
                // get the actual value of the title
                actualTitle = driver.getTitle();
                if (actualTitle.contentEquals(expectedTitle)) {
                    System.out.println("Test Passed!");
                  }
                else {
                    System.out.println("Test Failed");
                }
                // close Firefox
                driver.close();
                // exit the program explicitly
                System.exit(0);
    }
}
```

- **განვიხილოთ რეალიზებული Java კოდი:**

import org.openqa.selenium.WebDriver; აიმპორტებს WebDriver კლასს, driver ობიექტის აღწერის მიზნით.

import org.openqa.selenium.firefox.FirefoxDriver; აიმპორტებს FirefoxDriver კლასს, რომელიც ინიციალიზაციას უკეთებს driver ობიექტს.

თუ თქვენს მიერ აღწერილი ტესტი ასრულებს რთულ მოქმედებებს, უკავშირდება სხვა კლასებს, აკეთებს ბრაუზერის სქრინს ან მანიპულირებს გარე ფაილებზე, აუცილებლად დაგჭირდებათ სხვა პეკიჯების იმპორტი.

WebDriver driver = new FirefoxDriver(); მისი საშუალებით ხდება driver ობიექტის აღწერა, FirefoxDriver კლასი პარამეტრის გარეშე ნიშნავს, რომ გაეშვება Firefox-ის default პროფილი. default Firefox პროფილი იგივეა, რაც Firefox ბრაუზერის გაშვება safe mode-თი (არ შეიცავს არანაირ გაფართოებას).

driver.get (baseUrl); WebDriver-ის **get()** მეთოდი გამოიყენება ბრაუზერის ახალი სესიის წარმოსაქმნელად, რომლის საწყისი URL-ი განისაზღვრება baseUrl ცვლადით.

actualTitle = driver.getTitle(); WebDriver კლასს აქვს **getTitle()** მეთოდი, რომელიც გამოიყენება მიმდინარე გვერდის სათაურის გასაგებად.

```
if (actualTitle.contentEquals(expectedTitle))
    {
        System.out.println("Test Passed!");
    }
  else {
        System.out.println("Test Failed");
     }
```

მოცემული პირობითი ოპერატორი უზრუნველყოფს მიმდინარე სათაურის შედარებას წინასწარ აღწერილ სათაურთან და შედეგად ბეჭდავს შემოწმების შედეგს.

driver.close(); close() მეთოდი გამოიყენება ბრაუზერის ფანჯრის დასახურად.

System.*exit*(0); ასრულებს Java პროგრამის შესრულებას**.** თუ თქვენ გამოიყენებთ მოცემულ ბრძანებას ბრაუზერის დახურვის ბრძანებამდე, მაშინ თქვენი Java პროგრამა დასრულდება და ბრაუზერი დარჩება გახსნილი.

რეალიზებული ტესტის გასაშვებად საჭიროა Eclipse-ს მენიუდან Run -> Run ბრძანებაზე გადასვლა. თუ ყველაფერი შესრულებულია სწორად, Eclipse გაუშვებს ბრაუზერს, გაივლის სცენარს და კონსოლში გამოიტანს "Test Passed!" შედეგს.

// იგივე ტესტის გაშვება IE-ზე:

```
package mypackage;
import java.io.File;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
public class myclass {
     public static void main(String[] args) {
      // declaration and instantiation of objects/variables
     File file = new File("C:\\iedriver\\IEDriverServer.exe");
     System.setProperty("webdriver.ie.driver",
file.getAbsolutePath());
     WebDriver driver = new InternetExplorerDriver();
      //WebDriver driver = new InternetExplorerDriver();
            String baseUrl = "https://www.google.com/";
            String expectedTitle = "Google";
            String actualTitle = "";
              // launch Firefox and direct it to the Base URL
            driver.get(baseUrl);
```

```
        // get the actual value of the title
        actualTitle = driver.getTitle();
        if (actualTitle.contentEquals(expectedTitle)) {
                        System.out.println("Test Passed!");
        }
          else {
                System.out.println("Test Failed");
            }
                // close Firefox
                driver.close();
                // exit the program explicitly
                System.exit(0);       }
}
```

Web გვერდის ელემენტებზე წვდომის გზა (ლოკატორი) მკვეთრად შეცვლილია Selenium2-ში. WebDriver-ს აქვს გაცილებით მდიდარი შესაძლებლობები გვერდის ელემენტის საპოვნელად. იგი ელემენტებს მიმართავს **By** სტატიკური კლასის საშუალებით. By კლასს გააჩნია შემდეგი მეთოდები:

- By.id("ifOfObject")
- By.linkText("TextUsedInTheLink")
- By.partialLinkText("partOfTheLink")
- By.tagName("theTHMLNodeType")
- By.className("cssClassOnTheElement")
- By.cssSelector("cssSelectorToTheElement")
- By.xpath("//Xpath/to/the/element")
- By.name("nameOfTheElement")

როგორც მოცემული სიიდან ჩანს, WebDriver განიხილავს 8 სახის ლოკატორს, მაშასადამე Selenium1-ის ლოკატორთა სიას დაემატა 3 სახის ლოკატორი: **tagName**, **className** და **partialLinkText.**

ბრაუზერის დასახურად WebDriver იყენებს 2 სახის ბრძანებას: close() და quit() (ნახ.4.27).

ნახ.4.27. close() და quit() ბრძანებები

close() ბრძანება დახურავს WebDriver ტესტის მიმდინარე ფანჯარას, ხოლო **quit()** ბრძანება უზრუნველყოფს WebDriver ტესტის მიერ გახსნილი ყველა ფანჯრის დახურვას. quit() ბრძანება ძირითადად გამოიყენება სცენარით გახსნილი pop-up ფანჯრების დასახურად. მაშასადამე, პირველი სურათი ახდენს close() ბრძანების იმიტაციას, ხოლო მეორე quit() ბრძანებისას [42].

განვიხილოთ ფაილის ატვირთვის ავტომატიზაცია bin.ge სერვერზე. ვთქვათ C: დისკოზე გვაქვს შენახული newhtml.html ფაილი, რომელიც WebDriver-ის საშუალებით

უნდა ავტვირთოთ http://bin.ge სერვერზე. აღწერილი სცენარის შესაბამისი Java კოდი მოცემულია ქვემოთ:

```
package mypackage;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Upload {
   public static void main(String[] args) {
                String baseUrl = "http://bin.ge/";
                WebDriver driver = new FirefoxDriver();
                driver.get(baseUrl);
          WebElement uploadElement =
                driver.findElement(By.id("uploadfile_0"));
    // enter the file path onto the file-selection input field
      uploadElement.sendKeys("C:\\newhtml.html");
   // check the "I accept the terms of service" check box
      driver.findElement(By.id("terms")).click();
      // click the "UploadFile" button
      driver.findElement(By.name("send")).click();
      }
   }
```

ქვემოთ აღწერილი კოდით შესაძლებელია youtube web გვერდის ავტომატიზაცია:

```
package mypackage;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.AfterClass;
```

```
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;
public class Upload {
    private WebDriver driver;
    private String baseUrl;
    @BeforeClass
    public void setUp() throws Exception {
      driver = new FirefoxDriver();
      baseUrl = "http://www.youtube.com";
    }
    @Test
    public void test() throws Exception {
      driver.get(baseUrl + "/");
  driver.findElement(By.id("masthead-search-term")).clear();
      driver.findElement(By.id("masthead-search-term"))
       .sendKeys("louis armstrong what a wonderful world");
  driver.findElement(By.id("search-btn")).click();
  driver.findElement(By.xpath("//a[contains(text(),'Louis
          Armstrong  What A  Wonderful World')]")).click();
   Thread.sleep(180000);
  }
    @AfterClass
    public void tearDown() throws Exception {
      driver.quit();
  }
}
```

ამგვარად, განხილულ იქნა Web-აპლიკაციების ავტომატური ტესტირება Selenium ტექნოლოგიით; ავტომატური

ტესტირების გამოყენების სფეროები და მისი შედარება ხელოვნურ ტესტირებასთან, უპირატესობები და ნაკლოვანებები.

Selenium ტექნოლოგიის ზოგადი აღწერა, მისი შექმნის ისტორია და განვითარება. ფართოდ მიმოხილულია Selenium ტექნოლოგიის ძირითადი შემადგენელი მატესტირებელი ინსტრუმენტები ამოცანების პრაქტიკული რეალიზებით.

Selenium RC სერვერი, შექმნილი Java ტექნოლოგიით, არის *პლატფორმისგან, დაპროგრამების ენებისგან და ბრაუზერისგან დამოუკიდებელი*. ჩამოთვლილი უპირატესობები განაპირობებს მის პოპულარობას, თუმცა მასაც გააჩნია ტექნოლოგიური ნაკლოვანებები, რომელთა თავიდან აცილების მიზნით შეიქმნა RC-ს შემდგომი განვითარება Selenium WebDriver.

WebDriver მუშაობს ოპერაციული სისტემის დონეზე, იგი ინტერნეტ ბრაუზერს ბრძანებებს უგზავნის ოპერაციული სისტემიდან. ეს ფრეიმვორკი არ შემოიფარგლება JavaScript–ით, როგორც ეს იყო Selenium-თან მიმართებაში. მისი საშუალებით შესაძლებელია მივწვდეთ ფაილურ სისტემას, მაშასადამე შესაძლებელია ფაილის ატვირთვის ტესტის რეალიზება.

ტესტების გაშვებისას WebDriver არ მოითხოვს პროქსი სერვერის დაყენებას, როგორც ეს იყო Selenium RC-ს დროს.

V თავი

IT-სერვისების პარალელური ტესტირება

ავტომატური ტესტირების ინსტრუმენტი Selenium Grid საშუალებას იძლევა ტესტების გაშვება განვახორციელოთ პარალელურად განსხვავებულ მანქანებზე, ბრაუზერებსა და ოპერაციულ სესტემებზე. მას შეუძლია აგრეთვე განაწილებული ტესტების გაშვება. ტესტირების შესრულების დრო საგრძნობლად მცირდება. მნიშვნელოვანია ტესტ-ქეისების შექმნა Java ტექნოლოგიაზე JUnit და TestNG ფრეიმვორკების საშუალებით. ტესტირების პროცესში შესაძლებელია ბრაუზერში გახსნილი გვერდების სქრინების გაკეთება. ეს მიანიშნებს Selenium ტექნოლოგიის პოპულარობასა და მრავალი პლატფორმის მხარდაჭერაზე. ამ თავში განვიხილავთ პარალელური ტესტირების ძირითად საკითხებს.

5.1 Selenium Grid: პარალელური ტესტირება

განვიხილოთ Selenium ტექნოლოგიის კიდევ ერთი ინსტრუმენტი, კერძოდ, **Selenium Grid**. დავახასიათოთ ზოგადად მისი სტრუქტურა და შემადგენელი კომპონენტები, აღვწეროთ თუ როგორ ხდება Selenium Grid ჰაბის (ცენტრის) ოპერაციულ სისტემაზე დაყენება, Selenium RC სერვერების განსაზღვრა და ტესტების გაშვება Grid-ზე, ტესტების პარალელურ რეჟიმში მუშაობა და ა.შ. [166,167].

Selenium Grid არის ავტომატური ტესტების გასაშვები ინსტრუმენტი, რომლის საშუალებითაც შესაძლებელია ტესტირების პროცესის ჩატარება *განსხვავებულ მანქანებზე,*

სხვადასხვა ბრაუზერებზე, სხვადასხვა ოპერაციულ სესტემებზე პარალელურად. იგი იძლევა დროის ერთიდაიმავე მომენტში რამდენიმე ტესტის გაშვების შესაძლებლობას სხვადასხვა მანქანაზე, ბრაუზერსა და ოპერაციულ სისტემაზე. აგრეთვე Grid-ს შეუძლია განაწილებული ტესტების გაშვება.

ზოგადად ამბობენ, რომ არსებობს ორი ძირითადი მიზეზი რატომაც შეიძლება დაგვჭირდეს Selenium Grid-ის გამოყენება:

- ავტომატური ტესტები რომ გაეშვას სხვადასხვა ბრაუზერზე, ბრაუზერის სხვადასხვა ვერსიებზე, და თვითონ ბრაუზერები გაეშვას განსხვავებულ ოპერაციულ სისტემებზე;
- რომ მოხდეს ავტომატური სცენარების მუშაობის დროის შემცირება.

Selenium Grid გამოიყენება *ოპტიმიზაციის მიზნით*, კრძოდ, ტესტირების შესრულების დროის დასაჩქარებლად. იგი შესრულების სიჩქარეს ზრდის რამდენიმე მანქანის გამოყენებით და მათზე პარალელური ტესტების რეალიზებით. მაგალითად, თუ გვაქვს შესასრულებელი 100 ტესტი, და ჩვენ დავაყენებთ Selenium Grid-ს 4 სხვადასხვა მანქანაზე, მოცემული ტესტების გასაშვებად დაგვჭირდება 4-ჯერ ნაკლები დრო, ვიდრე ტესტების ერთ მანქანაზე მიმდევრობით გაშვებისას.

დიდი ტესტ-სცენარებისათვის და ხანგრძლივი ტესტებისთვის, როგორიცაა მაგალითად, ბევრი მონაცემების ვალიდაცია, პარალელური ტესტირება შეიძლება იყოს მნიშვნელოვანი, დროის დაზოგვის თვალსაზრისით [42].

Selenium Grid-ის სისტემაზე დასაყენებლად საჭიროა დამატებითი ინსტრუმენტის **Apache Ant**-ის გამოყენება. აღნიშნულის გამო ჯერ მოკლედ განვიხილოთ Ant ინსტრუმენტი, მოვახდინოთ მისი ინსტალაცია და შემდგომ დავუბრუნდეთ Grid-ს.

Ant (Another Neat Tool) არის პროგრამული ინსტრუმენტი, რომელიც გამოიყენება პროგრამული უზრუნველყოფის შექმნის პროცესების ავტომატიზაციისთვის [168]. იგი დაწერილია Java ენაზე, იყენებს Java პლატფორმას და არის ერთ-ერთი საუკეთესო ინსტრუმენტი Java პროექტის შესაქმნელად. Ant-ი, პროექტის შექმნის პროცესის განსაზღვრისთვის იყენებს XML ენას. „გაჩუმების“ პრინციპით, XML ფაილის დასახელებაა build.xml.

Ant არის უფასო Apache პროექტი. Windows ოპერაციული სისტემისთვის. მისი გადმოწერა შესაძლებელია შემდეგი მისამართიდან:

http://ant.apache.org/bindownload.cgi

Ant ინსტრუმენტის დაყენების შემდეგ შესაძლებელია Selenium Grid-ის ინსტალაცია. ბოლო ვერსიის Grid–ის გადმოსაწერად გამოიყენეთ მისი ოფიციალური საიტის ლინკი:

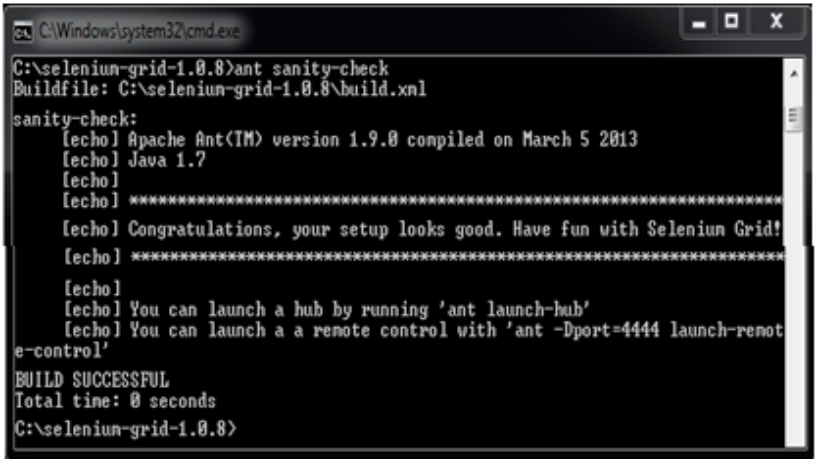
http://selenium-grid.seleniumhq.org/download.html

გადმოწერილი zip გაფართოების ფაილი ამოაარქივეთ და შეინახეთ თქვენთვის სასურველ მისამართზე, ხისტ დისკოზე.

განვიხილოთ Selenium Grid ინსტალაციის ტესტი, რათა შევამოწმოთ რამდენად სწორად გვაქვს შესრულებული

ინსტრუქციები და დავრწმუნდეთ, რომ Grid-ის გასაშვებად გვაქვს ყველა საჭირო ინსტრუმენტი. ამისთვის საჭიროა განვახორციელოთ ე.წ. **sanity-check** ტესტი [42].

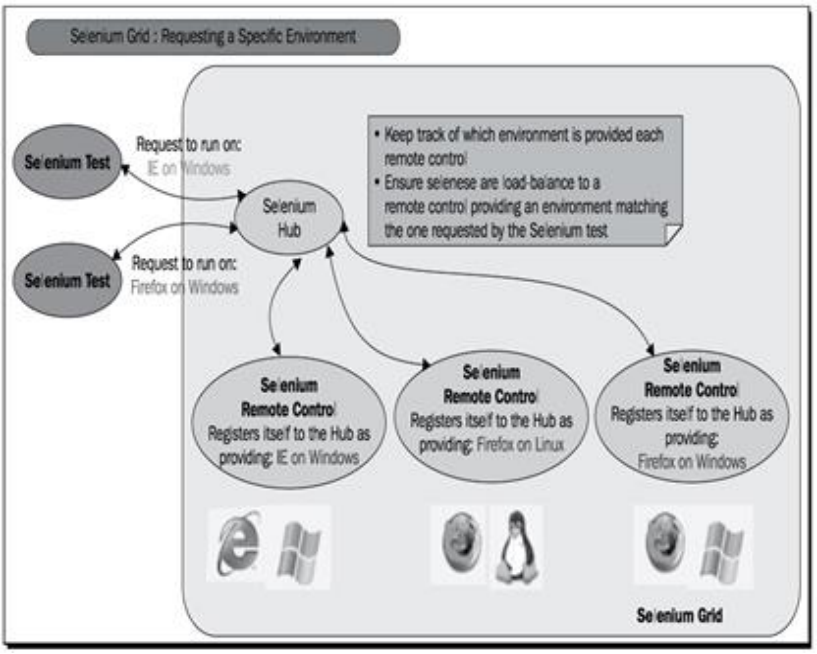
გავხსნათ Windows ბრძანებათა ველი (cmd), კონსოლიდან მივმართოთ Grid დირექტორიას და ჩავწეროთ შემდეგი ბრძანება: **ant sanity-check**. მოცემული ბრძანების წარმატებით შესრულების შემთხვევაში მიიღებთ შემდეგს (ნახ.5.1):



ნახ.5.1. cmd: sanity-check ტესტი

Selenium Grid-ს აქვს ე.წ. „ცენტრალური საწყისი წერტილი", რომელსაც უკავშირდება ტესტები. ამ ცენტრალური ნაწილიდან მიეწოდება ბრძანებები Selenium RC სერვერს. ესი ცენტრი წარმოადგენს Grid-ის ე.წ. **ჰაბს** (Hub) /ნახ.5.2/.

ჰაბს აქვს Web ინტერფეისი, სადაც მოიცემა ჰაბთან დაკავშირებული Selenium RC სერვერები და მათი მდგომარეობები – არის თუ არა ისინი ამუშავებული [42].



ნახ.5.2. Grid Hub დიაგრამა

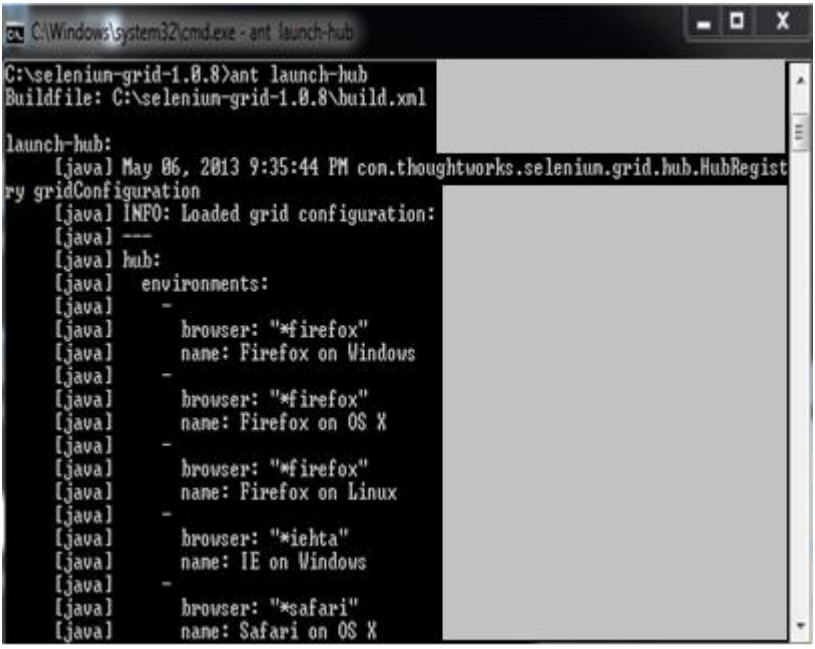
Selenium Grid ჰაბის გასაშვებად საჭიროა ერთი მარტივი cmd ბრძანება:

1) გავხსნათ cmd ფანჯარა და კონსოლიდან მივმართოთ Grid დირექტორიას;

2) გაუშვით შემდეგი ბრძანება: **ant launch-hub;**

3) მოცემული ბრძანების შესრულებისას უნდა მივიღოთ შემდეგი სურათი (ნახ.5.3).

მოცემულ cmd ფანჯრიდან ჩანს, რომ გაშვებულია Selenium Grid Hub–ი. როგორც აღვნიშნეთ, ჰაბს გააჩნია Web ინტერფეისი, ამიტომ მისი გაშვების დანახვა შესაძლებელია ბრაუზერიდანაც.



ნახ.5.3. ჰაბის გაშვება

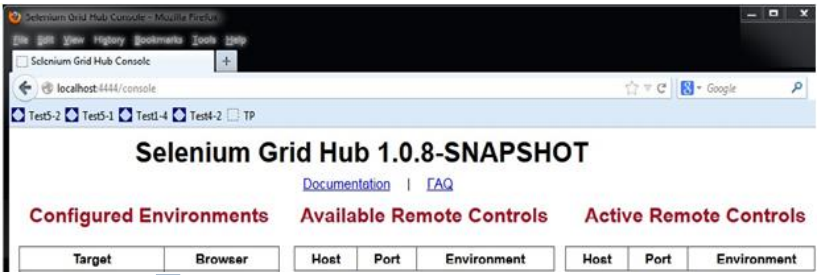
თუ ბრაუზერში გავხსნით შემდეგ URL–ს:

http://localhost:4444/ console,

დავინახავთ შემდეგ Web ინტერფეისს (ნახ.5.4).

მოცემული ინსტრუქციით გავაკეთეთ Selenium Grid ჰაბის გაშვება.

localhost მისამართით ვნახეთ ჰაბის Web ინტერფეისი, რომლის მარცხენა მხარეს მოიცემა „გაჩუმების“ პრინციპით კონფიგურირებული მოწყობილობები, ხოლო მარჯვენა ნაწილში RC კომპონენტების მდგომარეობები – თავისუფალი და აქტიური სერვერები.



Target	Browser
*firefox3	*firefox3
Safari on OS X	*safari
*chrome	*chrome
*firefox2	*firefox2
*firefoxproxy	*firefoxproxy
*pifirefox	*pifirefox
Firefox on Windows	*firefox
*iehta	*iehta
*piiexplore	*piiexplore
*iexploreproxy	*iexploreproxy
*opera	*opera
Firefox on OS X	*firefox
*safariproxy	*safariproxy
*firefox	*firefox
*safari	*safari
*iexplore	*iexplore
*googlechrome	*googlechrome
Firefox on Linux	*firefox
IE on Windows	*iehta

ნახ.5.4. Firefox: ჰაბის Web ინტერფეისი

ახლა, როდესაც წარმატებით გავუშვით Selenium Grid Hub, განვიხილოთ როგორ ხდება Selenium RC სერვერების დამატება ჰაბზე, რის განსახორციელებლადაც საჭიროა კვლავ Ant ბრძანების გამოყენება [42].

Selenium Grid ინსტრუმენტიდან განვახორციელოთ Selenium RC სერვერზე კავშირი და დავარეგისტრიროთ იგი ჰაბზე.

სიმარტივისთვის ბრაუზერად განვიხილოთ Firefox და მივიჩნიოთ რომ ჰაბი და RC სერვერი არის ერთიდაიმავე მანქანაზე.

გამშვებ ბრძანებაში ყოველთვის დაგვჭირდება პორტის ნომრის ხელით გაწერა, რადგან Selenium–ი ვერ ხედავს თავისუფალ პორტებს.

1) გავხსნათ cmd ფანჯარა და კონსოლიდან მივმართოთ Grid დირექტორიას;

2) გავუშვათ შემდეგი ბრძანება: **ant –Dport=5555 launch-remote-control;**

3) მოცემული ბრძანების შესრულების შედეგად მივიღებთ შემდეგ სურათს (ნახ.5.5).

ზემოთ აღვნიშნეთ, რომ Selenium Grid-ს ქსელის საშუალებით შეუძლია სხვადასხვა მანქანაზე განსხვავებულ ოპერაციულ სისტემებზე სცენარების გაშვება.

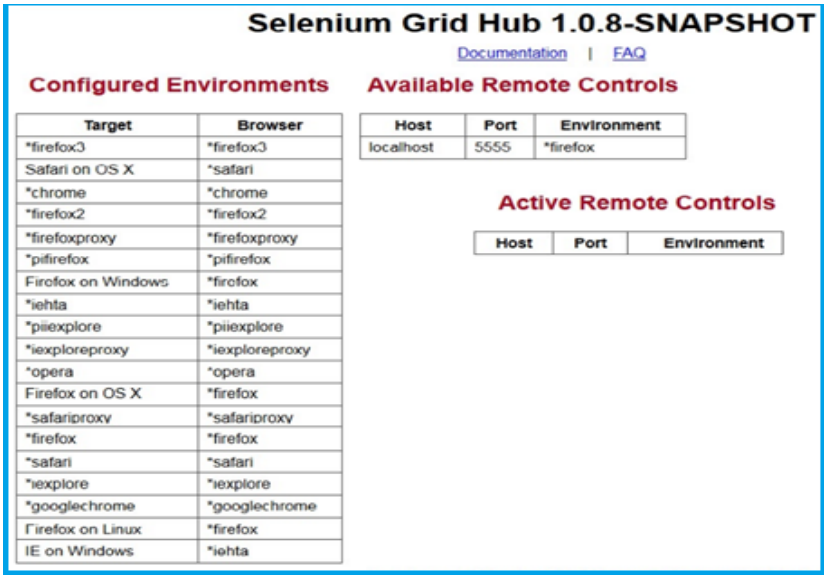
ამის პრაქტიკული რეალიზაციის მიზნით განვიხილოთ თუ როგორ ხდება სხვადასხვა ოპერაციულ სისტემაზე Selenium RC სერვერების გაშვება და მათი მართვა ჰაბის საშუალებით [154].

1) გავხსნათ ახალი cmd ფანჯარა და კონსოლიდან მივმართოთ Grid დირექტორიას;

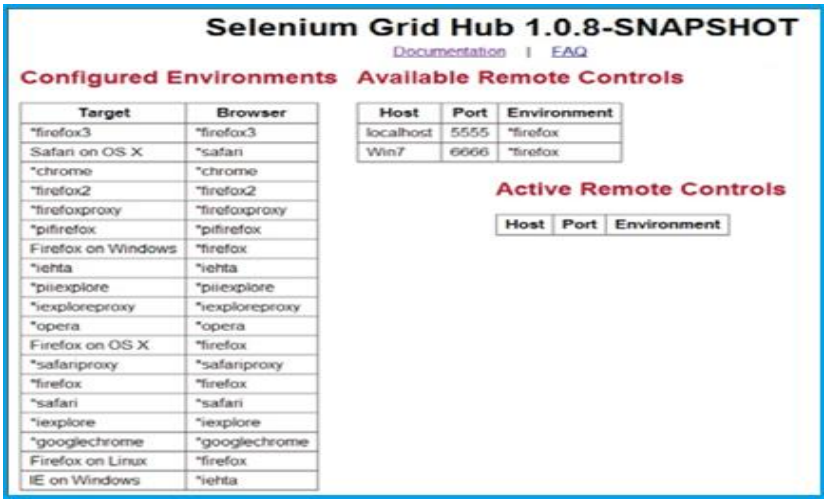
2) გავუშვათ შემდეგი ბრძანება:

ant –Dport=9999 -DhubURL=http://nameofmachine:port -Dhost=nameofcurrentmachine launch-remote-control;

3) მოცემული ბრძანების შესრულების შედეგად უნდა მივიღოთ შემდეგი სურათი (ნახ.5.6).

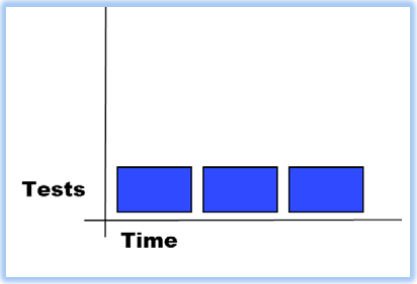


ნახ.5.5. Hub: Selenium RC სერვერის რეგისტრაცია ჰაბზე



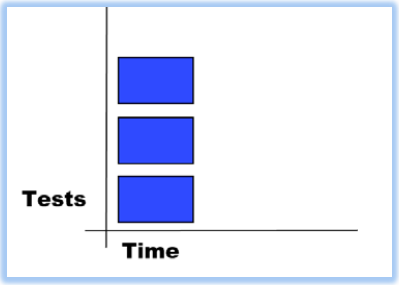
ნახ.5.6. Hub: Selenium RC სერვერის რეგისტრაცია განსხვავებულ მანქანაზე

ტრადიციულად Selenium ავტომატური ტესტები სრულდება იმ თანმიმდევრობით, როგორ მიმდევრობასაც მატესტირებელი ფრეიმვორკი გადაწყვეტს (ნახ.5.7) [166].



ნახ.5.7. მიმდევრობითი ტესტების დროის დიაგრამა

ახლა განვიხილოთ ტესტების შესრულება პარალელურ რეჟიმში, რაც ნიშნავს ტესტების შესრულების პროცესის დაჩქარებას n–ჯერ, სადაც n არის ბრაუზერების რაოდენობა (ნახ.5.8) [42].



ნახ.5.8. პარალელური ტესტების დროის დიაგრამა

მოცემული დიაგრამების პრაქტიკული რეალიზაციისათვის განვიხილოთ ორი, მიმდევრობითი და პარალელური ტესტირების მაგალითი.

მაგალითი 1: მიმდევრობითი ტესტები

განვიხილოთ Grid ინსტრუმენტში რეალიზებული სადემონსტრაციო მაგალითი, სადაც 4 Selenium ტესტი გაეშვება მიმდევრობით RC სერვერზე. ტესტების გაშვება განვიხილოთ Google Chrome ბრაუზერზე [42].

1) გავხსნათ cmd ფანჯარა, მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება: **ant launch-hub;**

2) გავხსნათ ახალი cmd ფანჯარა, კვლავ მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება:

ant –Denvironment="*googlechrome" launch-remote-control

3) გავხსნათ მე–3 cmd ფანჯარა, კვლავ მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება:

ant –Dbrowser="*googlechrome" run-demo-in-sequence

აღწერილი ინსტრუქციის სწორად შესრულების შემთხვევაში Selenium RC სერვერი გაეშვება 4-ჯერ Google Chrome ბრაუზერზე და თითოეული შეასრულებს განსხვავებულ ძებნის ტესტებს.

ყურადღება უნდა მიექცეს მთლიანი ტესტ-სუიტის გაშვების ხანგრძლივობას. სიმართლე რომ ვთქვათ, მოცემული მაგალითით არაფერი ახალი არ გაგვიკეთებია, მსგავს მიმდევრობით ტესტებს ჩვენ მიერ განხილული Selenium RC

სერვერიც აკეთებს. ჩვენთვის საინტერესოა პარალელური ტესტირება, რომელსაც აკეთებს Grid ინსტრუმენტი.

მაგალითი 2: პარალელური ტესტები

განვიხილოთ Grid ინსტრუმენტში რეალიზებული იგივე მაგალითი (მაგალითი 1), სადაც 4 Selenium ტესტი გაეშვება პარალელურად ერთ მანქანაზე. ტესტების გაშვება განვიხილოთ კვლავ Google Chrome ბრაუზერზე.

1) გავხსნათ cmd ფანჯარა, მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება: **ant launch-hub**

2) გავხსნათ ახალი cmd ფანჯარა, კვლავ მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება:

ant –Denvironment="*googlechrome" launch-remote-control

3) გავხსნათ სამი ახალი cmd ფანჯარა, ყოველი მათგანიდან მივმართოთ Grid ინსტრუმენტის დირექტორიას და თითოეულში შესაბამისად ჩავწეროთ შემდეგი ბრძანება:

ant -Denvironment="*googlechrome" -Dport=5556 launch-remote-control

ant -Denvironment="*googlechrome" -Dport=5557 launch-remote-control

ant -Denvironment="*googlechrome" -Dport=5558 launch-remote-control

ჩამოთვლილი ბრძანებების გაშვების შემდეგ ჰაბის Web ინტერფეისზე უნდა გამოჩნდეს ოთხი თავისუფალი RC სერვერი http://localhost:4444/console:

Available Remote Controls

Host	Port	Environment
localhost	5555	*googlechrome
localhost	5556	*googlechrome
localhost	5557	*googlechrome
localhost	5558	*googlechrome

4) გავხსნათ კიდევ ერთი cmd ფანჯარა, კვლავ მივმართოთ Grid ინსტრუმენტის დირექტორიას და გავუშვათ შემდეგი ბრძანება:

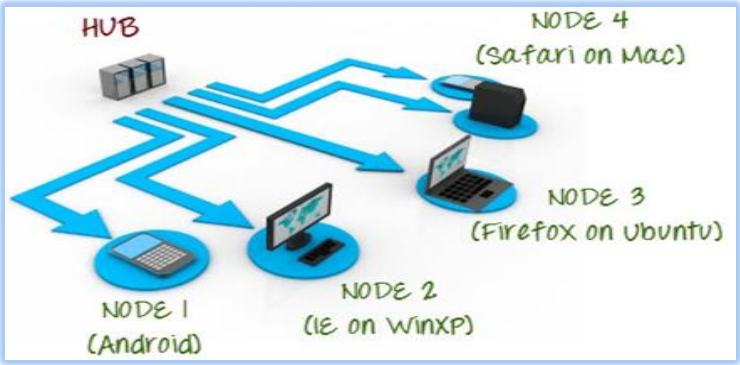
ant –Dbrowser="*googlechrome" run-demo-in-parallel

შეცდომების არარსებობის შემთხვევაში შესრულებაზე გაეშვება 4 Selenium RC სერვერი ერთდროულად, ანუ შესრულდება პარალელური ტესტირება. დროის ერთიდა-იმავე მომენტში გაეშვება სხვადასხვა ტესტი, რაც ცხადია პირველი (მიმდევრობითი) მაგალითის შესრულების სიჩქარეს მნიშვნელოვნად გააუმჯობესებს [42].

5.2. Selenium Grid 2, 3 და 4

განვიხილოთ Selenium Grid-ის ახალი ვერსიები: Grid 2, Grid 3 და Grid 4. ისინი განსხვავდება ჩვენ მიერ განხილული Grid 1.0 ვერსიისგან. დღეისთვის Grid 1.0 ტექნოლოგია უკვე მოძველებულია, რომელსაც Selenium–ი აღარ ავითარებს. 2.0 ვერსიაში Selenium Grid–ი შეუერთდა Selenium RC სერვერს, რაც ნიშნავს იმას, რომ ერთი .jar ფაილის გადმოწერით ჩვენ მივიღებთ Selenium RC სერვერს და Selenium Grid-საც, ორივეს ერთად [154, 169-171].

Selenium Grid–ი იყენებს *ჰაბ–კვანძების* ცნებებს, სადაც *ჰაბს* წარმოადგენს ის მანქანა რომლიდანაც ხდება ტესტების გაშვება, ხოლო გაშვებული ტესტები სრულდება განსხვავებული მანქანების მიერ, რომელთაც ეწოდება *კვანძები* (ნახ.5.9) [42].



ნახ.5.9. Selenium Grid 2.0

ქვემოთ ჩამოთვლილია Grid 1.0-ის და Grid 2.0–ის ძირითადი განსხვავებები [154]:

1) Grid 1.0-ს აქვს თავისი RC სერვერი, რომელიც განსხვავდება Selenium RC სერვერისგან. ისინი 2 სხვადასხვა პროგრამაა;

ამჟამად Grid 2.0 არის Selenium RC სერვერის JAR ფაილთან შეერთებული;

2) Grid 1.0-ის ინსტალაციამდე საჭიროა Apache Ant–ის კონფიგურაცია;

Grid 2.0 არ საჭიროებს Ant ინსტალაციას;

3) Grid 1.0 თავსებადია მხოლოდ Selenium RC ბრძანებებთან / სკრიპტებთან;

Grid 2.0 თავსებადია როგორც Selenium RC-ს, ასევე WebDriver სკრიპტებთან;

4) Grid 1.0-ის ერთ RC სერვერზე შესაძლებელია მხოლოდ ერთი ბრაუზერის ავტომატიზაცია;

Grid 2.0–ის ერთ RC სერვერს შეუძლია 5 ბრაუზერზე მეტის ავტომატიზაცია.

- **რას წარმოადგენს ჰაბი** ?

- ჰაბი არის ცენტრალური ადგილი, საიდანაც ხდება ტესტების გაშვება;
- Grid-ს აქვს მხოლოდ ერთი ჰაბი;
- ჰაბის გაშვება ხდება მხოლოდ ერთ მანქანაზე, ვთქვათ კომპიუტერზე, რომელზეც აყენია Windows და ბრაუზერი IE;
- ავტომატური ტესტების გაშვება შესაძლებელია განხორციელდეს როგორც ჰაბის შემცველ მანქანაზე, ასევე იმ კვანძებზე რომლებიც დარეგისტრირებულია ჰაბზე.

- **რას წარმოადგენს კვანძები** ?

- კვანძები არის Selenium ის ინსტრუმენტები, რომლებიც შესრულებაზე უშვებს ჰაბიდან მითითებულ ტესტებს;
- Grid-ს შეიძლება გააჩნდეს ერთი ან მეტი კვანძი;
- კვანძები შეიძლება აღიწეროს მრავალ მანქანაზე, განსხვავებულ პლატფორმასა და ბრაუზერზე;
- მანქანებზე დაყენებულ კვანძებს არ მოეთხოვება იყოს გაშვებული ჰაბის მსგავს პლატფორმაზე.

განვიხილოთ Grid 2.0-ის ინსტალაცია. დავუშვათ, რომ გვაქვს ორი მანქანა. პირველი მანქანა – სისტემა სადაც

გავუშვებთ ჰაბს, მეორე მანქანა – განვიხილავთ Grid-ის კვანძად [42]. სიმარტივისთვის ჰაბის მანქანას ვუწოდოთ „მანქანა A", ხოლო კვანძის მანქანას ვუწოდოთ „მანქანა B". ვთქვათ A მანქანის IP მისამართია 192.168.8.13, ხოლო B მანქანის 192.168.8.13.

ბიჯი 1: გადმოწერეთ Selenium RC სრვერი Selenium-ის ოფიციალური საიტიდან;

ბიჯი 2: შეინახეთ selenium-server.jar ფაილი სადმე ხისტ დისკოზე. ამით Selenium Grid_2.0-ის ინსტალაცია დასრულდა. მივყვეთ მომდევო პუნქტებს, გავუშვათ ჰაბი და მასზე დავარეგისტრიროთ კვანძი;

ბიჯი 3:

- მანქანა A-ზე გახსენით cmd ფანჯარა და კონსოლიდან მიმართეთ RC–ის დირექტორიას;
- Windows ბრძანებათა ველში გაუშვით შემდეგი ბრძანება:

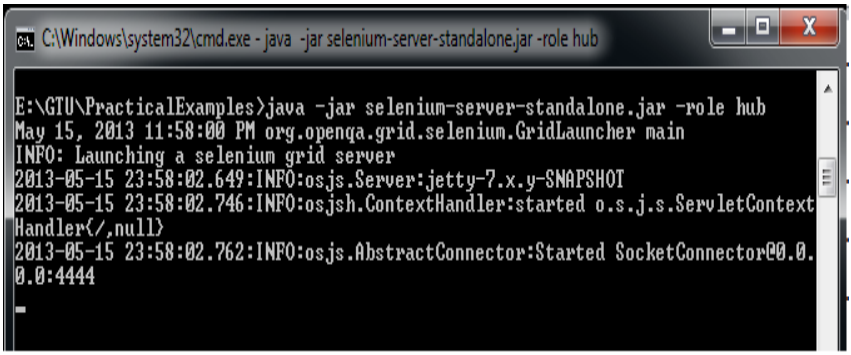
java -jar selenium-server.jar -role hub

- ჰაბი უნდა გაეშვას შესრულებაზე. ჩვენი cmd ფანჯარა დაახლოებით უნდა გამოიყურებოდეს შემდეგნაირად (ნახ.5.10).

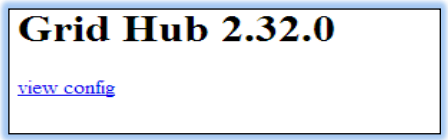
ბიჯი 4: ჰაბის გაშვება შევამოწმოთ ბრაუზერის გამოყენებით. Grid 2 „გაჩუმების" პრინციპით Web ინტერფეისისთვის იყენებს A მანქანის 4444 პორტს. გავხსნათ ბრაუზერი და გადავიდეთ:

http://localhost:4444/grid/console

ლინკზე (ნახ.5.11).



ნახ.5.10. cmd: გაშვებული ჰაბი



ნახ.5.11. ჰაბის Web ინტერფეისი

B მანქანიდანაც არის შესაძლებელი ჰაბის Web ინტერფეისზე კავშირის შემოწმება შემდეგი URL–ით:

http://iporhostnameofmachine:4444/grid/console

სადაც "iporhostnameofmachine" არის A მანქანის (ჰაბის მანქანა) IP მისამართი, ან მისი ჰოსტის სახელი.

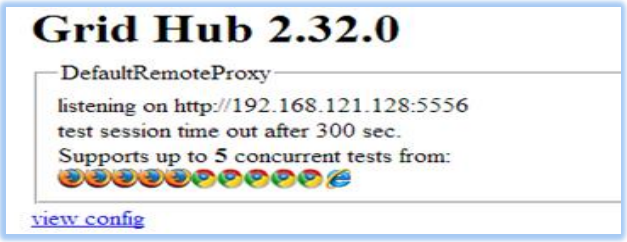
ბიჯი 5:

- მოცემულ ბიჯზე განვსაზღვროთ კვანძი. გადავიდეთ B მანქანაზე და გავხსნათ cmd ფანჯარა;
- კონსოლიდან მივმართოთ RC სერვერის დირექტორიას და ჩავწეროთ ქვემოთ აღწერილი ბრძანება:

java -jar selenium-server.jar -role webdriver -hub http://192.168.8.13:4444/grid/register -port 5556

- მოცემული ბრძანების შესრულების შედეგად Grid-ის ჰაბზე მოხდება კვანძის დარეგისტრირება (ნახ.5.12).

ბიჯი 6: Grid–ის Web ინტერფეისის განახლებით მიიღებთ დაახლოებით შემდეგ სურათს:

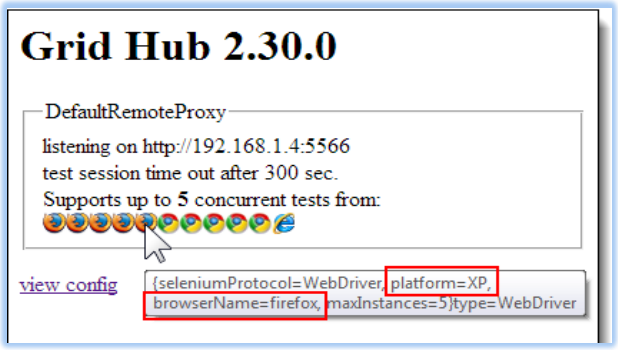


ნახ.5.12. ჰაბზე დარეგისტრირებული კვანძი

ამრიგად, მოცემული ინსტრუქციის საშუალებით დავაყენეთ Selenium Grid 2.0, გავუშვით ჰაბი და მასზე დავარეგისტრირეთ კვანძი.

Grid 2.0-ზე ტესტების გასაშვებად საჭიროა DesiredCapabilities და RemoteWebDriver ობიექტების გამოყენება. DesiredCapabilities ობიექტის საშუალებით განისაზღვრება ბრაუზერისა და ოპერაციული სისტემის ტიპი, ხოლო RemoteWebDriver ობიექტი გამოიყენება იმ კვანძის მისათითებლად, სადაც უნდა გაეშვას ჩვენი ავტომატური ტესტი.

DesiredCapabilities ობიექტის კომპონენტების განსასაზღვრის მიზნით იყენებენ Grid 2.0–ის Web ინტერფეისს (ნახ.5.13).



ნახ.5.13. რეგისტრირებული კვანძის ატრიბუტები

მოცემულ შემთხვევაში პლატფორმა არის XP, ხოლო ბრაუზრის დასახელება firefox. აღნიშნული DesiredCapabilities ობიექტის განმსაზღვრელ კოდის ფრაგმენტს ექნება შემდეგი სახე:

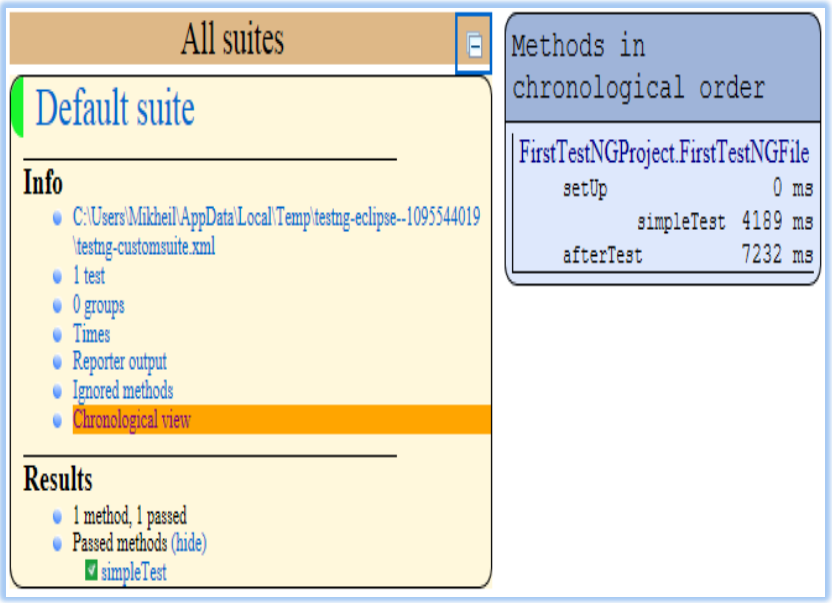
```
DesiredCapabilities capability = DesiredCapabilities.firefox();
capability.setBrowserName("firefox");
capability.setPlatform(Platform.XP);
```

ქვემოდ მოყვანილია მარტივი ტესტის მაგალითი Grid 2.0–ზე. იგი წარმოადგენს TestNG ფრეიმვორკის პროგრამულ კოდს, რომლის გაშვება შესაძლებელია A მანქანაზე, Eclipse გარემოში. მისი გაშვების შემდეგ ტესტის ავტომატიზაცია მოხდება B მანქანის Firefox ბრაუზერზე:

```
import org.openqa.selenium.*;
import org.openqa.selenium.remote.DesiredCapabilities;
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.remote.RemoteWebDriver;
```

```
import org.testng.Assert;
import org.testng.annotations.*;
public class FirstTestNGFile {
        WebDriver driver;
        String baseUrl, nodeURL;
        @BeforeTest
        public void setUp() throws MalformedURLException  {
                baseUrl = "http://newtours.demoaut.com/";
                nodeURL = "http://192.168.121.128:5556/wd/hub/";
        DesiredCapabilities capability =
                                DesiredCapabilities.firefox();
                capability.setBrowserName("firefox");
                capability.setPlatform(Platform.XP);
  driver = new RemoteWebDriver(new URL(nodeURL), capability);
        }
        @AfterTest
        public void afterTest() {
                driver.quit();
        }
        @Test
        public void simpleTest() {
                driver.get(baseUrl);
  Assert.assertEquals("Welcome: Mercury Tours", driver.getTitle());
        }
   }
```

ტესტის მუშაობის შედეგად უნდა მივიღოთ შემდეგი სახის რეპორტი (ნახ.5.14) [167].



ნახ.5.14. Eclipse: TestNG რეპორტ ფაილი

Selenium 3.0-ის მთავარი ცვლილება იყო ის, რომ, მასში ამოღებულ იქნა Selenium Core და შეიცვალა იგი WebDriver-ის მხარდაჭერით. ამან გავლენა მოახდინა Selenium RC API-ის ყველა მომხმარებელზე.

Selenium 4.0-ის ძირითადი ცვლილებაა ახალი CLI runner (გამშვები) პროგრამა, რომლის საფუძველია Node.JS, ნაცვლად ძველი CLI-ისა, romelic miuSaobda HTML-ზე [172].

- მას აქვს ტესტური მაგალითების პარალელურად შესრულების შესაძლებლობა და იძლევა ანალიზის შედეგების ინფორმაციას წარმატებით გავლილი და

ჩავარდნილი ტესტების, აგრეთვე დროითი მახასიათებლების შესახებ და ა.შ.

- ახალი IDE runner მთლიანად ეყრდნობა WebDriver-ს.

Grid 4: Selenium-ის ახალი სერვერი მოიცავს ყველაფერს, რაც საჭიროა Grid-ის გასაშვებად. ეს არის ერთი jar-ფაილი, რომელიც შეიცავს ყველა დამოკიდებულებას, ამგვარად, მისი ამუშავება java -jar selenium-server-4.0 -ს საშუალებით [171].

5.3. Selenium სერვერი და მატესტირებელი ფრეიმვორკები: JUnit, TestNG, Screenshots

განვიხილოთ ტესტ-ქეისების შექმნა Java ტექნოლოგიით. ავაგოთ Java ტესტები JUnit და TestNG ფრეიმვორკების საშუალებით. ტესტირების პროცესში განვახორციელოთ ბრაუზერში გახსნილი გვერდების სქრინების აგება და ა.შ.

Java ტესტ-სკრიპტების დასაწერად საჭიროა დაპროგრამების რომელიმე ინსტრუმენტი, მაგალითად, Java IDE.

თავდაპირველად, სიმარტივისათვის ტესტების შესაქმნელად გამოვიყენოთ Intellij IDEA, რადგან იგი შეიცავს ყველა ინსტრუმენტს, რაც საჭიროა წარმატებული ტესტების შესაქმნელად. გამოვიყენებთ აგეთვე Eclipse ინსტრუმენტსაც. ამრიგად, განსახილველი მაგალითების რეალიზაციისთვის დაგვჭირდება ორივე Intellij IDEA და Eclipse ინსტრუმენტული საშუალება [155].

Java ტესტ-ქეისების დაწერისას დაგვჭირდება JUnit „ფრეიმვორკი", რომლის გადმოსაწერად გამოვიყენებთ შემდეგ მისამართს - URL: http://junit.org/.

- ***JUnit*** – არის მატესტირებელი Java ფრეიმვორკი, რომელიც უზრუნველყოფს განმეორებადი ტესტების შექმნას. იგი ძირითადად გამოიყენება unit (მოდულური) ტესტების დასაწერად. არის უფასო, open source ტექნოლოგია.

Selenium RC ტექნოლოგიის ერთ-ერთი მთავარი უპირატესობაა პროგრამული ლოგიკის რეალიზება ტესტ–სკრიპტებში. პროგრამული ლოგიკა მსგავსია დაპროგრამების ყველა ენაში. აპლიკაციის შესრულება ძირითადად იმართება პირობითი ოპერატორებისა და ციკლების საშუალებით. პირობითი ოპერატორის მაგალითია **if ( )** ოპერატორი, ხოლო ციკლისა **for ( ).** Java სინტაქსით Selenium ტესტ-სკრიპტების რეალიზებით შესაძლებელია ნებისმიერი რთული შემთხვევის ავტომატიზაცია.

განვიხილოთ როგორ ხდება Selenium ტექნოლოგიაში პროგრამული ენის კომბინაციით ტესტირების პრობლემების გადაჭრა. თუ განხილული მარტივი HTML ტესტებიდან გადავალთ რთულ ტესტებზე, სადაც ხდება Web გვერდებზე არსებული დინამიკური ფუნქციების გამოკვლევა და დინამიკურად მიღებული მონაცემების დადასტურება, მაშინ ამ დინამიკური პროცესების ავტომატიზაციისათვის საჭირო იქნება პროგრამული ლოგიკა, რომ დავადასტუროთ მოსალოდნელი შედეგი.

ზოგადად, Selenium IDE-ს არ აქვს ციკლების და პირობითი ოპერატორების მხარდაჭერა. შესაძლებელია რაღაც პირობის განსაზღვრა JavaScript კოდის ჩადგმით Selenese ბრძანებებში.

მიუხედავად ამისა იტერაციები (ციკლები) შეუძლებელია და პირობითი ოპერატორების რეალიზებაც დაპროგრამების ენაზე გაცილებით მარტივად და გასაგებად აღიწერება.

ამასთანავე, შეცდომების აღმოსაჩენად შეიძლება დაგვჭირდეს გამონაკლისი შემთხვევების (*exception handling*) მართვა.

ჩამოთვლილი მიზეზების გამო განვიხილოთ პრაქტიკული მაგალითი, სადაც გამოჩნდება პროგრამული ტექნიკის ფართო გამოყენების საჭიროება.

ავტომატურ ტესტებში პროგრამული ენის გამოყენება ამდიდრებს ვერიფიკაციის შესაძლებლობებს.

იტერაცია ტესტ-სკრიპტში არის ერთ-ერთი ყველაზე გამოყენებადი ინსტრუმენტი. მაგალითად, ერთიდაიმავე ტესტში შეიძლება დაგვჭირდეს ძებნის ტესტის გაშვება მრავალჯერ.

HTML ფორმატის ტესტ-სკრიპტიდან ცხადია, რომ მსგავსი მოქმედებების შესასრულებლად საჭიროა ერთიდაიმავე კოდის გამეორება ყოველ იტერაციაზე.

როგორც ვიცით მსგავსი კოდის მრავალჯერადი კოპირება არ არის კარგი პროგრამული პრაქტიკა.

იგი მოითხოვს უფრო მეტ მუშაობას რეალიზაციისთვის. პროგრამული ენის გამოყენებით შესაძლებელია არსებული პრობლემის გადაჭრა, ძებნის „გაციკლება", უფრო დახვეწილი და მოქნილი გადაწყვეტილების პოვნა.

ქვემოთ მოცემულია ტესტი C# დაპროგრამების ენაზე (ნახ.5.15).

```
// Collection of String values.
String[] arr = {"ide", "rc", "grid"};
// Execute loop for each String in array 'arr'.
foreach (String s in arr) {
   sel.open("/");
   sel.type("q", "selenium " +s);
   sel.click("btnG");
   sel.waitForPageToLoad("30000");
   assertTrue("Expected text: " +s+ " is missing on page."
   , sel.isTextPresent("Results * for selenium " + s));
 }
```

ნახ.5.15. C# ტესტ-სკრიპტი

მოცემული ტესტი Java დაპროგრამების ენაზე (ნახ.5.16):

```
// Array of String values.
String[] str = {"ide", "rc", "grid"};
// Execute loop for each String in array ' str'.
for (String s : str) {
   sel.open("/");
   sel.type("q", "selenium " +s);
   sel.click("btnG");
   sel.waitForPageToLoad("30000");
   assertTrue("Expected text: " +s+ " is missing on page."
   , sel.isTextPresent("Results * for selenium " + s));
 }
```

ნახ.5.16. Java ტესტ-სკრიპტი

მოცემული ერთიდაიგივე ტესტ-ქეისის C# და Java პროგრამული ფრაგმენტებიდან ჩანს, რომ დაპროგრამების ენით დაწერილი ტესტ-სკრიპტი პროგრამული თვალაზრისით გაცილებით ელეგნტურია, ვიდრე HTML ფორმატით შექმნილი სკრიპტი.

HTML ტესტში ერთიდაიმავე ფრაგმენტის გადაწერა ხდება 3-ჯერ, ხოლო დაპროგრამების ენით, მოცემული ბიჯი აღვწერეთ ერთხელ და დავატრიალეთ ციკლით.

ტესტ-სკრიპტში პირობითი ოპერატორის გამოყენების საჭიროების საილუსტრაციოდ განვიხილოთ კიდევ ერთი მაგალითი.

ტესტირების პროცესის დროს, ძირითადად მოულოდნელი პრობლემა წარმოიშობა მაშინ, როდესაც ტესტ-ქეისში აღწერილი ელემენტი არ მოიძებნება მიმდინარე Web გვერდზე. მაგალითად, როდესაც გაეშვება შემდეგი სტრიქონი [154]:

```
selenium.type("q", "selenium " +s);
```

თუ ელემენტი 'q' არ არის web გვერდზე, მაშინ მივიღებთ შემდეგი ტიპის შეცდომას:

```
com.thoughtworks.selenium.SeleniumException: ERROR:
Element q not found
```

აღნიშნული შემთხვევა გამოიწვევს ტესტის ნაადრევად დასრულებას. მსგავსი რამ სცენარის შესრულებისას არ არის სასურველი, რადგან პრაქტიკაში ტესტ-სცენარს აქვს მრავალი სხვა ტესტ-ქეისების ქვემიმდევრობა შესასრულებელი.

მოცემულ პრობლემას გადავწყვეტთ პირობითი ოპერატორის გამოყენებით, რომლის საშუალებითაც შევამოწმებთ არსებობს თუ არა ელემენტი გვერდზე და მივიღებთ ალტერნატიულ გადაწყვეტილებას ელემენტის არარსებობის შემთხვევაში.

აღნიშნულის საილუსტრაციოდ გამოვიყენოთ Java პროგრამული ფრაგმენტი:

```
// If element is available on page then perform type operation.
        if(selenium.isElementPresent("q")) {
           selenium.type("q", "Selenium rc");
        } else {
  System.out.printf("Element:"+q+"is not available on page.")
        }
```

აღწერილი ფრაგმენტის უპირატესობაზე მეტყველებს ის, რომ მისი შესრულების შემდეგ შესაძლებელია გაგრძელდეს ტესტი, მიუხედავად იმისა, არსებობს თუ არა რაღაც UI (User Interface) ელემენტი მიმდინარე გვერდზე.

მაშასადამე, ტესტ-ქეისების დაპროგრამების Java ენაზე რეალიზებით Selenium მატესტირებელი ტექნოლოგია გახდა უფრო მდიდარი შესაძლებლობების მქონე, ადვილად გასაგები და დინამიკური პროცესების მართვადი ინსტრუმენტი.

ჩვენ მიერ გაშვებული JUnit ტესტები არ აგენერირებს სათანადო საშედეგო ფაილს. საშედეგო ფაილი წარმოადგენს შესრულებული სცენარის რეპორტს, მოცემულმა აპლიკაციამ გაიარა თუ არა ჩვენ მიერ განსაზღვრული სცენარი.

ავტომატური ტესტირებისას რეპორტ ფაილის გენერირება მნიშვნელოვანია, რადგან მისი საშუალებით შესაძლებელია მსჯელობა, მუშაობს თუ არა მოცემული აპლიკაცია მოცემული სცენარის მიმართ.

წარმოდგენილ მასალაში განვიხილეთ HTML ფორმატის ტესტ-სცენარის საშედეგო ფაილი, სადაც „ჩავარდნილი“ ტესტები მოიცემოდა წითელ ფერში, ხოლო წარმატებით

შესრულებული ტესტები მწვანეში. აღნიშნულ საკითხს გაცილებით კარგად ართმევს თავს Java-ს მატესტირებელი ფრეიმვორკი სახელად **TestNG**.

- ***TestNG –*** არის JUnit–ზე ბაზირებული მატესტირებელი ფრეიმვორკი, რომელიც ახდენს JUnit ფრეიმვორკის ნაკლოვანებების აღმოფხვრას. მის დასახელებაში არსებული NG ნიშნავს "Next Generation" (შემდგომი გენერაცია).

Selenium მომხმარებლების უმრავლესობა უპირატესობას ანიჭებს TestNG-ს, ვიდრე JUnit-ს.

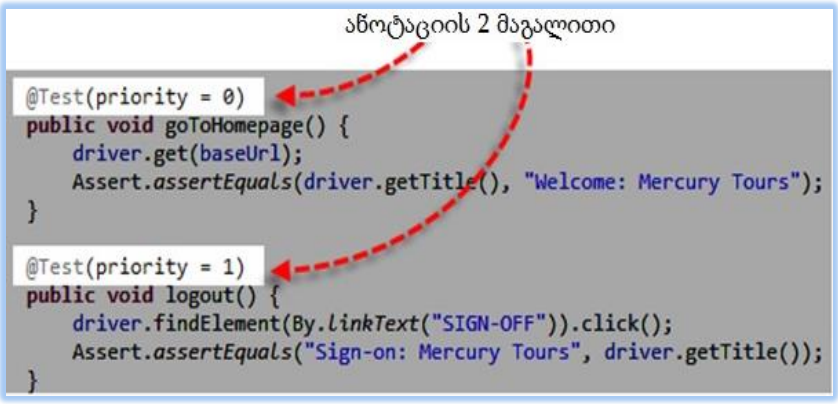
TestNG-ის აქვს მრავალი პოპულარული მახასიათებელი. ჩვენ აქ მიმოვიხილავთ ყველაზე მნიშვნელოვანს მათ შორის, რომლებიც კავშირშია Selenium ტექნოლოგიასთან [42,154].

TestNG-ის JUnit-თან შედარებით გააჩნია 3 ძირითადი უპირატესობა:

1) ანოტაციები არის უფრო ადვილად გასაგები;
2) ტესტ-ქეისები შესაძლებელია დაჯგუფდეს უფრო ადვილად;
3) შესაძლებელია პარალელური ტესტირება.

ანოტაცია წარმოადგენს სტრიქონს პროგრამულ კოდში, რომელიც საზღვრავს მის ქვემოთ აღწერილი მეთოდის გაშვების პრინციპებს.

ანოტაციას წინ უძღვის @ სიმბოლო. თვალსაჩინოებისთვის ანოტაციის მაგალითი წარმოდგენილია 5.17 ნახაზზე.



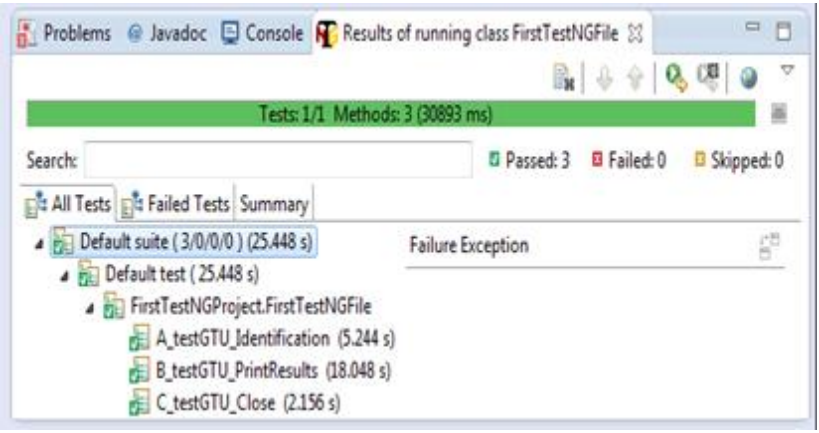
ნახ.5.17. TestNG ანოტაცია

მოცემულ მაგალითში goToHomepage() მეთოდი გაეშვება პირველი, ვიდრე logout(), რადგან მას აქვს პრიორიტეტის უფრო დაბალი მნიშვნელობა. მოცემული სურათიდან ჩანს, რომ ანოტაციები TestNG–ში აღიწერება უფრო მარტივად და გასაგებია ვიდრე JUnit–ში [42].

TestNG ფრეიმვორკს აქვს პარალელური ტესტების გაშვების შესაძლებლობა, მაშასადამე Selenium Grid პარალელური ტესტერებისთვის იგი უფრო პრივილეგირებული ტექნოლოგიაა, ვიდრე JUnit-ი.

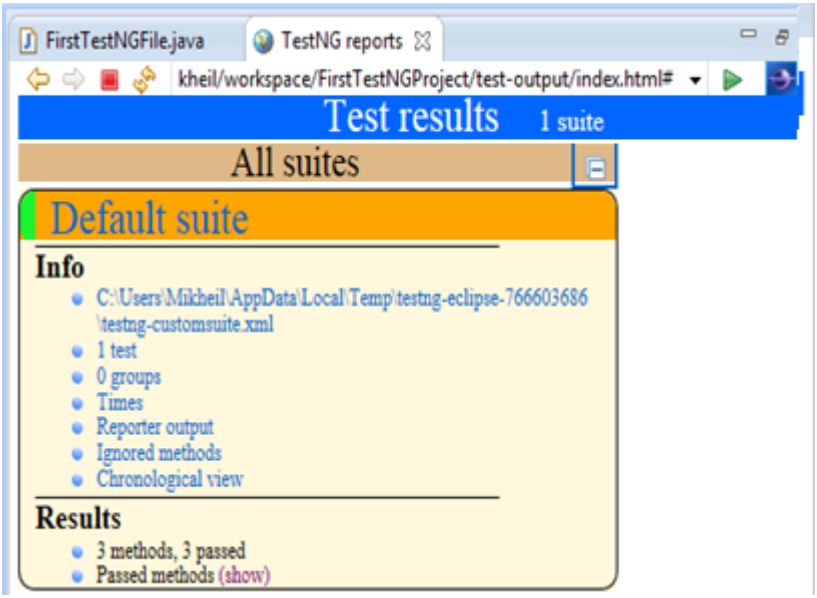
TestNG-ის უპირატესობად უნდა აღინიშნოს ისიც, რომ მას შეუძლია აღუწერავი გამონაკლისი შემთხვევების მართვა, რაც თავიდან გვაცილებს ტესტის ნაადრევად დასრულების შესაძლებლობას.

TestNG, გარდა ხისებური სტილით მოცემული და კონსოლში ჩაწერილი ტექსტური რეპორტისა (ნახ.5.18), აგენერირებს აგრეთვე HTML ფორმატის საშედეგო ფაილს (ნახ.5.19) [167].



ნახ.5.18. Eclipse: TestNG გრაფიკული რეპორტი

პარაგრაფის დასაწყისში ჩვენ შევეხეთ **@Test** ანოტაციის საკითხს. ახლა განვიხილოთ იგი უფრო დაწვრილებით.



ნახ.5.19. Eclipse: TestNG HTML რეპორტი

შესაძლებელია მრავალი @Test ანოტაციის გამოყენება ერთ TestNG ფაილში. @Test-ით ანოტირებული მეთოდები „გაჩუმების“ პრინციპით სრულდება ანბანური თანმიმდევრობით.

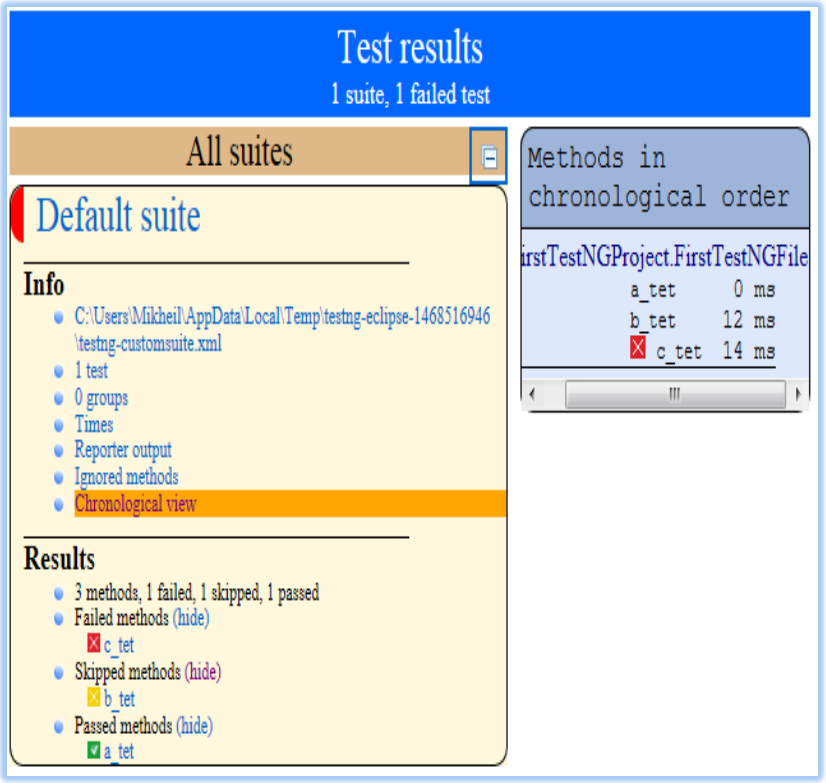
მიუხედავად იმისა, რომ მოცემულ ნახაზზე მეთოდები c_test(), a_test() და b_test() კოდში არ არის დალაგებული ანბანურად, ისინი გაეშვებიან ანბანური რიგის მიხედვით (ნახ.4.20) [42].

```
public class FirstTestNGFile {

    @Test
    public void c_test() {
        Assert.fail();
    }

    @Test
    public void a_test() {
        Assert.assertTrue(true);
    }

    @Test
    public void b_test() {
        throw new SkipException("Skipping b_test...");
    }
}
```

ნახ.5.20. TestNG ანოტაციები

აღნიშნულის სადემონსტრაციოდ შესრულებაზე გავუშვათ მოცემული კოდი და შევხედოთ საშედეგო ფაილს.

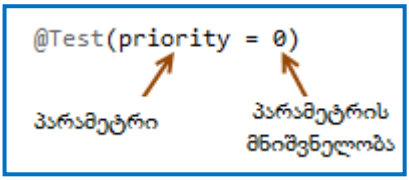
5.21 ნახაზიდან ჩანს, რომ ტესტები გაშვებულია ანბანის რიგის მიხედვით. იმისთვის რომ ანოტირებული მეთოდები გაეშვას სხვა მიმდევრობით და არა ანბანური რიგით, იყენებენ "priority" პარამეტრს.



ნახ.5.21. Eclipse: TestNG HTML რეპორტი

პარამეტრები საკვანძო სიტყვებია, რომლებიც ცვლის ანოტაციების ფუნქციებს [42].

- პარამეტრი მოითხოვს რაღაც მნიშვნელობის მინიჭებას, რაც ხორციელდება ჩვეულებრივი "=" ნიშნით, რომლის შემდეგაც იწერება მნიშვნელობა;
- პარამეტრი იწერება მრგვალ ფრჩხილებში, ანოტაციის დასახელების შემდეგ (ნახ.5.22)



ნახ.5.22

@Test ანოტაციებს TestNG ასრულებს ყველაზე დაბალი პრიორიტეტიდან მაღლისკენ. ამასთანავე არ არის საჭირო პრიორიტეტების მნიშვნელობების თანამიმდევრობის დაცვა, მაგალითად განვიხილოთ შემდეგი კოდის ფრაგმენტი (ნახ.5.23) [173].

```
public class FirstTestNGFile {

    @Test(priority = 3)
    public void c_test() {
        Assert.fail();
    }

    @Test(priority = 0)
    public void a_test() {
        Assert.assertTrue(true);
    }

    @Test(priority = 7)
    public void b_test() {
        throw new SkipException("Skipping b_test...");
    }
}
```

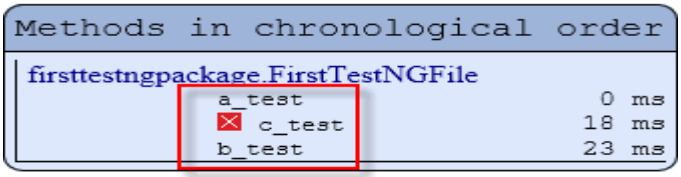
სიდიდით მეორე პრიორიტეტ–მნიშვნელობა აწუ იგი შესრულდება მეორე

ყველაზე დაბალი პრიორიტეტ–მნიშვნელობა აწუ იგი შესრულდება პირველი

ყველაზე მაღალი პრიორიტეტ–მნიშვნელობა აწუ იგი შესრულდება ბოლოს

ნახ.5.23. TestNG: პრიორიტეტებ-განსაზღვრული ტესტ-ქეისები

თუ მოცემულ კოდს გავუშვებთ შესრულებაზე, HTML რეპორტში, დავინახავთ მეთოდების შესრულების თანამიმდევრობას პრიორიტეტების მიხედვით (ნახ.5.24).



ნახ.5.24. ტესტ-ქეისების პრიორიტეტებით განსაზღვრული სია

განვიხილოთ TestNG მატესტირებელი ფრეიმვორკის ძირითადი ანოტაციები, რომლებიც შეიძლება გამოვიყენოთ Selenium ტესტების განსზღვრისას [173].

@BeforeTest – აღწერილი ანოტაციის ქვემოთ განსაზღვრული მეთოდი შესრულდება პირველი ტესტ-ქეისების გაშვებისას;

@AfterTest – აღწერილი ანოტაციის ქვემოთ განსაზღვრული მეთოდი შესრულდება ყველა ტესტ-ქეისის გაშვების შემდეგ.

მაგალითისათვის განვიხილოთ შემდეგი პროგრამული კოდის ფრაგმენტი (ნახ.5.25).

მოცემულ კოდში აღწერილი მეთოდები შესრულდება შემდეგი თანამიმდევრობით:

პირველი – launchBrowser()

მეორე – verifyHomepageTitle()

მესამე – terminateBrowser()

შევნიშნოთ, რომ კოდში აღწერილი მეთოდის ბლოკის გადაადგილებით, შესრულების მიმდევრობა არ იცვლება.

```
public class FirstTestNGFile {
    public String baseUrl = "http://newtours.demoaut.com/";
    public WebDriver driver = new FirefoxDriver();
    @BeforeTest
    public void launchBrowser() {
        driver.get(baseUrl);
    }
    @Test
    public void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle);
    }
    @AfterTest
    public void terminateBrowser() {
        driver.quit();
    } }
```

ნახ.5.25. TestNG: ანოტირებული ტესტ-მეთოდები

TestNG ანოტაციების შეჯამება [173]:

@BeforeMethod – მოცემული ანოტაციის ქვემოთ აღწერილი მეთოდი გაეშვება ყველაზე *პირველი*, ნებისმიერი მეთოდის გაშვებამდე;

@AfterMethod – მოცემული ანოტაციის ქვემოთ აღწერილი მეთოდი გაეშვება ყველაზე *ბოლოს*, ყველა მეთოდის გაშვების შემდეგ;

შევნიშნოთ, რომ ერთიდაიმავე ტესტ-ფაილში აღწერილი @BeforeMethod და @BeforeTest ანოტაციებიდან პირველი გაეშვება @BeforeTest, რადგან იგი სრულდება ტესტ-ქეისის გაშვებამდე პირველი.

@BeforeSuite: მისი საშუალებით ანოტირებული მეთოდი გაეშვება *პირველი მოცემულ ტესტ-სუიტში;*

@AfterSuite: მისი საშუალებით ანოტირებული მეთოდი გაეშვება ყველაზე *ბოლოს მოცემულ ტესტ-სუიტში;*

@BeforeClass: მისი საშუალებით ანოტირებული მეთოდი გაეშვება *პირველი მიმდინარე კლასში;*

@AfterClass: მისი საშუალებით ანოტირებული მეთოდი გაეშვება ყველაზე ბოლოს მიმდინარე კლასში;

@Test: მისი საშუალებით ანოტირებული მეთოდი არის ტესტ-ქეისის შემადგენელი ნაწილი.

დასკვნის სახით შეგვიძლია ჩამოვაყალიბოთ TestNG ჯავა მატესტირებელი ფრეიმვორკის ძირითადი მახასიათებლები [173]:

- TestNG არის მატესტირებელი ფრეიმვორკი, რომელსაც შეუძლია Selenium ტესტების შექმნა და ადვილად გასაგები რეპორტის გენერირება;
- TestNG-ის ძირითადი უპირატესობები JUnit–თან მიმართებაში: ადვილად გასაგები ანოტაციები, ტესტების დაჯგუფების უფრო ადვილი შესაძლებლობა და პარალელური ტესტირების მხარდაჭერა;
- Eclipse-ში კონსოლის ფანჯარაზე მოიცემა ტესტირების ტექსტური რეპორტი, ხოლო TestNG ფანჯარაზე გრაფიკული, ხისებური სტრუქტურით წარმოდგება ტესტირების დეტალური შედეგი: ტესტის ხანგრძლივობის დრო, მეთოდების შესრულების ქრონოლოგიური მიმდევრობა;
- TestNG აგენერირებს HTML ფორმატის რეპორტს;
- TestNG ანოტაციებს შეუძლია პარამეტრების გამოყენება ისევე, როგორც ჩვეულებრივ Java მეთოდებს.

ავტომატური ტესტებია ის სცენარები, რომლებიც გაიშვება სერვერებზე და სრულდება დამოუკიდებლად. პროგრამისტი ან ტესტერი მათ შესრულების პროცესში არ ერევა. თუ სცენარი შესრულდა ვინმეს ხელოვნური მანიპულირებით, მაშინ ამბობენ რომ მოცემული ტესტი არის ხელოვნური (manual) ტესტი [42].

ავტომატური ტესტირების დროს შეიძლება დადგეს ისეთი მექანიზმის გამოყენების საჭიროება, რომლითაც შევძლებთ გვერდის სქრინის გაკეთების, რათა სურათზე დავინახოთ ის შეცდომა, რომელმაც გამოიწვია ავტომატური ტესტის „ჩავარდნა“.

პირველი ბრძანება, რომელსაც განვიხილავთ Selenium ტექნოლოგიაზე, არის **captureScreenshot** ოპერატორი. იგი აკეთებს კომპიუტერის მიმდინარე სამუშაო მაგიდის სქრინს და არა ბრაუზერში გახსნილი გვერდის სქრინს. მოცემული ბრძანება გამოიყენება იმ შემთხვევაში, როდესაც გვაინტერესებს იმ მომენტში რა ხდება სამუშაო მაგიდაზე, როცა ტესტი „ვარდება“. შესაძლოა ამ დროს ბრაუზერი საერთოდ არ გამოჩნდეს სურათზე.

აღნიშნულ ბრძანებას აქვს **captureScreenshot(file)** ფორმატი, სადაც file არის გზა და ფაილის დასახელება, რათა განსაზღვრულ გზაზე მოცემული დასახელებით მოხდეს სქრინის შენახვა. თუ მივუთითებთ მხოლოდ ფაილის დასახელებას, იგი სქრინს შეინახავს Selenium RC სერვერის დირექტორიაში, ხოლო თუ მივუთითებთ ფაილის შესანახ გზას, იგი შექმნის აღნიშნულ სურათს მითითებულ ფოლდერში [42].

დროის თვალსაზრისით, ხანგრძლივი სცენარების ავტომატური ტესტირებისას სისტემაში, თავდაპირველად იქმნება საწყისი მონაცემები და შემდგომ მათზე განისაზღვრება (აიგება) მომდევნო მოქმედებები.

თუ რომელიმე ტესტ-ქეისი „ჩავარდება“, იგი დაფიქსირდება რეპორტ ფაილში. სცენარის დასრულების შემდეგ ხდება რეპორტ ფაილის ანალიზი. ჩავარდნილი სცენარის შემთხვევაში საჭიროა ტესტ-სკრიპტებში გაწერილი ლოგიკის ხელით შესრულება, რომ მივიღოთ ავტომატური ტესტირებისას წარმოქმნილი პრობლემა ბრაუზერში.

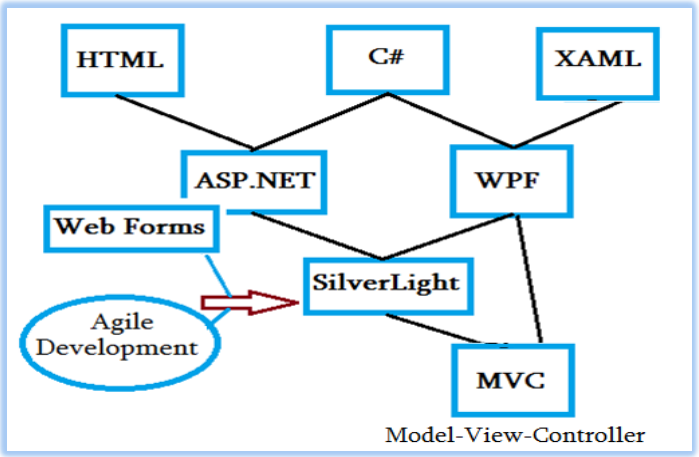
ცხადია, ეს მოითხოვს დამატებით რესურსებს, დროის ხარჯვას, სცენარის ანალიზს და ა.შ. მოცემულ შემთხვევაში ძალზე ხელსაყრელია შეცდომის ადგილების სქრინების ავტომატურ რეჟიმში გაკეთება.

პროგრამების ავტომატური ტესტირების დროს სქრინების გაკეთება არის გამოყენებადი, თუმცა ტესტირების პროცესის ვიდეოს ჩაწერა არის გაცილებით საინტერესო საკითხი.

VI თავი

პროგრამული აპლიკაციების დაპროექტება, დაპროგრამება და ტესტირება SOA-ის ბაზაზე

წინამდებარე თავში წარმოდგენილია Microsoft კომპანიის თანამედროვე პროგრამული ტექნოლოგიები და მათი გამოყენება ბიზნეს-აპლიკაციების დამუშავების მიზნით Visual Studio.NET სამუშაო გარემოში. 6.1 ნახაზზე ნაჩვენებია ჰიბრიდული პროგრამული ტექნოლოგიების განვითარების ზოგადი სქემა, Web- და Desctop-აპლიკაციების ასაგებად [75].



ნახ.6.1. პროგრამული აპლიკაციების ტექნოლოგიების ევოლუცია

ძირითადად, გავაანალიზებთ ASP.NET MVC ტექნოლოგიას, ვინაიდან მას მნიშვნელოვანი ადგილი უკავია თანამედროვე პროგრამული სისტემების აგების სფეროში.

6.1. ASP.NET MVC ტექნოლოგია და ტესტირება

➢ ***ASP.NET პლატფორმა:*** როგორც ცნობილია, ASP.NET Web Forms ტექნოლოგიის კრიტიკისა და არა-Microsoft ტექნოლოგიების (განსაკუთრებით – Ruby on Rails ტექნოლოგია, თავისი Agile Development მიდგომებით MVC არქიტექტურით, და HTTP პროტოკოლზე მუშაობის პრინციპით) მზარდი პოპულარობის საპასუხოდ, 2007 წლის ოქტომბერში მაიკროსოფთმა გააკეთა განცხადება ASP.NET პლატფორმაზე დაშენებული ახალი პროგრამული პლატფორმის გამოშვების შესახებ [174].

ამ ახალ ტექნოლოგიაში კომბინირებულია Model-View-Controller (MVC) არქიტექტურის ეფექტურობა, Agile Development მიდგომის უახლესი იდეოლოგია და ASP.NET პლატფორმის საუკეთესო ნაწილები [175]. იგი ტრადიციული ASP.NET Web Forms ტექნოლოგიის სრულყოფილი ალტერნატივაა, უპირატესია თითქმის ყველა შემთხვევაში, გარდა ტრივიალური ვებ საიტების პროექტირებისა.

ASP.NET MVC Framework პროგრამულმა პლატფორმამ იმპლემენტაცია გაუკეთა MVC არქიტექტურას და ამით დიდი კონკურენცია გაუწია Ruby on Rails და მის მსგავს პროგრამულ პლატფორმებს, ხოლო MVC არქიტექტურა კი .NET სამყაროში წინა პლანზე წამოწია. სხვა, არა-Microsoft პლატფორმაზე მომუშავე დეველოპერების მიერ წლობით დაგროვილი გამოცდილებისა და საუკეთესო პრაქტიკების გადაღებით გამდიდრებულმა ASP.NET MVC პლატფორმამ ბევრად წინ გაუსწრო კონკურენტუნარიან პლატფორმებს.

- ***ტესტირება:***

MVC არქიტექტურა მაღალი ხარისხის ტესტირების საშუალებას იძლევა. ეს შედეგი არქიტექტურული მიდგომის პრინციპებიდან გამომდინარეობს – ლოგიკური ნაწილები ერთმანეთისგან გამიჯნულია და სხვადასხვა შრეზეა განლაგებული [176].

გარდა ამისა, MVC პროგრამული პლატფორმის შექმნისას, მისმა არქიტექტორებმა MVC Framework-ის კომპონენტზე ორიენტირებული დიზაინის თითოეული ლოგიკური კომპონენტი საფუძვლიანად დააპროექტეს Unit Testing და Mocking Tools ხელსაწყოებთან სრული შესაბამისობის მისაღწევად.

Visual Studio პროგრამაში Unit Test პროექტების შესაქმნელად დამატებულია ე.წ. ოსტატები (Wizards), რაც შემდგომში ინტეგრირებადია ტესტირების სხვა ინსტრუმენტებთანაც, როგორიცაა NUnit, xUnit. MVC-აპლიკაციაში იოლად რეალიზებადია ტესტები სხვადასხვა კომპონენტთან მიმართებაში (როგორიცაა Controller, Action), სიმულაციები, სცენარები.

პროგრამისტს შეუძლია მომხმარებლის მოქმედების სიმულაცია გააკეთოს ტესტირების სკრიპტებით და არ ევალება იმის ცოდნა, თუ როგორ არის HTML ელემენტების და CSS კლასების სტრუქტურა, ან დაგენერირებული ID მნიშვნელობები.

- ***ღია კოდი (Open Source)***

MVC პროგრამული პლატფორმა გახსნილი, ანუ „ღია კოდია“ – მისი პროგრამული კოდის გადმოწერა უფასოა და

ჩამოტვირთულ პროექტში შესაძლებელია საკუთარი ექსპერიმენტების ჩაატარებაც კი. ეს განსაკუთრებით მოსახერხებელია საკუთარი რთული კომპონენტების შემუშავებისას ან როდესაც Debug-პროცესში შეცდომის კვალს მივყავართ სისტემურ კომპონენტთან და საჭიროა მის კოდში შესვლა.

MVC კოდის გადმოწერა შესაძლებელია შემდეგი URL მისამართიდან:

http://aspnetwebstack.codeplex.com

6.2. MVC აპლიკაციის აგება „უნივერსიტეტი“-ს საპრობლემო სფეროსთვის

პრაქტიკული რეალიზაციისა და ექპერიმენტისათვის ვიხილავთ უნივერსიტეტის დომეინის (საპრობლემო სფეროს) შესაბამისი ვებ-აპლიკაციის აგების პროცესს. პირობითად, დავარქვათ მას Technical_University, რომელიც დემონსტრირებას უკეთებს ASP.NET MVC-5 პლატფორმისა და Entity Framework-6 გამოყენებას Visual Studio.NET სამუშაო გარემოში [38].

პროგრამული უზრუნველყოფა შემუშავებულია „Code First” მიდგომით, რომელშიც თავდაპირველად იწერება პროგრამული კოდი, შემდეგ იქმნება მონაცემთა ბაზა) [177].

აპლიკაცია წარმოადგენს „უნივერსიტეტის“ ვებ-საიტის *იმიტაციურ მოდელს.* საიტზე გათვალისწინებულია სტუდენტების, ლექტორების, სასწავლო საგნების, ფაკულტეტებისა და სასწავლო ჯგუფების აღრიცხვა, აგრეთვე სტატისტიკური ინფორმაციის გაანგარიშება სტუდენტების განაწილების შესახებ და სხვ.

აპლიკაციის საწყისი გვერდი ინფორმაციული შინაარსისაა და აღწერს პროგრამული უზრუნველყოფის არქიტექტურასა და ტექნიკურ მახასიათებლებს (ნახ.6.2). რადგან ვებ-საიტი სასწავლო დანიშნულებას ატარებს, იგი მომხმარებელს სთავაზობს პროგრამული ლოგიკის კოდს და URL მისამართს, საიდანაც პროექტის ჩამოტვირთვა არის შესაძლებელი.

ნახ.6.2. აპლიკაციის საწყისი გვერდი

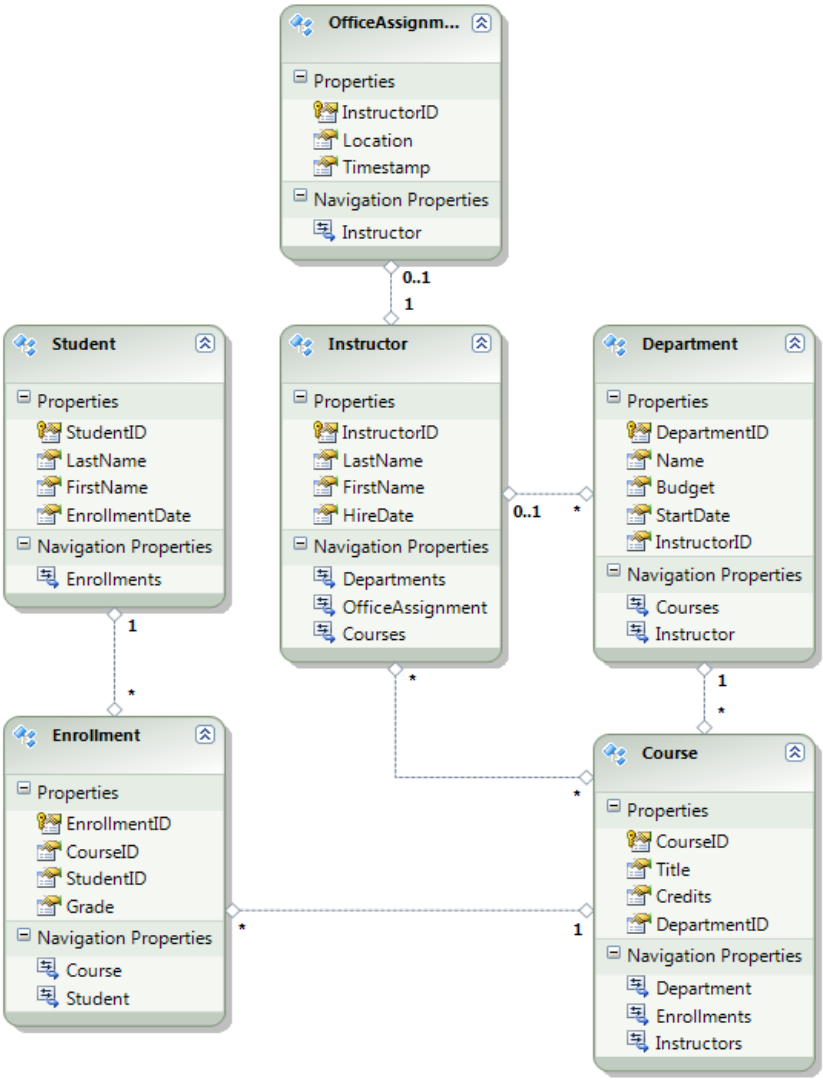
აპლიკაციის დანარჩენ სექციებში თემატურად გადანაწილებულია ინფორმაცია უნივერსიტეტის წევრებისა და სასწავლო პროცესების შესახებ.

მაგალითისათვის, განვიხილოთ ლექტორების ნუსხა (ნახ.6.3). ზედა ცხრილი უნივერსიტეტის ლექტორთა ინფორმაციაა. ჩანაწერის მონიშვნის შემდეგ ცხრილის ქვემოთ გააქტიურდება მეორე ცხრილი, სადაც ნაჩვენებია მონიშნული ლექტორის სასწავლო საგნების სია.

ნახ.6.3. ლექტორთა რეესტრი (Views/Instructor/Index.cshtml)

თითოეულ საგანს აქვს მოსანიშნი, რომლის არჩევის შემთხვევაში გამოჩნდება მესამე ცხრილი, სადაც გამოდის მონიშნულ სასწავლო საგანზე დარეგისტრირებულ სტუდენტთა და მათი აკადემიური შეფასებების ჩამონათვალი.

6.4 ნახაზზე ნაჩვენებია მონაცემთა მოდელის სტრუქტურა, რომელიც შედგება ურთიერთდაკავშირებული ცხრილებისგან [38].



ნახ.6.4. აპლიკაციის მონაცემთა მოდელის სტრუქტურა (Entity Framework 6)

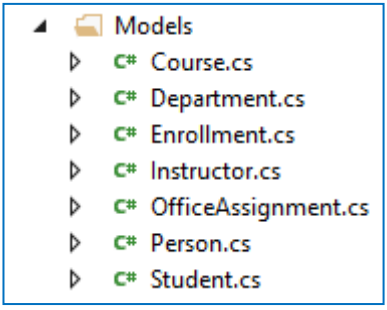
აპლიკაციის დომეინის ცხრილებია:

- Instructor - ლექტორების სიმრავლე;
- Student - სტუდენტების სიმრავლე;
- Course - სასწავლო კურსები;
- Enrollment - სასწავლო ჯგუფები;
- Grade - შეფასებები (enum ტიპის მნიშვნელობები A,B,C,D,F);
- Department - სასწავლო ფაკულტეტები;
- OfficeAssignment - მისამართები და ადგილმდებარეობები.

➢ MVC აპლიკაციის ლოგიკური ორგანიზება

TechnicalUniversity ვებ-აპლიკაციის არქიტექტურა MVC სტანდარტების მიხედვით არის აგებული. პროექტი დაყოფილია ლოგიკურ ნაწილებად და პროგრამული კოდის ფაილები ორანიეზებულია დირექტორიებში: Models, Controllers და Views.

- **Models დირექტორია**



ნახ.6.5. Models დირექტორიაში გაერთიანებულია აპლიკაციის Model-კლასები

Model კლასები ვიზუალურად გაერთიანებულია Model დირექტორიაში, ხოლო პროგრამულად – TechnicalUniversity.Models ბიბლიოთეკაში.

მოდელის კლასებში ხდება *ვალიდაციების* და *წესების განსაზღვრა*, მონაცემთა ფორმატირების გაწერა (FormatString), სავალდებულოობის (Required) მითითება.

მაგალითისთვის, ასეთი შეზღუდვა: „პიროვნების სახელის მითითება სავალდებულოა და არ შეიძლება 50 სიმბოლოზე მეტი იყოს“ (ნახ.6.6).

```
[Required]
[StringLength(50, ErrorMessage = "სახელი არ შეიძლება 50 სიმბოლოზე მეტი იყოს.")]
[Column("FirstName")]
[Display(Name = "სახელი")]
23 references
public string FirstMidName { get; set; }
```

ნახ.6.6. ვალიდაცია

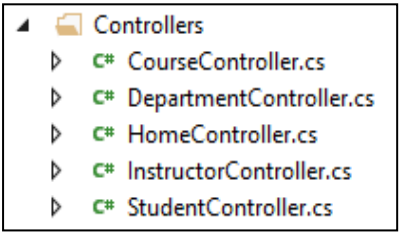
ეს შეზღუდვა იმპლემენტირებულია [StringLength] ატრიბუტით, ხოლო თარიღის გამოსატანი ფორმატი კი - ატრიბუტით [DisplayDateFormat] (ნახ.6.7).

```
26 references
public class Instructor : Person
{
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
                   ApplyFormatInEditMode = true)]
    [Display(Name = "დაქირავების თარიღი")]
    5 references
    public DateTime HireDate { get; set; }
```

ნახ.6.7. თარიღის ფორმატირება

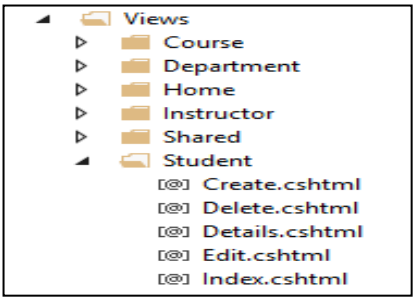
- **Controllers დირექტორია**

TechnicalUniversity პროგრამის ძირითადი ლოგიკური ერთეულებია: კურსი (CourseController.cs), ფაკულტეტი (DepartmentController.cs), ლექტორი (InstructorController.cs), სტუდენტი (StudentController.cs). ამ ობიექტებზე დაშვებული ფუნქცია განსაზღვრულია Controllers დირექტორიაში გაერთიანებულ Controller-კლასებში. მაგალითად, ახალი სტუდენტის დამატება, რედაქტირება, წაშლა და დათვალიერება – ყველა ეს ფუნქცია რეალიზებულია StudentController.cs კონტროლერ ფაილში (ნახ.6.8).



ნახ.6.8. Controllers დირექტორიაში თავმოყრილია აპლიკაციის ყველა Controller-კლასი

- **Views დირექტორია**



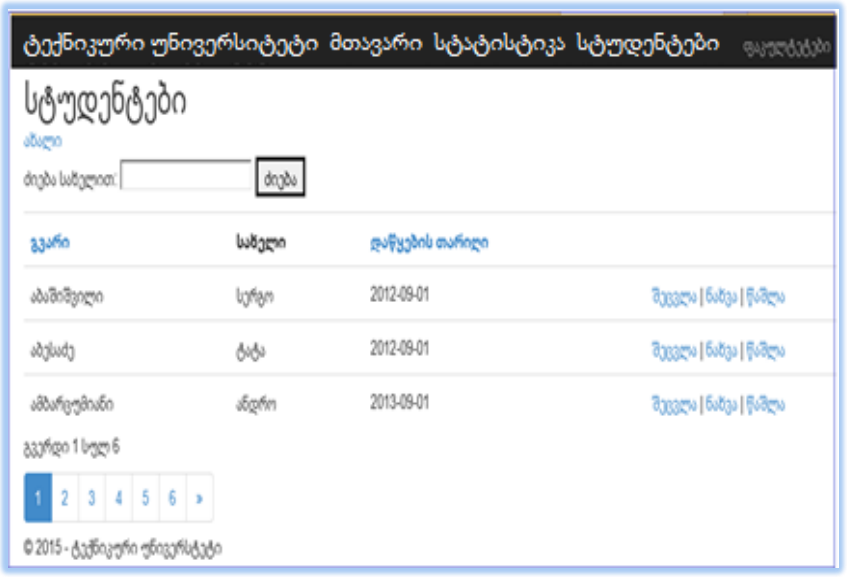
ნახ.6.9. View დირექტორიაში თავმოყრილია აპლიკაციის ყველა View ფაილი (cshtml)

პრეზენტაციის დონის წარმოდგენა (View) კიდევ ერთი მნიშვნელოვანი განმასხვავებელი მომენტია ASP.NET Web Forms ტექნოლოგიასა და ASP ტექნოლოგიას შორის. Web Forms აპლიკაციებში პრეზენტაციის დონეზე გვხვდება .aspx გაფართოების ფაილები, ხოლო პროგრამული ლოგიკა გატანილია .aspx-ფაილების უკანა მხარეს - .aspx.cs ფაილებში (code behind).

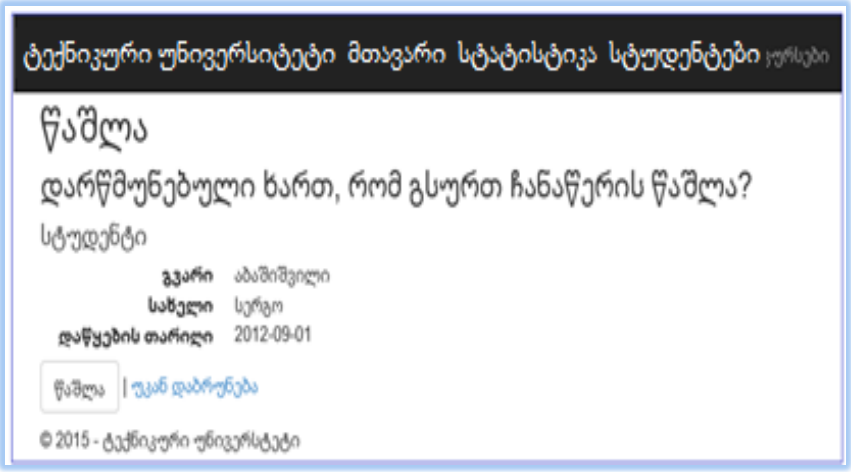
MVC-აპლიკაციებში პრეზენტაციის დონე რეალიზებულია .cshtml გაფართოების ფაილებით, ხოლო პროგრამული ლოგიკა გატანილია Controllers დონეზე. ამასთანავე .cshtml ფაილების რაოდენობა დამოკიდებულია შესასრულებელი ლოგიკური ოპერაციების რაოდენობაზე. მაგალითად სტუდენტების რეესტრი იმპლემენტირებულია Students – დირექტორიაში გაერთიანებული .cshtml View-ების ერთობლიობით (ნახ.6.10) [38].

თითო ოპერაციისთვის განკუთვნილია თითო ფაილი (ნახ.6.11, 6.12):

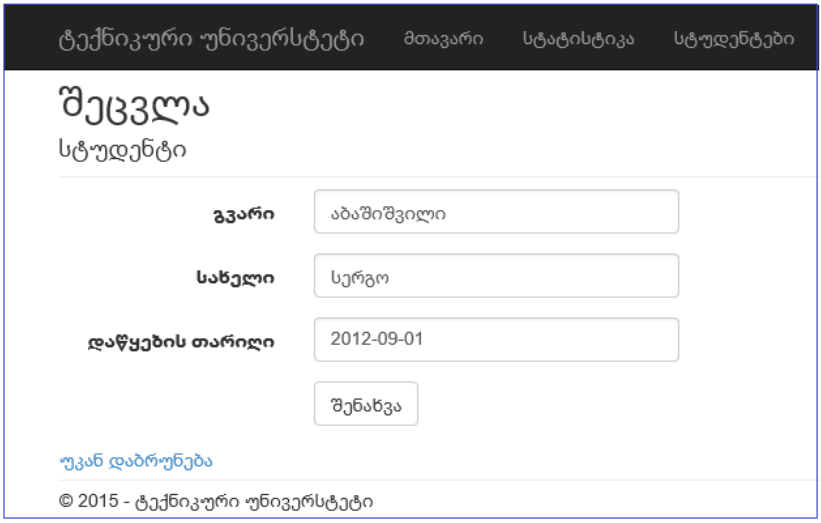
- Create.cshtml – ახალი სტუდენტის დამატების ფორმა;
- Delete.cshtml – სტუდენტის წაშლის ფორმა;
- Details.cshtml – სტუდენტის დეტალური ინფორმაციის ამსახველი ფორმა;
- Edit.cshtml – სტუდენტის ინფორმაციის რედაქტირების ფორმა;
- Index.cshtml – მთავარი View - სტუდენტების სია. თითოეული სტუდენტის ჩანაწერის გასწვრივ დატანილია ღილაკები, რომელზე დაჭერისას მომხმარებელი მოხვდება დანარჩენ .cshtml View-ებზე.



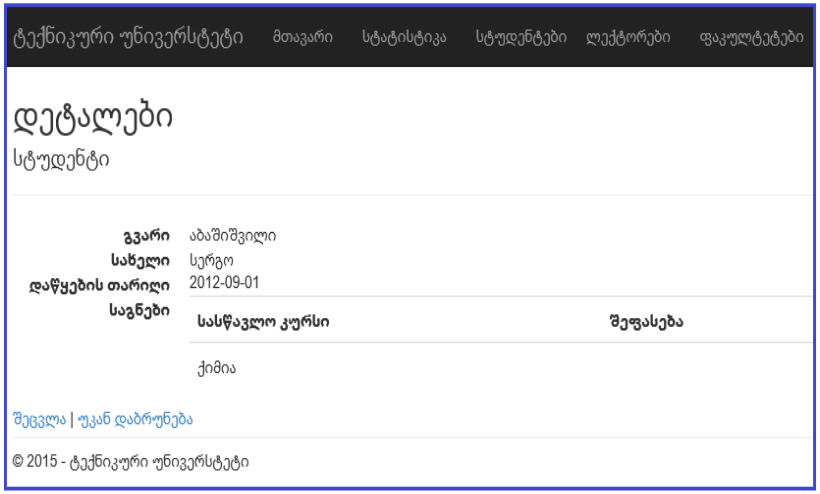
ნახ.6.10. სტუდენტების სია (Index.cshtml)



ნახ.6.11. სტუდენტების წაშლის ფორმა (Delete.cshtml)



ნახ.6.12. სტუდენტის რედაქტირების გვერდი (Edit.cshtml)



ნახ.6.13. სტუდენტის დეტალური ინფორმაციის გვერდი (Details.cshtml)

```
@model TechnicalUniversity.Models.Student
@{ ViewBag.Title = "Details";}
<h2>დეტალები</h2>
<div>
    <h4>სტუდენტი</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt> @Html.DisplayNameFor(model => model.LastName) </dt>
        <dd> @Html.DisplayFor(model => model.LastName) </dd>
        <dt> @Html.DisplayNameFor(model => model.FirstMidName) </dt>
        <dd> @Html.DisplayFor(model => model.FirstMidName) </dd>
        <dt> @Html.DisplayNameFor(model => model.EnrollmentDate) </dt>
        <dd> @Html.DisplayFor(model => model.EnrollmentDate) </dd>
        <dt> საგნები </dt>
        <dd>
            <table class="table">
                <tr>
                    <th>სასწავლო კურსი</th>
                    <th>შეფასება</th>
                </tr>
                @foreach (var item in Model.Enrollments)
                {
                    <tr>
                        <td>
                            @Html.DisplayFor(modelItem => item.Course.Title)
                        </td>
                        <td>
                            @Html.DisplayFor(modelItem => item.Grade)
                        </td>
                    </tr>
                }
            </table>
        </dd>
    </dl>
</div>
<p>
    @Html.ActionLink("შეცვლა", "Edit", new { id = Model.ID }) |
    @Html.ActionLink("უკან დაბრუნება", "Index")
</p>
```

ნახ.6.14. დეტალური ინფორმაციის გვერდის კოდი

როგორც ნახაზიდან ჩანს, .cshtml ფაილში HTML კოდი საკმაოდ სუფთად და მარტივად არის წარმოდგენილი. ფაილის წონა მსუბუქია და იოლად მართვადი, რაც MVC აპლიკაციების ერთ-ერთი მთავარი ღირსებაა.

6.3. Web-აპლიკაციაში რეპორტების ინტეგრაცია SQL Server Reporting Services ბაზაზე

თანამედროვე, მაღალკონკურენტულ ბიზნეს გარემოში ინფორმაციის ფლობას გადამწყვეტი მნიშვნელობა ენიჭება. ტექნოლოგიის განვითარებასთან ერთად ინფორმაციის შეგროვება გაიოლდა, თუმცა მისი გაანალიზება კვლავ რჩება რთულ და ფაქიზ თემად.

ბიზნეს ანალიზის და ანგარიშგებათა (რეპორტების) შემუშავებისთვის გადამწყვეტი ფაქტორია კარგი, მოქნილი სამუშაო ინსტრუმენტების (Business Intelligence Tools) არსებობა და მათი ეფექტური გამოყენება [54,56, 178-180].

- **SQL Server Reporting Services**

SQL Server Reporting Services (SSRS) არის Microsoft კომპანიის ინსტრუმენტი, რომელიც შემუშავებულია მონაცემთა ანალიზისა და ანგარიშგებების გენერაციის მიზნით [181,182]. იგი ბიზნეს ანალიზის პლატფორმის (BI) ერთ-ერთი კომპონენტია და მთლიანობაში იძლევა მონაცემთა ანალიზის მოქნილ საშუალებებს. ეს კომპონენტებია:

- SQL Server: ტრადიციული მონაცემთა ბაზის მანქანა, რომელზეც ასევე ინახება SSRS რეპორტების კატალოგი;
- SQL Server Analysis Services (SSAS): კომპონენტი ასრულებს ისეთ ანალიზურ პროცესებს, როგორიცაა მონაცემთა

აგრეგაცია და წარმოდგენა სხვადასხვა ჭრილში (მაგალითად, გეოგრაფიული მდებარეობა, დრო);

- *SQL Server Integration Services (SSIS)*: კომპონენტი გამოიყენება მონაცემთა ამოსაღებად, ტრანსფორმირების და ჩატვირთვის მიზნით (Extract, Transform, Load - ETL);
- *SQL Server Reporting Services (SSRS)*: სერვერზე დაფუძნებული, განვრცობადი პლატფორმაა, რომელშიც ხდება ინფორმაციის დამუშავება, ფორმატირება და მომხმარებლისთვის ტრადიციული თუ ინტერაქტიული ფორმატით მიწოდება. SSRS კონფიგურირებადია და ხასიათდება მრავალი ფუნქციით, როგორიცაა მაგალითად, მონაცემთა ვიზუალიზაცია დიაგრამებისა და გრაფიკების სახით, ანგარიშგებათა სხვადასხვა ფორმატით ექსპორტირება (HTML, Excel, PDF), მომხმარებლის ელ-ფოსტაზე გადაგზავნა, დაკონფიგურირება და SharePoint კორპორატიულ პორტალებში ჩაშენება.

➢ რეპორტების ინტეგრაცია Web-აპლიკაციებში

Microsoft-ის კომპანიამ რეპორტინგის სერვისის (Reporting Services) დაპროექტებისას თავიდანვე გაითვალისწინა მისი განვრცობადობის აუცილებლობა და სერვისის ღია ინტერფეისით წვდომადი გახადა ვებ-აპლიკაციებისა და დაპროგრამების სამუშაო გარემოთა ფართო სპექტრი [183].

განვიხილოთ რეპორტინგის ფუნქციით აპლიკაციის გამდიდრების 3 გავრცელებული ვარიანტი:

- Reporting Server Web Service (ასევე ცნობილია როგორც Reporting Services SOAP API);
- ReportViewer კონტროლი;
- წვდომა URL გზავნილის საშუალებით, და ა.შ. [38].

6.4. ERM & ORM დაპროექტების პროცესი

6.2 პარაგრაფში ჩვენ განვიხილეთ MVC აპლიკაციის აგების ამოცანა „უნივერსიტეტის“ საპრობლემო სფეროსთვის. საინფორმაციო სისტემის შექმნა ან არსებული სისტემის მოდიფიკაცია ინტეგრაციის პრინციპების საფუძველზე მოითხოვს ამ დომეინის ობიექტ-ორიენტირებული, პროცეს-ორიენტირებული და სერვის-ორიენტირებული მიდგომების კომპლექსურ გამოყენებას [54].

სისტემის შესაბამისი ინფრასტრუქტურის დასამუშავებლად კი აუცილებელია დღეისათვის არსებული *უსაფრთხოების ისეთი სტანდარტებისა და მეთოდოლოგიების გამოყენება, როგორიცაა BSI, ITIL, COBIT*, რაც საბოლოო ჯამში უზრუნველყოფს უსაფრთხო განაწილებული ინფორმაციული სისტემის შექმნას, მის შემდგომ მასშტაბირებას და განვითარებას [18].

მეორე მხრივ, ინტეგრაციის პროცესში სისტემის ცალკეული კომპონენტების დასამუშავებლად დროითი პარამეტრების და *პროგრამული პროდუქტის ხარისხის გასაუმჯობესებლად* აუცილებელი ხდება თანამედროვე CASE ტექნოლოგიების გამოყენება, როგორიცაა მაგალითად, Enterprise Architect (BPMN/UML-ის ინსტრუმენტული საშუალება), Natural Object Role Modeling Architect (NORMA) და სხვ., რომლებიც მაიკროსოფთის Visual Studio.NET Framework პაკეტის სამუშაო გარემოს თავსებადია [178, 184,185].

ქვემოთ მოცემულია „უნივერსიტეტის“ მართვის საინფორმაციო სისტემის საპილოტო ვერსიის მაგალითზე

მონაცემთა განაწილებული ბაზის და მომხმარებელთა ინტერფეისების დაპროექტების და პროგრამული რეალიზაციის პროცედურების დეტალური აღწერა ზემოაღნიშნული ტექნოლოგიებისა და SOA-ის საფუძველზე.

პროგრამული აპლიკაციის სასიცოცხლო ციკლის ეტაპების შესაბამისად (ანალიზი, დაპროექტება, დეველოპმენტი, ტესტირება, დანერგვა, თანხლება), საუნივერსიტეტო მართვის საინფორმაციო სისტემის შექმნა, UML-ტექნოლოგიით, ითვალისწინებს სისტემის ფუნქციონალური და არაფუნქციონალური მოთხოვნილებების განსაზღვრას, ობიექტორიენტირებული ანალიზის და დაპროექტების განხორციელებას, შესაბამისად სისტემასთან მომხმარებელთა მუშაობის ინტერაქტიული სცენარების, კლასთა-ასოსიაციების და მდგომარეობათა დიაგრამების აგებით სხვადსხვა შემთხვევით მოვლენათა შესაბამისად [75, 184].

თუ ობიექტორიენტირებული მოდელირების ტერმინებით ვისარგებლებთ, დასმული ამოცანის გადაწყვეტის „გასაღები“ კლასების, ობიექტების, კლასთაშორისი კავშირების, ობიექტ-როლური და არსთა-დამოკიდებულების მოდელებისა და სხვა სახის დიაგრამების აგებაა.

შემდეგ კი, კლასთა-ასოციაციებისა და არსთა-დამოკიდებულების დიაგრამათა საფუძველზე განხორციელდება მიზნობრივი სისტემის პროგრამული *კოდების რეალიზაციის ავტომატიზებული პროცესი ტესტირებით* [186,187].

ამგვარად, ჩვენ განვიხილეთ კონკრეტული ინფოსისტემის („*უნივერსიტეტი*“) აგების ამოცანათა სპექტრი და ამ

პროცესში გამოვყავით *პროგრამული სისტემის ტესტირების ფუნქციური ამოცანა,* მისი რეალიზაციით [188].

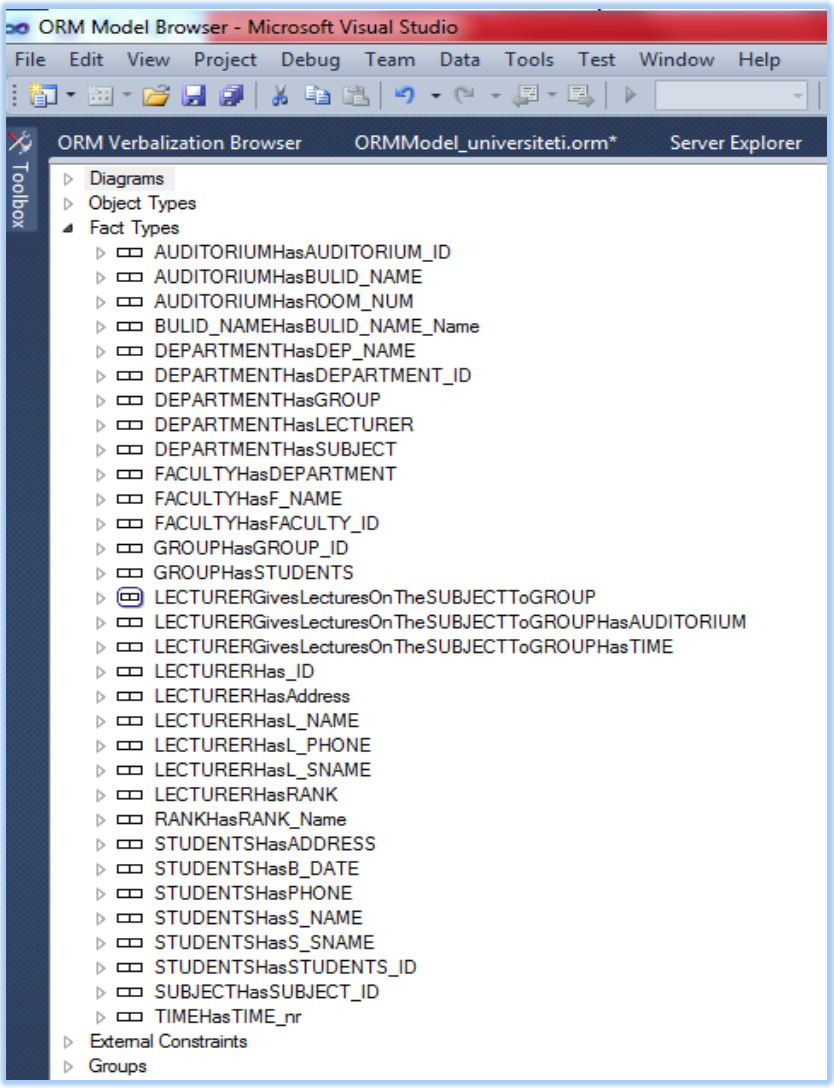
6.4.1. სისტემის ობიექტ-როლური მოდელის (ORM) დაპროექტება

განვიხილოთ ზოგადად „უნივერსიტეტის“ კლასის ობიექტისათვის მონაცემთა განაწილებული ბაზის დაპროექტების ამოცანა. უნივერსიტეტის საპრობლემო სფეროს არაფორმალიზებული აღწერის (ტერმინთა ლექსიკონი) ობიექტებია: *ფაკულტეტი, დეპარტამენტი, სტუდენტი, ლექტორი, საგანი (აკადემიური დისციპლინები, რომლებიც იკითხება შესაბამისი კათედრებსა და სპეციალობების მიხედვით), სასწავლო გეგმები, სილაბუსები (პროგრამები), ლექციები, პრაქტიკული და ლაბორატორიული სამუშაოები, აუდიტორიები, გამოცდები, ტესტირება* და ა.შ.

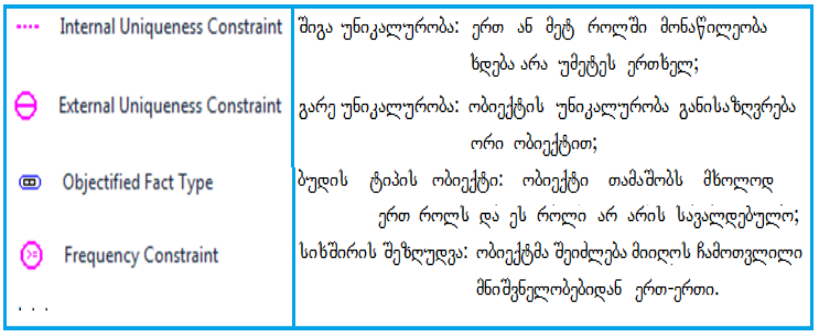
ობიექტ-როლური მოდელირების თეორიის წესების თანახმად საჭიროა აიგოს უნივერსიტეტის სასწავლო პროცესის შესაბამისი ORM-დიაგრამა. ამისათვის ფაქტ-შეზღუდვების ერთობლიობით (რომელსაც ადგენს სისტემის ბიზნეს-ანალიტიკოსი და საბოლოო მომხმარებელი), რომლებშიც ასახულია საპრობლემო სფეროს შესახებ ცოდნა (კლასებისა და ობიექტების ძირითადი ტერმინები და ქცევის წესები, დებულებით დადგენილი კანონზომიერებები და სხვ.), გადაიტანება Ms_Visual Studio.NET Framework სამუშაო გარემოში, ობიექტ-როლური მოდელის, NORMA-პაკეტის (Natural ORM Architect) ინტერფეისზე [185]. მაგალითად, ასეთი ფაქტები შეიძლება იყოს:

f1 : ლექტორს აქვს გვარი
f2 : ლექტორი მუშაობს დეპარტამენტში
f3 : ლექტორს აქვს თანამდებობა
f4 : ლექტორს აქვს ტელეფონი
f5 : ლექტორს აქვს ელ-ფოსტა
f6 : ლექტორს აქვს ხარისხი
f7 : ლექტორი კითხულობს საგანს
f8 : ლექტორი ასწავლის #-ჯგუფს
f8 : სტუდენტი არის #-ჯგუფში
. . .
f50 : ლექტორი მუშაობს #-დეპარტამენტში
f51 : დეპარტამენტი ეკუთვნის #-ფაკულტეტს
f52 : #-ჯგუფი ეკუთვნის #-დეპარტამენტს
f53 : ფაკულტეტს აქვს დასახელება
. . .
f100 : სტუდენტი არ შეიძლება იყოს ერთზე მეტ ჯგუფში
f101 : ლექტორი არ შეიძლება იყოს სრულ შტატზე ერთზე მეტ დეპარტამენტში
f102 : ლექტორი დროის ერთ მომენტში არ შეიძლება იყოს ორ სხვადასხვა აუდიტორიაში
და ა.შ.

6.15 ნახაზზე მოცემულია Visual Studio.NET სამუშაო გარემოში ORM-ით მიღებული შედეგები „უნივერსიტეტის“ ფაქტების შეტანის საფუძველზე. 6.16 ნახაზზე ასახულია ORM-ის თეორიაში არსებული შეზღუდვის მაგალითი, რომლებიც გამოყენებულია ჩვენ სქემაში. შეზღუდვების აღწერა ხდება კატეგორიალური მიდგომის საფუძველზე, რომელიც აერთიანებს სალაპარაკო ენის ფორმალური გრამატიკის და ლოგიკურ-ალგებრულ წესებს. შედეგად მიიღება ერთ-, ორ- ან n-ადგილიანი პრედიკატები [67].



ნახ.6.15. „უნივერსიტეტის“ ფაქტების აღწერის ფრაგმენტი



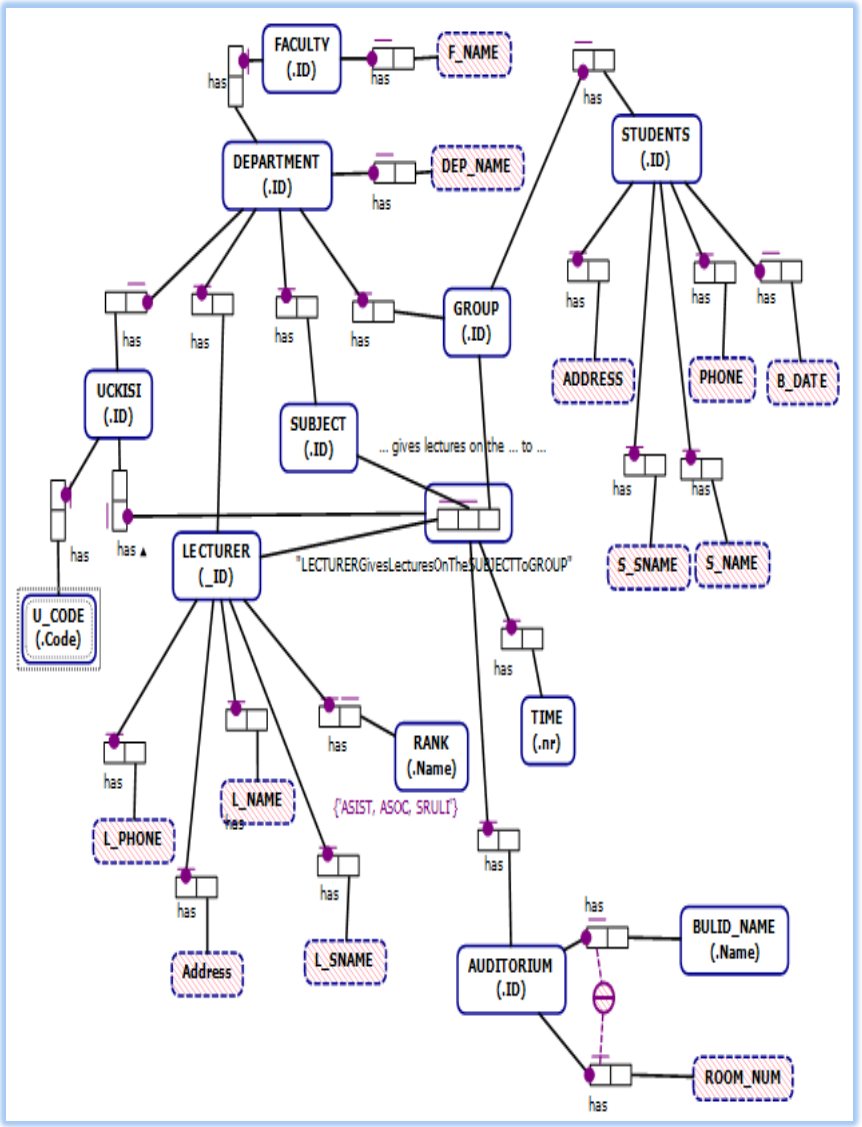
ნახ.6.16. შეზღუდვების აღწერის მაგალითები ORM-დიაგრამაზე

ზემოჩამოთვლილი ფაქტებიდან NORMA ინსტრუმენტი გვაძლევს შემდეგი სახის ORM-დიაგრამას (ნახ.6.17). აქ შესაძლებელია ახალი ფაქტის დამატება, არსებულის მოდიფიკაცია / წაშლა.

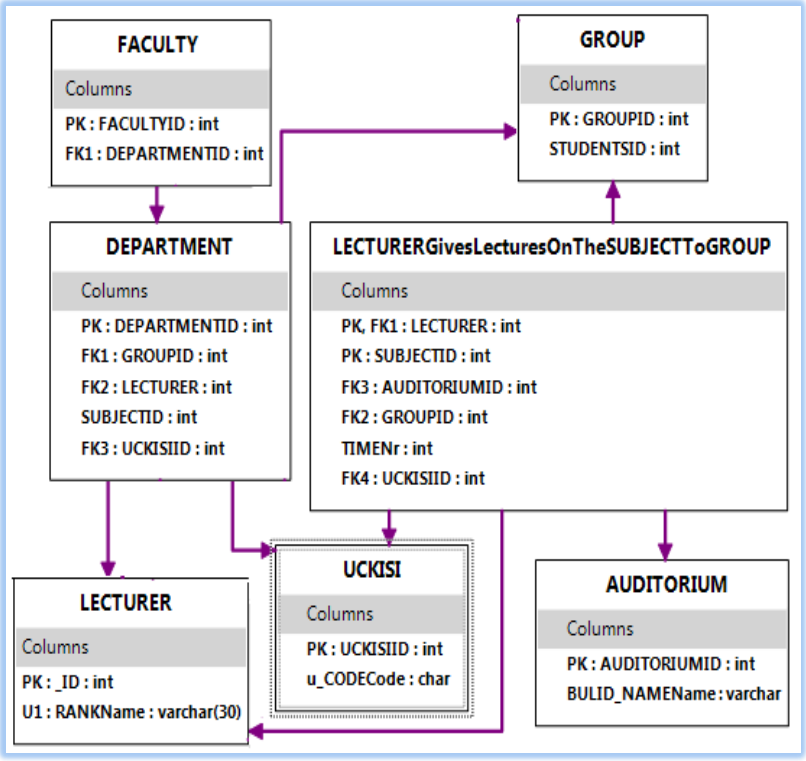
6.4.2. არსთა-დამოკიდებულების მოდელის (ERM) დაპროექტება

მომდევნო ეტაპზე განხორციელდება ობიექტ-როლური მოდელის ავტომატიზებული გადაყვანა არსთა-დამოკიდებულების მოდელში (ORM -> ERM). სისტემის დამპროექტებელი გააქტიურებს NORMA პაკეტის მენიუდან გენერაციის პროცედურას. ORM-დიაგრამიდან გენერირებული ERM-დიაგრამა მოცემულია 6.18 ნახაზზე.

შესაბამისად გამოკვეთილია შვიდი ობიექტი (არსი – Entity): ფაკულტეტი (Faculty), დეპარტამენტი (Department), ჯგუფი (Group), სტუდენტი (Students), ლექტორი (Lecturer), საგანი (Subject), აუდიტორია (Auditorium), საგამოცდო_უწყისი (Uckisi) და ა.შ..



ნახ.6.17. „უნივერსიტეტის“ ORM დიაგრამის ფრაგმენტი
(საპრობლემო სფეროს კონცეპტუალური მოდელი-1)



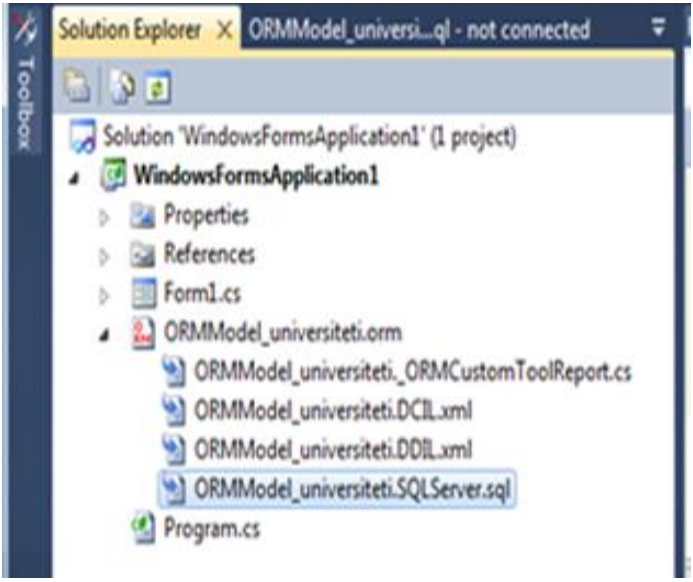
ნახ.6.18. ORM დიაგრამიდან გენერირებული ER-მოდელი (საპრობლემო სფეროს კონცეპტუალური მოდელი-2)

დიალოგში შესაძლებელია ER-მოდელის ობიექტების (Tables) განლაგების შეცვლა, რათა ვიზუალურად უფრო მოხერხებული მდებარეობა მიიღოს თითოეულმა, ობიექტთა-შორისი კავშირების რაც შეიძლება ნაკლები გადაკვეთებით. ცხრილებში ჩანს ობიექტის სახელი და ატრიბუტთა დასახელებები, ტიპების მითითებით. აგრეთვე ასახულია პირველადი (PK) და მეორეული (FK) გასაღებური ატრი-ბუტები და ა.შ.

6.4.3. მონაცემთა ბაზის სერვერზე განთავსება

მონაცემთა ბაზის კონცეპტუალური ERM სქემის აგების შემდეგ საჭიროა მის საფუძველზე სისტემის მიერ დაიწეროს შუალედური ტექსტური ტიპის DDL-ფაილი (Data Definition Language), რომელიც მომავალში SQL Server მონაცემთა ბაზების მართვის სისტემამ უნდა გამოიყენოს.

ამგვარად, ER-დიაგრამიდან, ცხრილებითა და ატრიბუტებით, ავტომატურად გენერირდება .DDL ფაილები, რომლებიც შემდგომ სტრუქტურულად მოთავსდება Ms SQL Server მონაცემთა ბაზაში. 6.19 ნახაზზე ნაჩვენებია ამ ეტაპის პროცესის ინიცირების დიალოგური სქემა.



ნახ.6.19. DDL ფაილის გენერაცია ER-მოდელიდან MsVisual Syudio-ში

6.1 ლისტინგში მოცემულია ავტომატურად გენერირებული DDL-ფაილის ტექსტის ფრაგმენტი.

```
-- Listing_6.1 -----DDL file -------------------------
CREATE SCHEMA ORMModel1
GO
GO
CREATE TABLE ORMModel1.FACULTY
(  FACULTYID INTEGER IDENTITY (1, 1) NOT NULL,
   DEPARTMENTID INTEGER NOT NULL,
   CONSTRAINT FACULTY_PK PRIMARY KEY(FACULTYID)  )
GO
CREATE TABLE ORMModel1.DEPARTMENT
(  DEPARTMENTID INTEGER IDENTITY (1, 1) NOT NULL,
   GROUPID INTEGER NOT NULL,
   LECTURER INTEGER NOT NULL,
   SUBJECTID INTEGER IDENTITY (1, 1) NOT NULL,
   CONSTRAINT DEPARTMENT_PK PRIMARY KEY(DEPARTMENTID)   )
GO
CREATE TABLE ORMModel1."GROUP"
(  GROUPID INTEGER IDENTITY (1, 1) NOT NULL,
   STUDENTSID INTEGER IDENTITY (1, 1) NOT NULL,
   CONSTRAINT GROUP_PK PRIMARY KEY(GROUPID)       )
GO
CREATE TABLE ORMModel1.LECTURER
(  "_ID" INTEGER IDENTITY (1, 1) NOT NULL,
   RANKName NATIONAL CHARACTER VARYING(30) NOT NULL,
   CONSTRAINT LECTURER_PK PRIMARY KEY("_ID"),
   CONSTRAINT LECTURER_UC UNIQUE(RANKName),
   CONSTRAINT LECTURER_RANKName_RoleValueConstraint1 CHECK
(RANKName IN (N'ASIST, ASOC, SRULI'))  )
GO
CREATE TABLE
ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
(  LECTURER INTEGER NOT NULL,
   SUBJECTID INTEGER IDENTITY (1, 1) NOT NULL,
   AUDITORIUMID INTEGER NOT NULL,
   GROUPID INTEGER NOT NULL,
   TIMENr INTEGER NOT NULL,
   CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_PK
PRIMARY KEY(LECTURER, SUBJECTID)     )
GO
CREATE TABLE ORMModel1.AUDITORIUM
```

```
(  AUDITORIUMID INTEGER IDENTITY (1, 1) NOT NULL,
   BULID_NAMEName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
   CONSTRAINT AUDITORIUM_PK PRIMARY KEY(AUDITORIUMID)
)
GO
ALTER TABLE ORMModel1.FACULTY ADD CONSTRAINT FACULTY_FK
FOREIGN KEY (DEPARTMENTID) REFERENCES ORMModel1.DEPARTMENT
(DEPARTMENTID) ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.DEPARTMENT ADD CONSTRAINT
DEPARTMENT_FK1 FOREIGN KEY (GROUPID) REFERENCES
ORMModel1."GROUP" (GROUPID) ON DELETE NO ACTION ON UPDATE NO
ACTION
GO
ALTER TABLE ORMModel1.DEPARTMENT ADD CONSTRAINT
DEPARTMENT_FK2 FOREIGN KEY (LECTURER) REFERENCES
ORMModel1.LECTURER ("_ID") ON DELETE NO ACTION ON UPDATE NO
ACTION
GO
ALTER TABLE
ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP ADD
CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK1
FOREIGN KEY (LECTURER) REFERENCES ORMModel1.LECTURER ("_ID")
ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE
ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP ADD
CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK2
FOREIGN KEY (GROUPID) REFERENCES ORMModel1."GROUP" (GROUPID)
ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE
ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP ADD
CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK3
FOREIGN KEY (AUDITORIUMID) REFERENCES ORMModel1.AUDITORIUM
(AUDITORIUMID) ON DELETE NO ACTION ON UPDATE NO ACTION
GO
GO
```

შეიძლება ჩვთვალოთ, რომ ამ DDL ფაილის კოპირებით Ms SQL Server–ში შეიქმნება შესაბამისი ბაზა, ცხრილებით, ატრიბუტებით და კავშირებით. რა თქმა უნდა, შესაძლებელია

აქაც დამპროექტებლის ჩარევა ბაზის სტრუქტურის ზოგიერთი კომპონენტის შესასწორებლად, საჭიროების შემთხვევაში

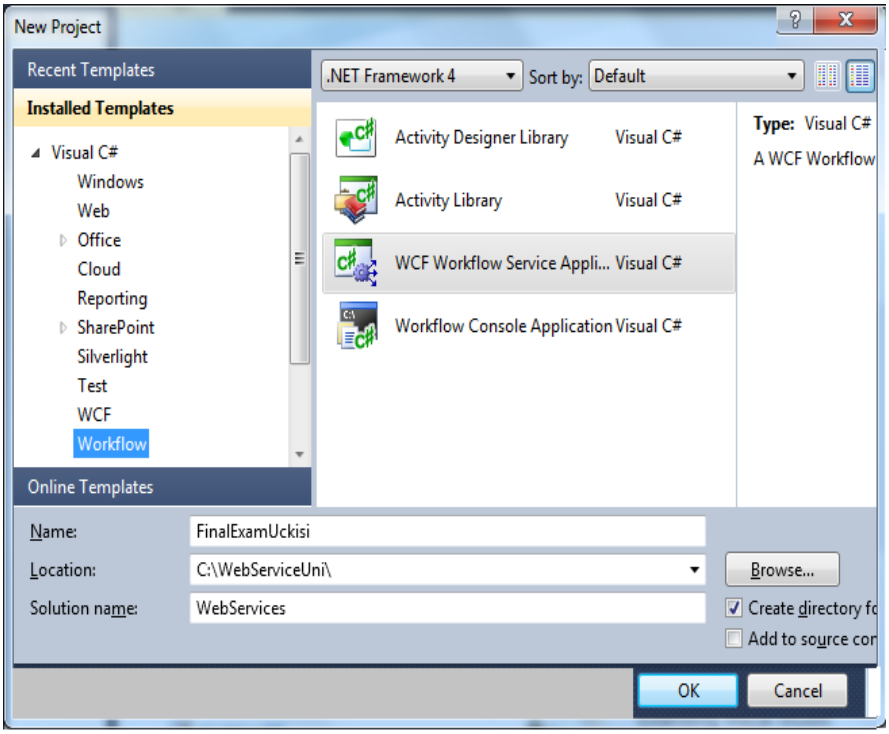
6.5. ბიზნეს-პროცესის სერვისის შექმნა

როგორც ცნობილია, ბიზნეს-პროცესი შეიძლება განთავსდეს ვებ-სერვისში, რომელიც უზრუნველყოფს ბიზნეს-პროცესის გადაწყვეტილების (შედეგის) მიწოდებას კლიენტებისთვის (ვებ-აპლიკაციებისთვის). ვებ-სერვისი იღებს მოთხოვნას კლიენტისგან, ასრულებს მის დამუშავებას და უბრუნებს პასუხს. ეს პროცედურები სრულდება *Receive* და *Send* ქმედებებით.

განვიხილოთ ჩვენი მაგალითის რეალიზაცია ჰიბრიდული ტექნოლოგიების: WPF და WCF ბაზაზე [75,189]. WPF (Windows Presentation Foundation) და WCF (Windows Communication Foundation) ტექნოლოგიები Visual Studio .NET Framework გარემოში ქმნის მომხმარებელზე ორიენტირებულ მაღალი ხარისხის დიზაინის აპლიკაციებს, C#.NET (ლოგიკის ნაწილი) და XAML (დიზაინის ნაწილი) ენების საფუძველზე [190,191].

ავამუშავოთ Visual Studio და შევქმნათ ახალი პროექტი WCF Workflow Service Application template გამოყენებთ. შევიტანოთ პროექტის სახელი FinalExamUckisi და solution-ის სახელი WebServices (ნახ.6.20).

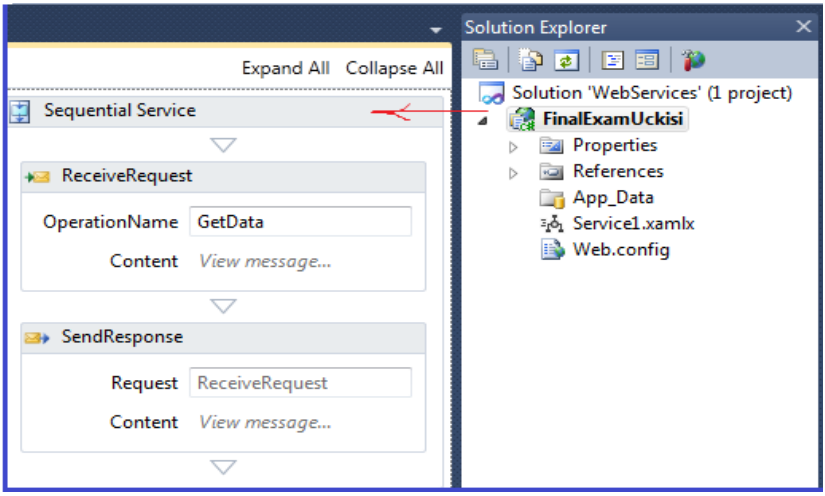
შეიქმნება საინიციალიზაციო workflow Sequence ბლოკი, რომელიც შეიცავს Receive და SendReply ქმედებებს, როგორც 6.21 ნახაზზეა ნაჩვენები.



ნახ.6.20. Visual Studio.NET-ში WCF-პროექტის შექმნა

თავიდან საჭიროა ამ ქმედებების კონფიგურაცია სერვისის კონტრაქტის განსაზღვრის მიზნით, რომელსაც ისინი დააკმაყოფილებს. შემდეგ დავამატოთ დამუშავების ბიზნესპროცესი, რომელიც განხორციელდება Receive და SendReply ქმედებებს შორის.

შაბლონი შექმნის საწყის ბიზნესპროცესს ფაილში, სახელით Service1.xamlx. შევცვალოთ Solution Explorer-ში ეს სახელი FinalExamUckisi.xamlx -ით.



ნახ.6.21. FinalExamUckisi პროექტის workflow Sequence ბლოკი

სერვისი, რომელიც მაგალითის სახით უნდა შევქმნათ, „საფინალო გამოცდისთვის“, ძებნის შესაბამის უწყისებს მითითებული აკადემიური ჯგუფის, ლექტორის და აკადემიური საგნის მიხედვით. ელექტრონული საგამოცდო უწყისები ეკუთვნის ფაკულტეტის დეკანატს, ხოლო მათი დამუშავება ხდება დეპარტამენტთა ლექტორების მიერ”.

➢ სერვისის კონტრაქტის განსაზღვრა

WCF სისტემებში იყენებენ კონტრაქტების სამ დონეს: მონაცემთა კონტრაქტი, შეტყობინებათა კონტრაქტი და სერვისის კონტრაქტი [192].

მონაცემთა კონტრაქტის დანიშნულებაა შეთანხმება კლიენტსა და სერვისს შორის ერთმანეთთან გასაცვლელ

მონაცემებზე (ითვალისწინებენ მონაცემთა სტრუქტურებს, პარამეტრებს, მოწესრიგებას და ა.შ.).

შეტყობინებათა კონტრაქტი უზრუნველყოფს SOAP (Simple Object Access Protocol) შეტყობინებების კონტროლს, რომელიც გამოიყენება ქსელში სხვადასხვა შეტყობინების გასაცვლელად XML ფორმატში. იგი უზრუნველყოფს აგრეთვე ინფორმაციის გაცვლის უსაფრთხოებას შეტყობინებების დონეზე.

სერვისის კონტრაქტი (ანუ კონტრაქტი მომსახურებისათვის) განსაზღვრავს ოპერაციათა სახეებს, რომლებსაც უზრუნველყოფს სერვისი. იგი კლიენტს აწვდის ასევე ინფორმაციას: შეტყობინებაში მონაცემთა ტიპების შესახებ, ოპერაციათა ადგილმდებარეობის შესახებ, ინფორმირების პროტოკოლისა და სერიალიზაციის ფორმატის შესახებ, შეტყობინებათა გაცვლის შაბლონების შესახებ (ცალმხრივი, ორმხრივი ან კითხვა/პასუხის ტიპებით).

ჩვენი პროექტისთვის სერვისის კონტრაქტის ასაგებად უნდა შევქმნათ საგამოცდო უწყისის ინფორმაციის კლასი C# ენაზე – UckisiInfo.cs. ამისთვის Solution Explorer-ში მარჯვენა ღილაკით FinalExamUckisi პროექტზე ვირჩევთ Add->Cla სახელით UckisiInfo.cs, რომლის ტექსტი მოცემულია 6.2 ლისტინგში.

```
// ----- ლისტინგი_6.2 – Service Contract ---
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
```

```
namespace FinalExamUckisi
{   // ---- სერვისის კონტრაქტის განსაზღვრა ---
IfinalExamUckisi,   რომელიც
    // --- შედგება ერთი მეთოდისგან  -  LookupUckisi () ------
   [ServiceContract]
   public interface IFinalExamUckisi
   {   [OperationContract]
      UckisiInfoList LookupUckisi(UckisiSearch request);    }
   //-- მოთხოვნის შეტყობინების განსაზღვრა , UckisiSearch ---
   [MessageContract(IsWrapped = false)]
   public class UckisiSearch
    {  private String _JGUPI;
      private String _Sagani;
      private String _Lectori;
      public UckisiSearch() {  }
      public UckisiSearch(String sagani, String lectori, String jgupi)
      {
         _Sagani  = sagani;
         _Lectori = lectori;
         _JGUPI   = jgupi;      }
      #region Public Properties
      [MessageBodyMember]
      public String Sagani
      {   get { return _Sagani; }
         set { _Sagani = value; }    }
      [MessageBodyMember]
      public String Lectori
      {   get { return _Lectori; }
         set { _Lectori = value; }   }
      [MessageBodyMember]
```

```
        public String JGUPI
        {   get { return _JGUPI; }
            set { _JGUPI = value; }     }
        #endregion Public Properties
    }
    // --- UckisiInfo კლასის განსაზღვრა -----
        [MessageContract(IsWrapped = false)]
    public class UckisiInfo
    {   private Guid _ExamUckisiID;
        private String _JGUPI;
        private String _Sagani;
        private String _Lectori;
        private String _Status;
        public UckisiInfo() {   }
        public UckisiInfo(String sagani, String lectori, String jgupi, String
status)
        {   _Sagani = sagani;
            _Lectori = lectori;
            _JGUPI = jgupi;
            _Status = status;
            _ExamUckisiID = Guid.NewGuid();    }
        #region Public Properties
        [MessageBodyMember]
        public Guid ExamUckisiID
        {   get { return _ExamUckisiID; }
            set { _ExamUckisiID = value; }     }
        [MessageBodyMember]
        public String Sagani
        {   get { return _Sagani; }
            set { _Sagani = value; }           }
```

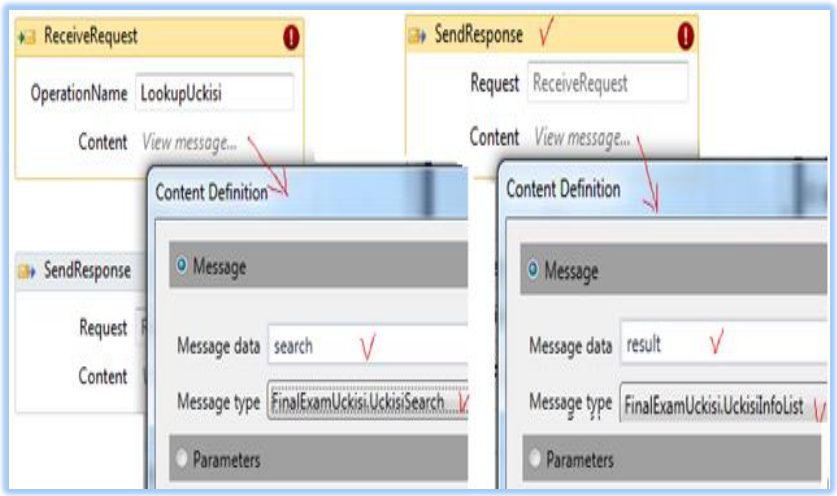
```
        [MessageBodyMember]
        public String Lectori
        {   get { return _Lectori; }
            set { _Lectori = value; }         }
        [MessageBodyMember]
        public String JGUPI
        {   get { return _JGUPI; }
            set { _JGUPI = value; }             }
        [MessageBodyMember]
        public String status
        {   get { return _Status; }
            set { _Status = value; }             }
        #endregion Public Properties
    }
    // ------ საპასუხო შეტყობინების განსაზღვრა --- UckisiInfoList ---
      [MessageContract(IsWrapped = false)]
    public class UckisiInfoList
    {   private List<UckisiInfo> _UckisiList;
        public UckisiInfoList()
        { _UckisiList = new List<UckisiInfo>();     }
        [MessageBodyMember]
        public List<UckisiInfo> UckisiList
        {    get { return _UckisiList; }         }
    }
}
```

სერვისის კონტრაქტი IFinalExamUckisi შეიცავს ერთადერთ მეთოდს LookupUckisi(). იგი მონაცემებს გადასცემს UckisiSearch კლასს, რომელსაც აქვს სხვადასხვა თვისებები, საჭირო საგამოცდო უწყისის მოსაძებნად, მაგალითად, ლექტორი, საგანი, ჯგუფი. ის აბრუნებს უკან UckisiInfoList

კლასს, რომელიც შეიცავს UckisiInfo კლასების კოლექციას. F6 ამოქმედებით აიგება გადაწყვეტა (solution). ამგვარად, ჩვენ განვსაზღვრეთ შეტყობინებები, რომლებიც გადაიცემა სერვის–მეთოდების მიერ პარამეტრების სახით.

- **Receive და SendReply კონფიგურირება**

შემდეგ ეტაპზე FinalExamUckisi.xamlx-ში ReceiveRequest ქმედებისთვის OperationName თვისებაში ვათავსებთ LookupUckisi სახელს. WorkflowService-დიზაინერში შევქმნით ორ ახალ ცვლადს (Variables): search ცვლადი, შემომავალი შეტყობინების შესანახად და result ცვლადი, გამოსატანი შედეგისთვის (ნახ.6.22). Message ბუტონი უნდა იყოს ჩართული და ტიპებიც არჩეული.



ნახ.6.22. შემავალი მოთხოვნის შეტყობინების და გამომავალი შედეგის ცვლადების განსაზღვრა

- **PerformLookup ქმედების აგება (ძებნის შესრულება)**

ძებნის განსახორციელებლად (მონაცემთა ბაზებთან მიმართვის მიზნითაც) ვქმნით ახალ კლასს PerformLookup.cs, რომელიც Solution Explorer-ში პროექტისთვის FinalExamUckisi აირჩევა Add->NewItem, Workflow-კატეგორიაში, Code Activity-ით. ამ ახალი კლასის ტექსტი მოცემულია 6.3 ლისტინგში.

```
// ---- ლისტინგი_6.3 --- PerformLookup ----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
namespace FinalExamUckisi
{
  public sealed class PerformLookup : CodeActivity
   {
    public InArgument<UckisiSearch> Search { get; set; }
   public OutArgument<UckisiInfoList> UckisiList
{get;set;}
   protected override void Execute(CodeActivityContext
                                                    context)
    { string lectori = Search.Get(context).Lectori;
      string sagani = Search.Get(context).Sagani;
      string jgupi = Search.Get(context).JGUPI;

      UckisiInfoList l = new UckisiInfoList();

     l.UckisiList.Add(new UckisiInfo(sagani, lectori,
jgupi, "Available"));
```

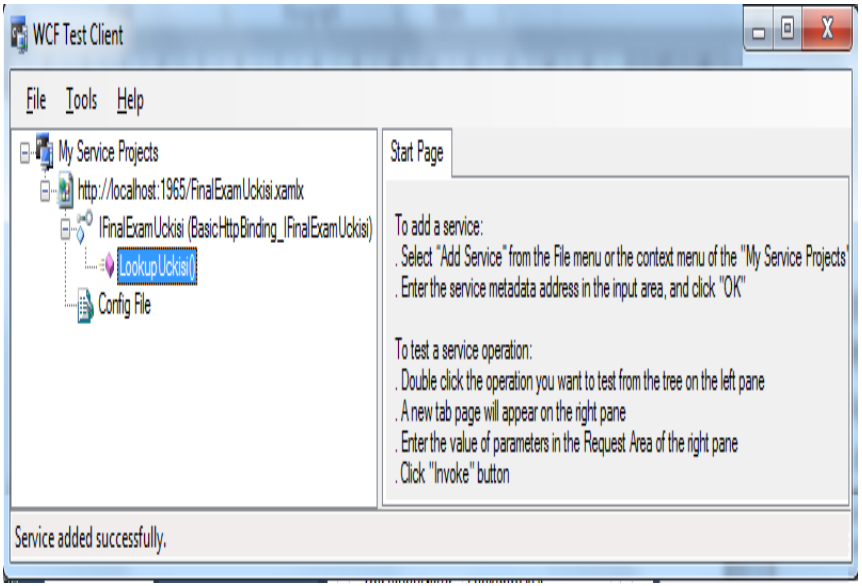
```
      l.UckisiList.Add(new UckisiInfo(sagani, lectori,
jgupi, "CheckedOut"));
     l.UckisiList.Add(new UckisiInfo(sagani, lectori,
jgupi, "Missing"));
     l.UckisiList.Add(new UckisiInfo(sagani, lectori,
jgupi, "Available"));
       UckisiList.Set(context, l);        }
    }
}
```

PerformLookup ქმედება ჩაემატება ინსტრუმენტების პანელზე. იგი უნდა გადმოვიტანოთ „ReceiveRequest" და „SendResponse" ქმედებებს შორის შესასრულებლად. ამასთანავე მისი Search თვისებისთვის ჩავწერეთ search, ხოლო UckisiList თვისებისთვის კი - result.

6.6. სერვისის ტესტირება

აპლიკაციის საბოლოო გამართვამდე უნდა მოვახდინოთ აგებული სერვისების ტესტირება. F5-ით ავამუშავოთ სერვისის გამართვის პროცედურა (debug). ვინაიდან ეს Web-სერვისია, Visual Studio ავტომატურად აამუშავებს WCF Test Client-ს [192].

ეს მეტად მოსახერხებელი უტილიტაა. იგი ჩატვირთავს Web-სერვისებს და აღმოაჩენს მეთოდებს, რომლებიც გათვალისწინებულია. ეს შედეგი ჩანს 6.23 ნახაზის მარცხენა პანელზე.

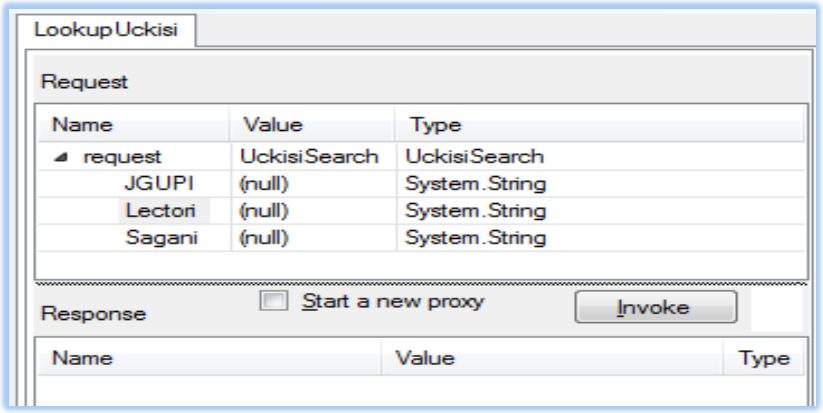


ნახ.6.23. ტესტირების პროცედურა

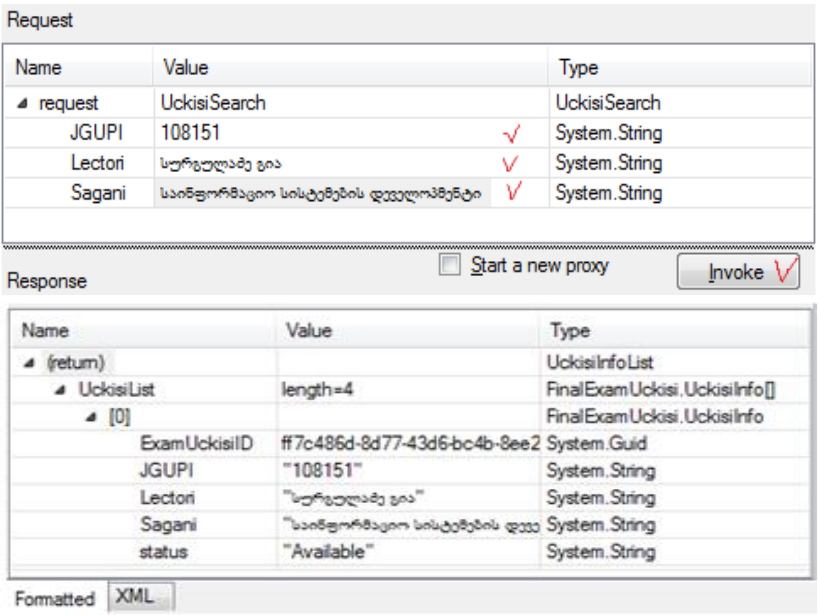
LookupUckisi() მეთოდზე 2-ჯერ დაჭერით მარჯვენა პანელის ზედა ნაწილში გამოიყოფა ადგილი შემოსული შეტყობინების განსათავსებლად (ნახ.6.24).

შევიტანოთ კონკრეტული მნიშვნელობები Lectori, JGUPI, Sagani და ავამოქმედოთ Invoke ღილაკი. ცხრილებში გამოჩნდება შედეგები (ნახ.6.25).

6.26 ნახაზზე ნაჩვენებია Request და Response ცხრილების შესაბამისი ფაილები XML ფორმატში.



ნახ.6.24. საწყისი მონაცემების შესატანი ფანჯარა



ნახ.6.25. WCF Client Test უტილიტით სერვისის ტესტირების შედეგების ნახვა

ნახ.6.26. ტესტირების შედეგების XML ტექსტები

ამგვარად, „უნივერსიტეტი“-ს მართვის საინფორმაციო სისტემის მაგალითზე შემუშავებულია მონაცემთა განაწილებული ბაზის და მომხმარებელთა ინტერფეისების ავტომატიზებული დაპროექტების და პროგრამული რეალიზაციის პროცესები (ტესტირების პროცედურების გათვალისწინებით), ჰიბრიდული აპლიკაციების (Windows- და Web-სისტემების) აგების ტექნოლოგიებისა და სერვის-ორიენტირებული არქიტექტურის საფუძველზე. მომხმარებელთა ინტერფეისების Web-სერვისები აგებულია WPF, Workflow და WCF ტექნოლოგიებით.

VII თავი

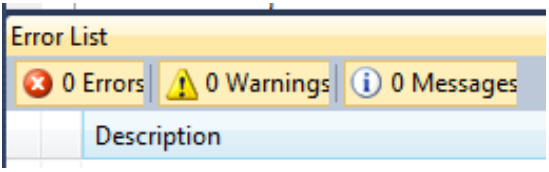
პროგრამული აპლიკაციის გამართვა, ტესტირება და მისი კოდის ხარისხის შეფასება

7.1. პროგრამული აპლიკაციის გამართვა: შეცდომებისა და გამონაკლის შემთხვევათა აღმოჩენა და გამორიცხვა

პროგრამებისა და აპლიკაციების გამართვის პროცესში გამონაკლისი შემთხვევებისა და შეცდომების აღმოჩენა და მათი გამორიცხვა.

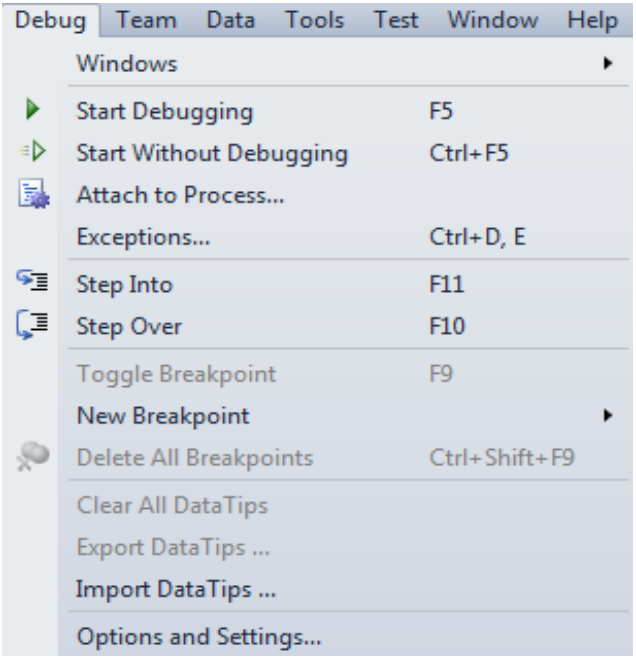
როგორც ცნობილია, პროგრამის გამართვისა და ტესტირების (შესრულების) პროცესში ადგილი აქვს პროგრამული შეცდომების გამოვლენას. ეს შცდომები სამი სახისაა: *სინტაქსური, პროცედურული და ლოგიკური.*

სინტაქსური შეცდომების აღმოჩენა ხდება C#-ენის კომპილატორის საშუალებით და გამოიტანება პროგრამული ტექსტების რედაქტორის ქვედა ნაწილში, Error List ფანჯარაში (ნახ.7.1). მათი პოვნა და შესწორება შედარებით ადვილია.



ნახ.7.1

პროცედურული შეცდომები ვლინდება პროგრამის შესრულების პროცესში, როცა პროგრამაში აღარაა სინტაქსური შეცდომები და ხდება მისი ამუშავება: Start Debugging ან F5 (ნახ.7.2), შესაძლებელია ისეთი შეცდომის გამოვლენა, რომელიც წყვეტს პროგრამის შესრულების პროცესს (ანუ სრულდება ავარიულად).

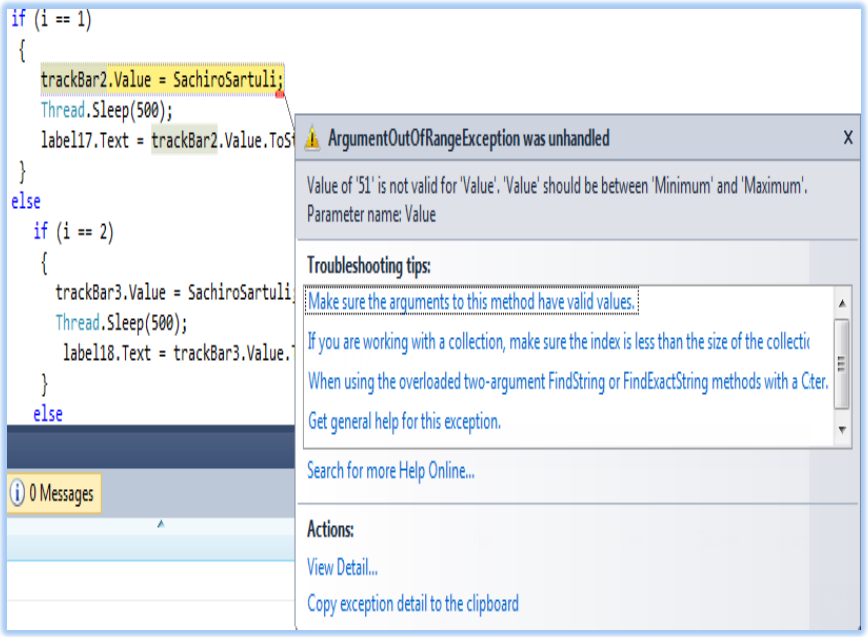


ნახ.7.2

დებაგერს გამოაქვს ამ დროს სათანადო შეტყობინება (ნახ.7.3), რომელიც პროგრამისტის მხრიდან მოითხოვს ანალიზსა და შეცდომის გამორიცხვას (Exception Handling).

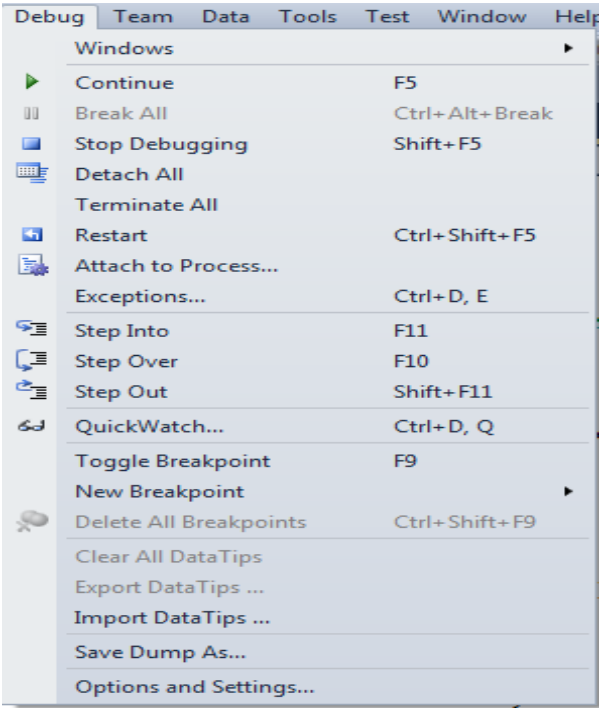
ლოგიკური შეცდომების აღმოჩენა შედარებით რთულია. პროგრამა ამ დროს მუშაობსა და სრულდება ნორმალურად (არაავარიულად), მაგრამ შედეგები „საეჭვოა“.

ტესტირების პროცესში, რომელიც აუცილებლად მოსდევს პროგრამის გამართვას, საჭიროა ასეთი „კვლევის“ ჩატარება, რათა გამოვლენილ იქნას მოსალოდნელი, ფარული ლოგიკური შეცდომები.



ნახ.7.3

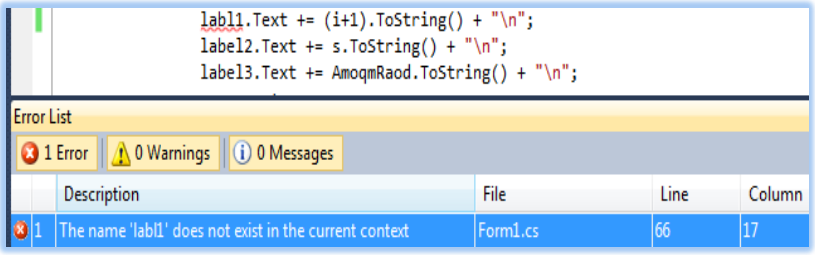
C# ენის რედაქტორს აქვს კარგი ინსტრუმენტული საშუალებები (Debugger) ამ ტიპის შეცდომების მოსაძებნად და აღმოსაფხვრელად (ნახ.7.4).



ნახ.7.4

1) *სინტაქსური შეცდომების აღმოფხვრის საშუალებანი:*

თუ პროგრამის არაკორექტულ კოდში შეცდომითაა ჩაწერილი ენის ოპერატორი (მაგალითად, “Whail” ნაცვლად while -ისა) ან კონსტრუქცია (მაგალითად, “case: “, რომელსაც არ უძღვის წინ switch() {…}) და ა.შ. როგორც ზემოთ აღვნიშნეთ, ამ დროს კომპილატორი მიუთითებს არსებულ შეცდომასა და მის ადგილმდებარეობას (ნახ.7.5).



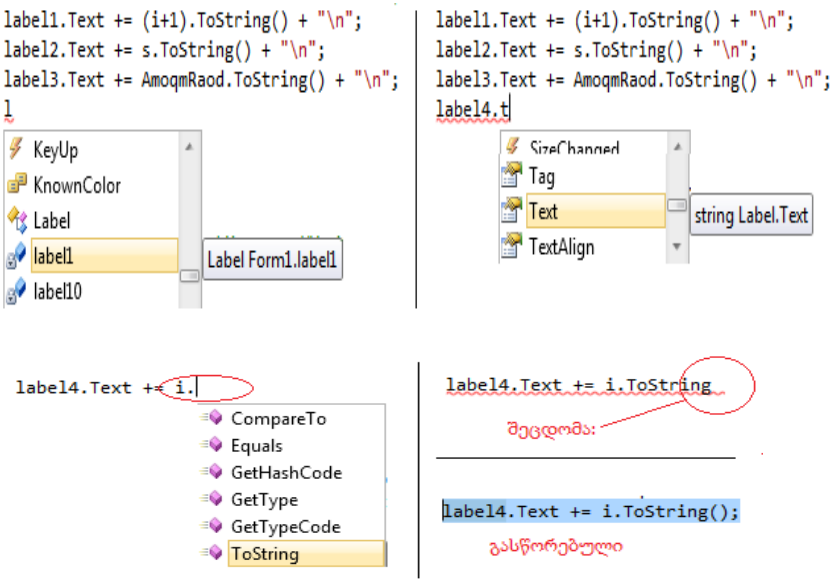
ნახ.7.5

სინტაქსური შეცდომები თუ არ გასწორდა, მაშინ ვერ მოხერხდება პროგრამის ობიექტური (.obj) და შესრულებადი (.exe) კოდების ფორმირება.

სინტაქსური შეცდომების თავიდან ასაცილებლად C# ენის რედაქტორს აქვს სხვადასხვა ვიზუალური დამხმარე საშუალება. მაგალითად, პროგრამაში ოპერატორების ტექსტის შეტანისას, ან ობიექტის მეთოდისა და მოვლენის არჩევისას (წერტილის „.“ დასმისას) ხდება ვიზუალური ბლოკის (Intellisense - ავტოდამატება) შემოთავაზება (ნახ.7.6), საიდანაც ამოირჩევა საჭირო სიტყვა და Enter-კლავიშით სწრაფად ჩაისმება მითითებულ ადგილას.

ეს გამორიცხავს როგორც ოპერატორის (ობიექტის თვისების, მეთოდისა და ა.შ.) არასწორ სინტაქსურ ჩაწერას, ისე არარელევანტური სიტყვის მითითებას (სიტყვა, რომელიც აქ „უადგილოა“).

შესაძლებლია აგრეთვე კოდში „გახსნილ-დასახურ“ ფრჩხილების რაოდენობის კონტროლი, რაც ძალზე ხშირი შეცდომების წყაროა. მთლიანად, შეიძლება ითქვას, რომ ენის ასეთი ვიზუალური კონტროლისა და დამხმარე საშუალებები პროგრამისტის მუშაობას ეფექტურს ხდის.



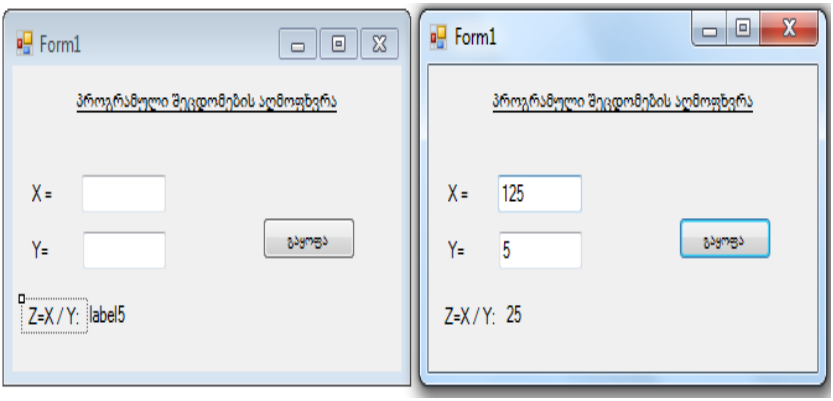
ნახ.7.6

2) განსაკუთრებული შემთხვევები: შეცდომები პროგრამის შესრულებისას და მათი აღმოფხვრა

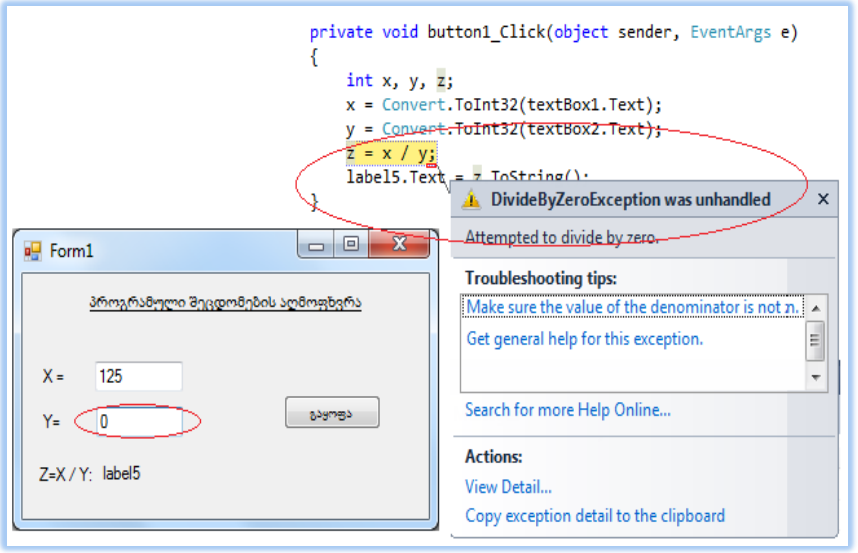
თუ პროგრამის ტექსტი სინტაქსური შეცდომებისაგან თავისუფალია, ის შეიძლება ამუშავდეს შესარულებლად. ამ დროს შესაძლებელია ისეთი შეცდომების გამოვლენა, რომლებიც პროგრამას ავარიულად დაასრულებს ან საერთოდ არ დაასრულებს („გაჭედავს“). მაგალითად, უსასრულო ციკლი ან სხვ. განსაკუთრებული შემთხვევა. განვიხილოთ ასეთი მაგალითები:

ამოცანა_1: ავაგოთ პროგრამის კოდი, რომელიც შეასრულებს მთელი რიცხვების შეტანასა და გაყოფის ოპერაციას. 7.7 ნახაზზე ნაჩვენებია ფორმა ორი ტექსტბოქსით

(შესატანად), label5 შედეგის გამოსატანად და ღილაკი „გაყოფა“ პროგრამული კოდით (ნახ.7.8).



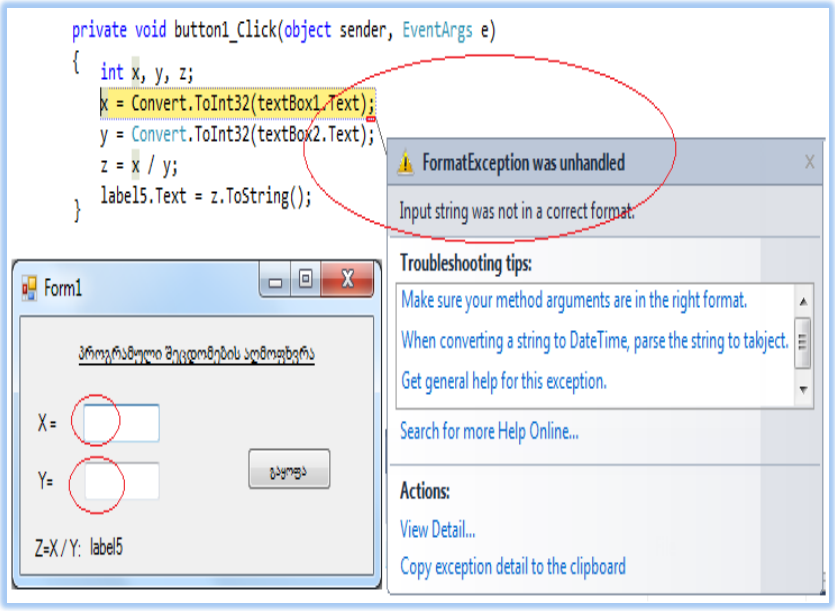
ნახ.7.7



ნახ.7.8

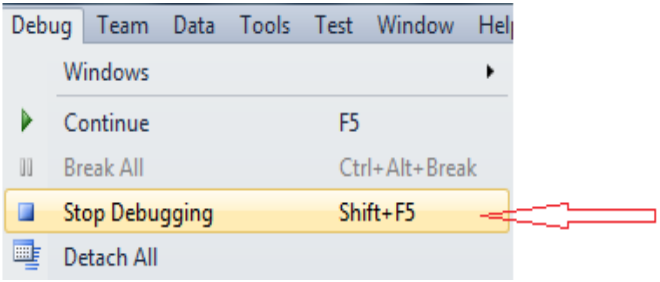
როგორც ვხედავთ, პროგრამა მუშაობს თითქოს ნორმალურად, ასრულებს გაყოფას. ტესტირების პროცესში, რომელიც გულისხმობს კოდის ფუნქციის გამოკვლევას საწყისი მონაცემების სხვადასხვა მნიშვნელობისათვის, ვიღებთ „ნულზე გაყოფის“ შეცდომას (ნახ.7.8).

პროგრამაში საჭიროა ამ სიტუაციის გათვალისწინება (მომხმარებელმა ყოველთვის შეიძლება შეიტანოს შემთხვევით ან „არცოდნის“ გამო 0 !). ანუ თუ იქნება შეტანილი „0“, მაშინ პროგრამამ „გვერდი აუაროს“ (გამორიცხოს, აღმოფხვრას) ასეთი ტიპის შეცდომა და თან შეატყობინოს მომხმარებელს, რომ შეიტანოს კორექტული რიცხვი (0-საგან განსხვავებული) (ნახ.7.9).



ნახ.7.9

რედაქტირების რეჟიმში გადასასვლელად საჭიროა მენიუდან „დებაგერის შეჩერება“ (ნახ.7.10).

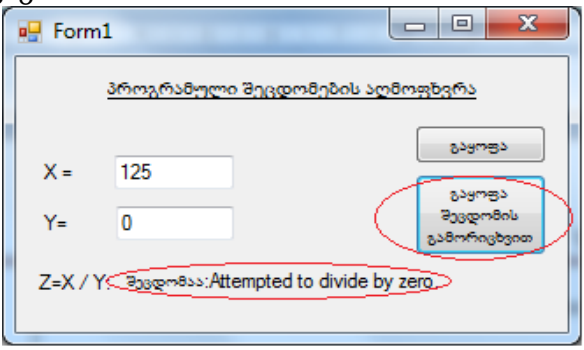


ნახ.7.10

ახლა შესაძლებელია ზემოაღწერილი ტიპის შეცდომებისათვის გამორიცხვის პროცედურის კოდის ფორმირება. მაგალითად, 7.1 ლისტინგზე მოცემულია ასეთი კოდი.

```
//ლისტინგი_7.1 --- Exception -----
private void button2_Click(object sender, EventArgs e)
{
  int x, y, z;
  try
   {
     x = Convert.ToInt32(textBox1.Text);
     y = Convert.ToInt32(textBox2.Text);
     z = x / y;
     label5.Text = z.ToString();
   }
  catch (Exception nul_Div)   // ობიექტი nul_Div
   {
    label5.Text = "შეცდომაა:" + nul_Div.Message;
   }
}
```

7.11 ნახაზზე ნაჩვენებია ამ კოდის მუშაობის შედეგი, რომელიც მოთავსებულია ღილაკზე „გაყოფა შეცდომის გამორიცხვით“.



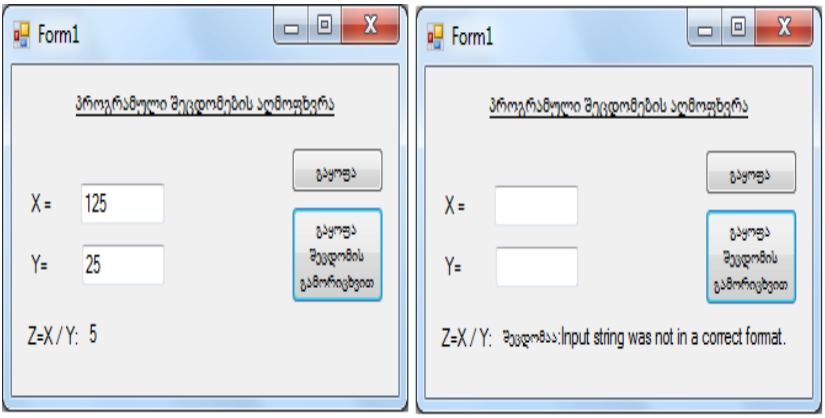
ნახ.7.11

საყურადღებო: გამოყენებულია კონსტრუქცია try { }… catch{ }. საკვანძო სიტყვა try ინიცერებას უკეთებს გამორიცხვის მექანიზმს. ამ წერტილიდან პროგრამა იწყებს {..} ბლოკის შესრულებას. ამ ბლოკის ოპერატორების შესრულებისას თუ გაჩნდა გამორიცხვის (Exception) შემთხვევა, მაშინ მას „დაიჭერს“ catch და შეცვლის გამორიცხვის სიტუაციას თავის {...} ბლოკით.

ჩვენ შემთხვევაში catch ბლოკში მოთავსებულია Exception კლასის ობიექტი (`nul_Div`), რომელიც შეიცავს ინფორმაციას აღმოცენებული შეცდომის შესახებ. Message თვისებით ხდება შეტყობინების გამოტანა ეკრანზე. პროგრამა ბოლომდე სრულდება არაავარიულად.

7.12 ნახაზზე მოცემულია try…catch - გამორიცხვის მექანიზმით შესრულებული პროგრამული კოდის შედეგები: როცა რიცხვები შეტანილია ნორმალურად გაყოფის ოპერაციისათვის (ამ დროს არ ხდება „შეცდომის“

დაფიქსირება try-ში), და მეორე, როცა არასწორადაა შეტანილი საწყისი მონაცემები (აქ იმუშავებს შეცდომების გამორიცხვის ბლოკი).



ნახ.7.12

განსაკუთრებულ შემთხვევათა დამუშავების try…catch მექანიზმი შეიძლება გაფართოვდეს ბიბლიოთეკაში არსებული Exception-კლასის საფუძველზე. აქ იგულისხმება სპეციფიური შეცდომების აღმოჩენის შესაძლებლობა, მაგალითად, ტიპების გარდაქმნისას, ნულზე გაყოფისასა და ა.შ. თუ შეცდომის სახე წინასწარ არაა განსაზღვრული, მაშინ გამოიყენება ზოგადი Exception კლასის ობიექტი.

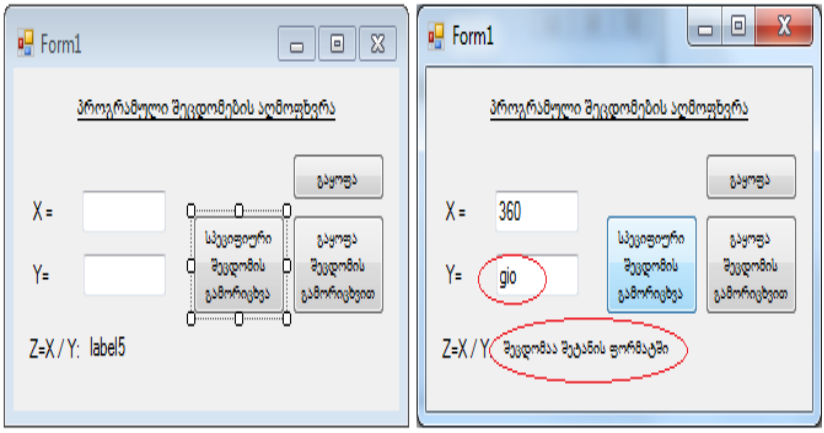
7.2_ლისტინგში მოცემულია ღოლაკის „შეცდომის გამორიცხვა“ შესაბამისი კოდის ფრაგმენტი.

```
// ლისტინგი_7.2 ---- სპეციფიური და ზოგადი შეცდომები --
private void button3_Click(object sender, EventArgs e)
 {
  int x, y, z;
  try
```

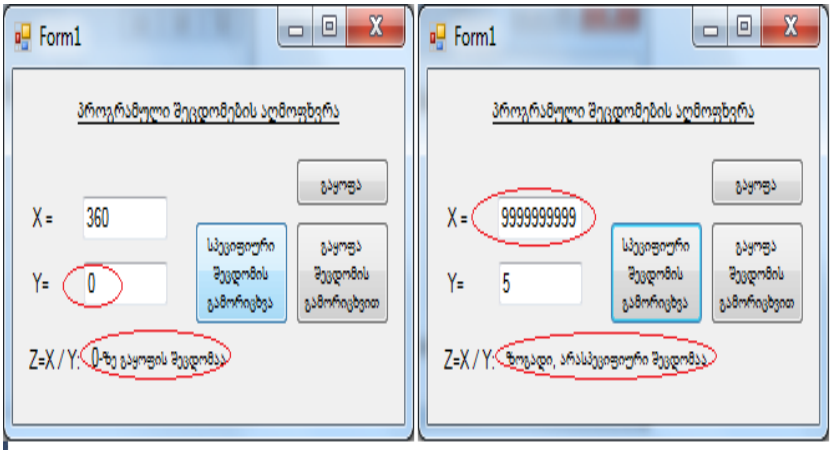
```
  {
    x = Convert.ToInt32(textBox1.Text);
    y = Convert.ToInt32(textBox2.Text);
    z = x / y;
    label5.Text = z.ToString();
  }

 catch(FormatException nul_Div) //ტიპის გარდაქმნის შეცდომა
  {
    label5.Text = "შეცდომაა შეტანის ფორმატში\n" +
                  nul_Div.Message;
  }

 catch(DivideByZeroException nul_Div) // 0-ზე გაყოფის შეცდ.
  {
    label5.Text = "0-ზე გაყოფის შეცდომაა\n" +
                nul_Div.Message; ;
  }

catch (Exception nul_Div)  // ზოგადი შეცდომა
  {
   label5.Text = "ზოგადი, არასპეციფიური შეცდომაა\n"+
                nul_Div.Message;
  }
}
```

ლისტინგში catch{…} ბლოკების მიმდევრობას აქვს მნიშვნელობა (თუ სრულდება პირველი, წყდება პროცესი. თუ არა, გადადის შემდეგზე).

შედეგები ასახულია 7.13-ა,ბ ნახაზებზე.



ნახ.7.13-ა



ნახ.7.13-ბ

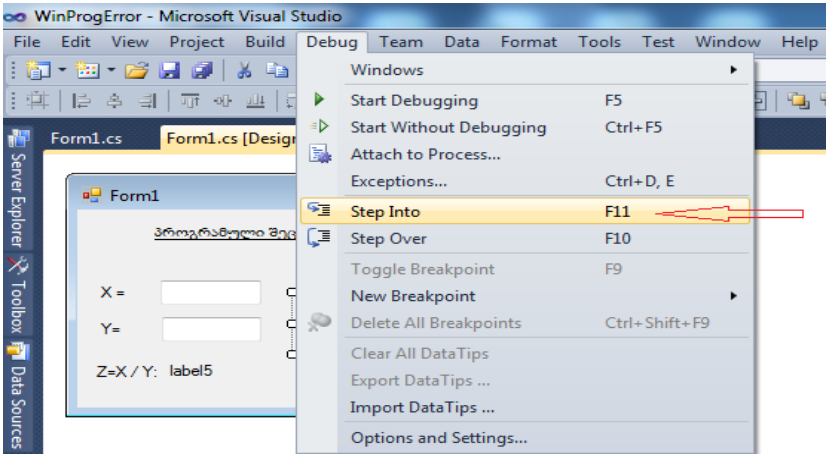
3) ლოგიკური შეცდომები და პროგრამის გამართვა (Debugging) ბიჯური შესრულების რეჟიმში

როგორც აღვნიშნეთ, ლოგიკური შეცდომები მაშინ აღმოჩნდება, როდესაც სინტაქსური და შესრულების

პროცესის შეცდომები აღარაა, მაგრამ სასურველ (დაგეგმილ სავარაუდო) შედეგს პროგრამა არ გვაძლევს.

ეს ნიშნავს, რომ პროგრამის აგების ლოგიკა არასწორია!

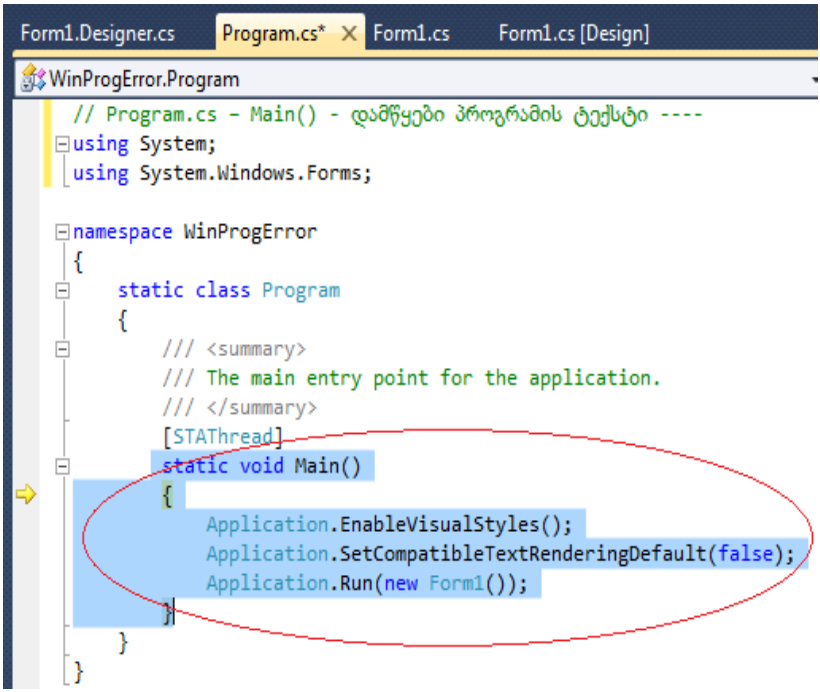
ასეთი შეცდომების აღმოჩენა საკმაოდ რთულია და მოითხოვს ტესტირების პროცესისა და პროგრამის შესრულების მიმდევრობის შედეგების ანალიზს. ვიზუალური C# ენა ფლობს პროგრამის გამართვის (Debugging) კარგ დამხმარე საშუალებებს. განვიხილოთ ისინი ჩვენი WinProgError პროექტის მაგალითზე (ნახ.7.14). გავხსნათ პროექტი, მოვამზადოთ Form1 ფორმა და მენიუს Debug-ში ავირჩიოთ Step Into (ან F11 ღილაკი კლავიატურის ზედა რიგში).



ნახ.7.14

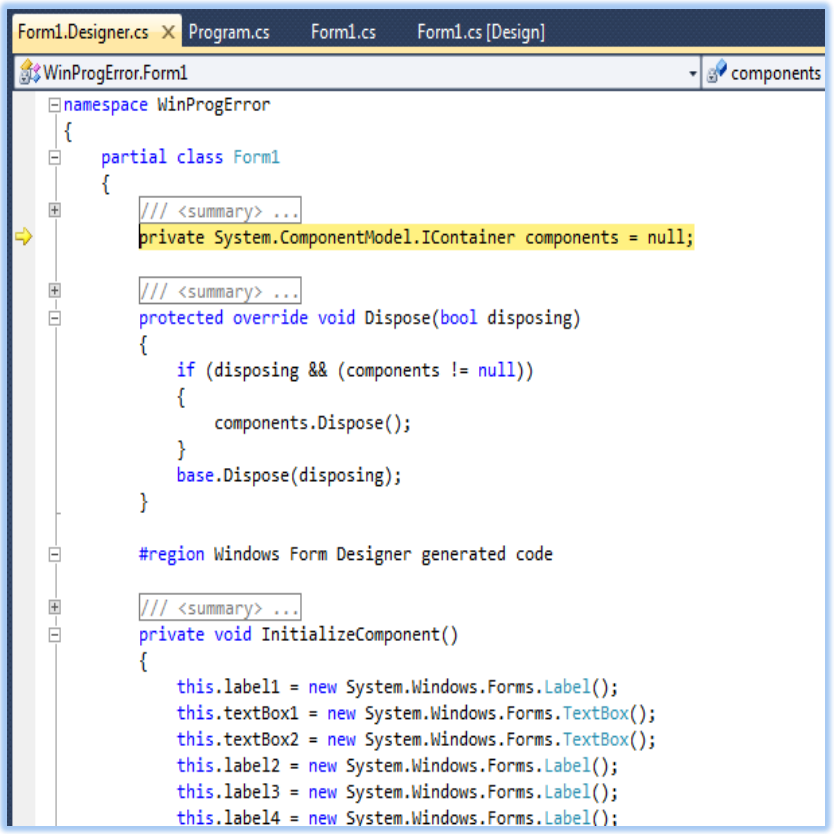
ჩაირთვება პროგრამის გამართვის ბიჯური (Step Into) რეჟიმი. ეკრანზე დიზაინის ფორმა შეიცვლება (იხ. Solution Explorer) Program.cs დამწყები პროგრამის ტექსტით, რომელშიც მოთავსებულია Main() მთავარი ფუნქცია (ნახ.7.15). მარცხნივ ჩანს ყვითელი ისარი, რომელიც F11-ით

ბიჯურად გადაადგილდება იმ სტრიქონზე, რომელიც სრულდება მოცემულ მომენტში.



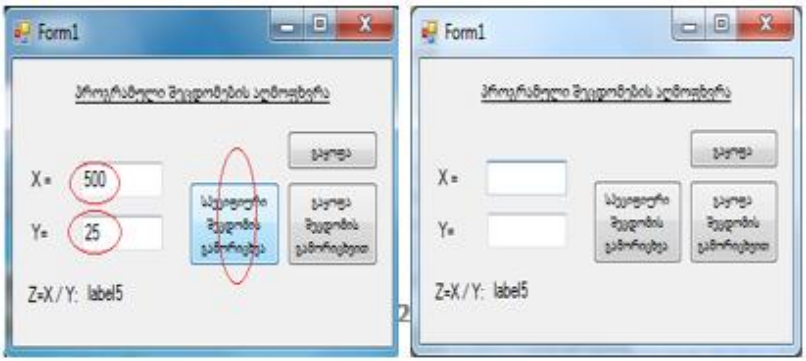
ნახ.7.15

Application.Run (new Form1()); სტრიქონის ამუშავებით ისარი გადადის (იხ. Solution Explorer) Form1.Designer.cs პროგრამაში (ნახ.7.16). შემდეგ InitializeComponent()-ში გაივლის ფორმაზე დალაგებულ ყველა ელემენტს.



ნახ.7.16

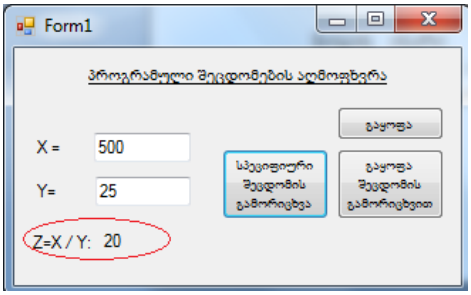
Form1.Designer.cs პროგრამის ბიჯურად გავლის შემდეგ Main()-იდან ამუშავდება Run და ეკრანზე გამოვა Form1 (ნახ.7.17), სადაც უნდა შევიტანოთ X და Y მნიშვნელობები და ავამოქმედოთ ღილაკი „სპეციფიური შეცდომის გამორიცხვა“ (ნახ.7.18). ბიჯის ისარი გადადის Form1.cs პროგრამის ტექსტზე (ნახ.718), სადაც button3_Click მოვლენის შესაბამის try {…} ბლოკში შედის.



ნახ.7.17

```
Form1.Designer.cs   Program.cs   Form1.cs   Form1.cs [Design]
WinProgError.Form1

        private void button3_Click(object sender, EventArgs e)
        {
            int x, y, z;
            try
            {
                x = Convert.ToInt32(textBox1.Text);
                y = Convert.ToInt32(textBox2.Text);
                z = x / y;
                label5.Text = z.ToString();
            }
```

ნახ.7.18



ნახ.7.19

ვინაიდან X და Y-ის შეტანილი მნიშვნელობები დასაშვებია, შედეგში აისახება:

label5.Text=z.ToString()

მნიშვნელობა, ანუ 20 (ნახ.7.19).

თუ ახლა განვიხილავთ Y=0 შემთხვევას, და ავამოქმედებთ იგივე ბუტონს, მაშინ try ბლოკში მომზადდება განსაკუთრებული შემთხვევა, როცა გამყოფი 0-ია.

```
private void button3_Click(object sender, EventArgs e)
{
    int x, y, z;
    try
    {
        x = Convert.ToInt32(textBox1.Text);
        y = Convert.ToInt32(textBox2.Text);
        z = x / y;
        label5.Text = z.ToString();
    }
```

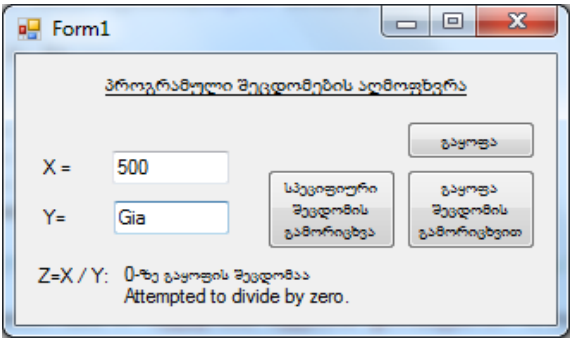
ნახ.7.20

მოქმედება გადაეცემა catch ბლოკს:

```
catch (DivideByZeroException nul_Div)  // 0-ზე გაყოფის შეცდომა
{
    label5.Text = "0-ზე გაყოფის შეცდომაა\n" + nul_Div.Message;
}
```

ნახ.7.21

დავუშვათ, რომ X ან Y არარიცხვითი სიმბოლო ან სტრიქონია, მაგალითად, “G, Gia,…” (ნახ.7.22).



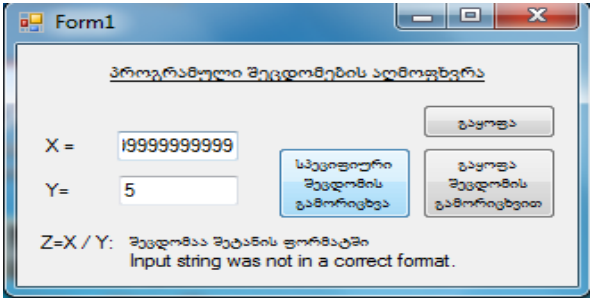
ნახ.7.22

F11-ით გადაადგილების შემდეგ პროგრამის ტექსტის აქტიური ფრაგმენტი იქნება შემდეგი (ნახ.7.23).

```
private void button3_Click(object sender, EventArgs e)
{
    int x, y, z;
    try
    {
        x = Convert.ToInt32(textBox1.Text);
        y = Convert.ToInt32(textBox2.Text);
        z = x / y;
        label5.Text = z.ToString();
    }
    catch (FormatException nul_Div)  // ტიპის გარდაქმნის შეცდომა
    {
        label5.Text = "შეცდომაა შეტანის ფორმატში\n" + nul_Div.Message;
    }
```

ნახ.7.23

მესამე, ზოგადი ანუ არასპეციფიური შემთხვევაა, ამ დროს სისიტემა თვითონ აღმოაჩენს, თუ რა სახის შეცდომასთან გვაქვს საქმე. მაგალითად, თუ X ან Y - ში შევიტანთ „დიდ რიცხვს“ (ნახ.7.24), მაშინ კოდის ფრაგმენტი ასე გამოიყურება (ნახ.7.25).



ნახ.7.24

```
            catch (Exception nul_Div)  // ზოგადი შეცდომა
            {
                label5.Text = "ზოგადი, არასპეციფიური შეცდომაა\n" + nul_Div.Message;
            }
```

ნახ.7.25

პროგრამასთან მუშაობის დასამთავრებლად დავხუროთ Form1. ამ დროს მართვა (ისარი) გადაეცემა Form1.Designers.cs (ნახ.7.26) და ბოლოს პროგრამას Form1.cs, რომელშიც არის Main(), და რომლითაც დაიწყო თავიდან ამ პროგრამის მუშაობა (ნახ.7.27).

Form1.Designer.cs | Program.cs | Form1.cs | Form1.cs [Design]

WinProgError.Form1

```
namespace WinProgError
{
    partial class Form1
    {
        /// <summary> ...
        private System.ComponentModel.IContainer components = null;

        /// <summary> ...
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
```

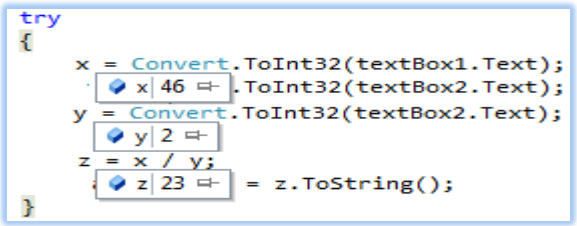
ნახ.7.26

```
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
```

ნახ.7.27

ამით დასრულდება დებაგერის მუშაობის პროცესი.

პროგრამის ანალიზის პროცესში შესაძლებელია ცვლადების მნიშვნელობათა ვიზუალური შემოწმება. ამისათვის მაუსის კურსორი უნდა მივიტანოთ ცვლადთან. 7.28 ნახაზზე ნაჩვენებია პროგრამაში X, Y და Z-ის მნიშვნელობები.

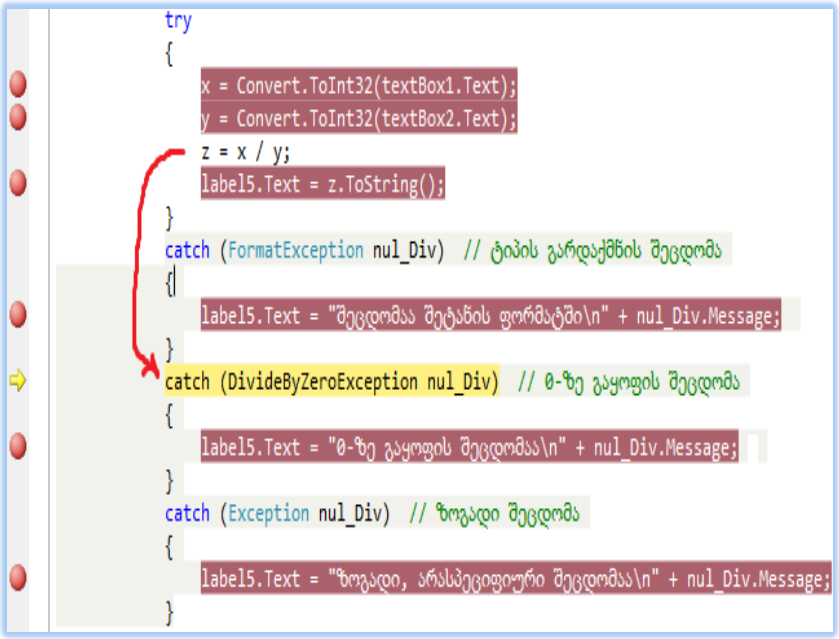


ნახ.7.28

დიდი პროგრამების ანალიზის დროს ბიჯურ რეჟიმში მუშაობა არაეფექტურია, სჭირდება ხანგრძლივი დრო. უფრო მოსახერხებელია ვიზუალური კონტროლის ორგანიზების მეორე ხერხი, რომელიც წყვეტის წერტილების კონცეფციითაა ცნობილი.

წყვეტის წერტილი არის პროგრამის კოდის ის ადგილი (სტრიქონი), სადაც წყდება პროგრამის შესრულება. ამ სტრიქონში მოთავსებული გამოსახულება (ოპერატორი ან მეთოდი) არ შესრულდება და მართვა მომხმარებელს გადაეცემა.

წყვეტის წერტილის შესაქმნელად კოდის საჭირო სტრიქონის გასწვრივ მარცხენა ველში მოვათავსოთ კურსორი და დავაჭიროთ თაგვის მარცხენა კლავიშს (ან F9 კლავიშს). გამოჩნდება შინდისფერი წრე. პროგრამის კოდში შეგვიძლია შევქმნათ წყვეტის რამდენიმე წერტილი (ნახ.7.29).



ნახ.7.29

მენიუდან

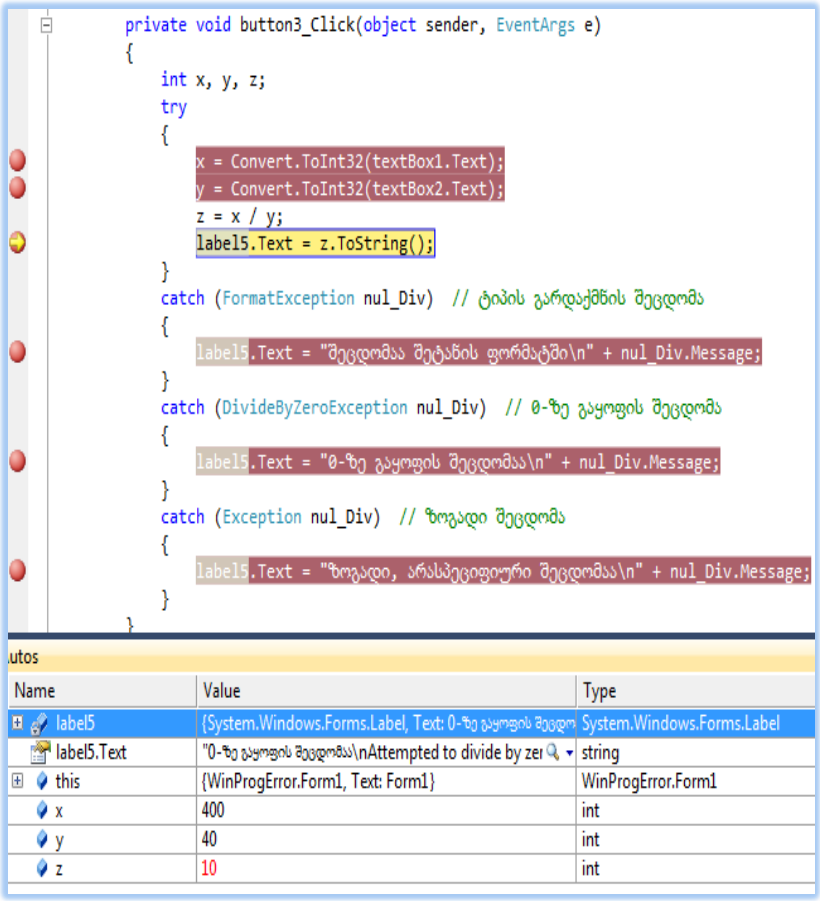
Debug→Windows→Autos

არჩევით რედაქტორის ქვედა ნაწილში გამოიტანება ცვლადების მონიტორინგის ფანჯარა, რომელშიც ერთდროულად ჩანს რამდენიმე ცვლადი მათი აქტუალური მნიშვნელობებით (ნახ.7.30).

მენიუდან

Debug→Windows→Locals

არჩევით მონიტორინგის ფანჯარაში გამოიტანება მხოლოდ ლოკალური ცვლადები და მათი მნიშვნელობები.

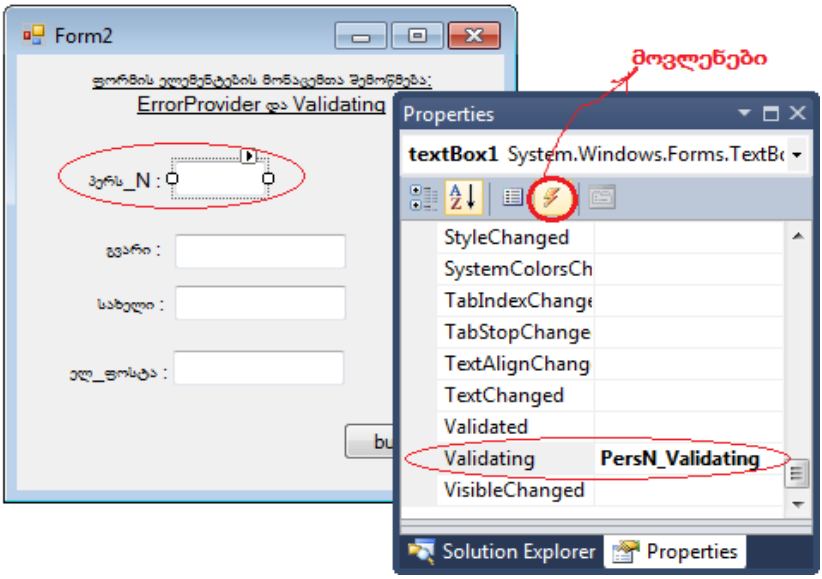


ნახ.7.30

4) შესატან მონაცემთა კონტროლი

ვინდოუს-ფორმის ელემენტების შევსების პროცესში, როცა მომხმარებელს უხდება მათი ხელით შეტანა, შესაძლებელია შეცდომების არსებობა. მაგალითად, ამას ხშირად აქვს ადგილი ტქსტბოქსების შევსებისას.

იმისათვის, რომ შესაძლებელი იყოს შესატან მონაცემთა კონტროლი, საჭიროა ErrorProvider ვიზუალური კომპონენტის გადმოტანა ინსტრუმენტების პანელიდან და საკონტროლო ელემენტების Properties-ში CausesValidation თვისებაში "True" მნიშვნელობის არსებობა (ნახ.7.31).



ნახ.7.31

ტექსტბოქსის შევსების შემდეგ, როცა მომხმარებელი გადადის სხვა ელემენტზე, ხდება ამ ტექსტბოქსში შეტანილი მნიშვნელობის შემოწმება. თუ რა ლოგიკით შემოწმდება ტექსტბოქსში შეტანილი მონაცემი, დამოკიდებულია ჩვენ მიერ განსაზღვრულ მეთოდზე, რომელიც მიებმება მოვლენის დამმუშავებელს (Event Handling).

ამგვარად, ახლა საჭიროა მოვლენის დამმუშავებელის შექმნა. მაგალითად, ვდგებით „პერს_N" ტექსტბოქსზე და

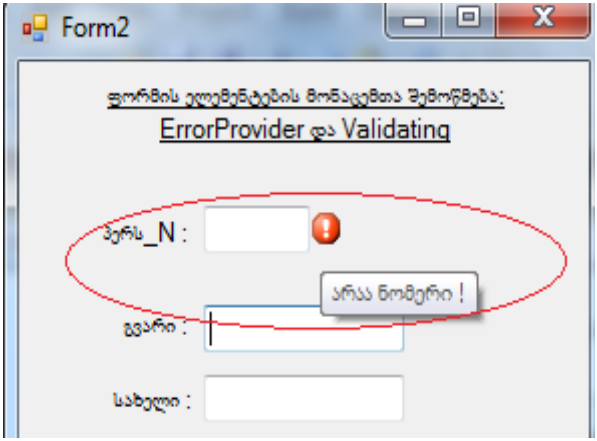
Properties-ში Validating თვისებას ვაძლევთ სახელს: PersN_Validating. დამმუშავებლის მეთოდის კოდი მოცემულია 7_3 ლისტინგში.

```
// ლისტინგი_7.3 ---  მოვლენების დამმუშავებელი ------
private void PersN_Validating(object sender, CancelEventArgs e)
{
  if (textBox1.Text.Length == 0)
    {
    errorProvider1.SetError(textBox1, "არაა ნომერი !");
   }
  else
     errorProvider1.SetError(textBox1, "");
}
```

შედეგი მოცემულია 7.32 ნახაზზე. ველში „პერს_N“ არ ჩაწერეს მონაცემი, ისე გადავიდნენ სხვა ტექსტბოქსზე. ამ დროს მოხდა მოვლენის შემოწმება და შეცდომის აღმოჩენა, რომ ველში არაა შეტანილი მონაცემი (textBox1.Text.Length არის 0).

ჩაირთვება errorProvider1.SetError და „პერს_N“-ის გვერდით გამოიტანს წითელი ფერის ძახილის ნიშანს (გაფრთხილება).

მაუსის კურსორის მიტანისას ამ ველზე ჩნდება ქართული წარწერა „არაა ნომერი“ (SetError-ის მეორე პარამეტრი).

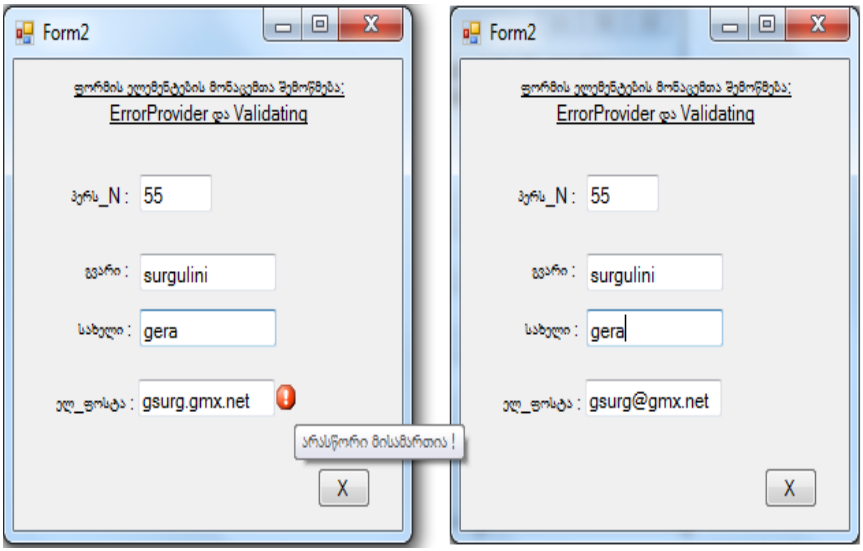


ნახ.7.32

მეოთხე ტქსტბოქსში უნდა ჩაიწეროს ელ_ფოსტის მისამართი. იგი სტრიქონული მონაცემია, რომელიც აუცილებლად უნდა შეიცავდეს '@' და ' . ' სიმბოლოებს. თუ რომელიმე აკლია, მაშინ შეცდომაა და უნდა ამუშავდეს მოვლენის დამმუშავებელი ამ ველისთვის (7.4_ლისტინგი).

```
// ლისტინგი_7.4 --- eMail_Validating მეთოდი --------
 private void eMail_Validating(object sender, CancelEventArgs e)
{
  string email = textBox4.Text;
     // კონტროლის ლოგიკა
   if(email.IndexOf('@')==-1 || email.IndexOf('.')==-1)
   {
    errorProvider1.SetError(textBox4,"არასწორი  მისამართია !");
   }
  else
    errorProvider1.SetError(textBox4, "");
}
```

შედეგი ნაჩვენებია 7.33 ნახაზზე.



ნახ.7.33

7.2. პროგრამული აპლიკაციების ტესტირება

განხილულია პროგრამული კოდების ტესტირების პროცესი Visual Studio.NET ინტეგრირებულ გარემოში.

Unit testing და Coded UI არის Microsoft-ის ტესტირების ინსტრუმენტები, რომლებიც სრულდება Visual Studio.NET-გარემოში.

Unit testing და Coded UI არის Microsoft-ის ტესტირების ინსტრუმენტები, რომლებიც სრულდება Visual Studio.NET-გარემოში.

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 მეთოდისათვის დაიწეროს ტესტი. ეს უზრუნველყოფს კოდის სწრაფად შემოწმებას, ხომ არ მიიყვანა კოდის ბოლო ცვლილებამ პროგრამა რეგრესიამდე ანუ შეცდომების გაჩენამდე პროგრამის უკვე ტესტირებულ ნაწილებში [38].

Coded UI ტესტი კი ავტომატურად იწერს, შესრულებაზე უშვებსა და ამოწმებს ტესტ-ქეისებს.

ასეთი ტესტების წერა შესაძლებელია C# ან Visual Basic-ზე Visual Studio გარემოში. Unit testing ტესტირების ტექნოლოგია განვიხილოთ ვირტუალური ობიექტის, მაგალითად, ფინანსური ობიექტის, ბანკის მაგალითზე.

7.2.1. დასატესტი პროგრამის პროექტის შექმნა

Visual Studio.NET-ში საჭიროა ავირჩიოთ:

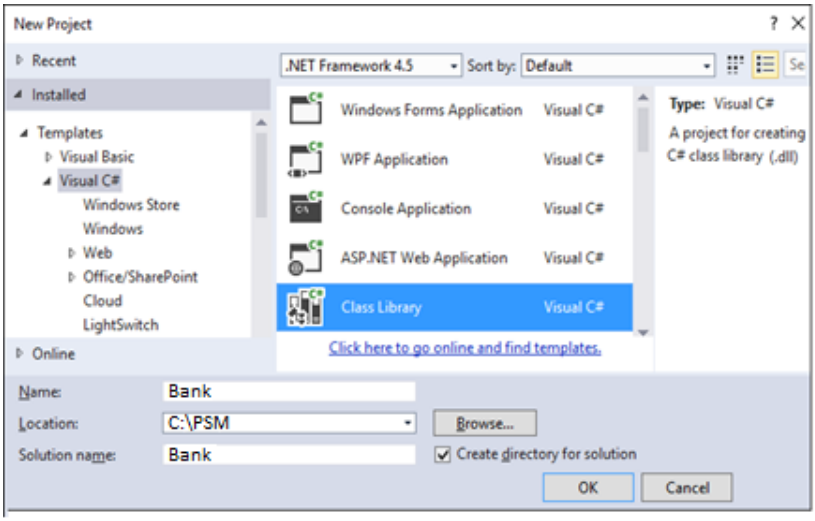
File -> New -> Project.

შედეგად გამოჩნდება დიალოგური ფანჯარა (ნახ.7.34). სადაც ავირჩევთ:

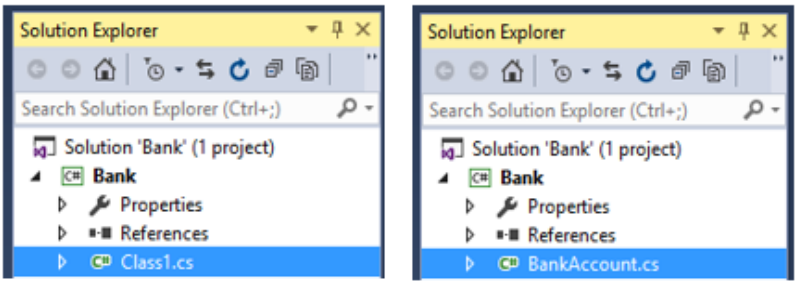
Visual C# => ClassLibrary

პროექტი სახელით Bank.

მიიღება 7.35 ნახაზზე ნაჩვენები Solution Explorer ფანჯარა. აქ Class1.cs სახელი შევცვალოთ BankAccount.cs -ით.



ნახ.7.34. დასატესტი კლასის პროექტის შექმნა



ნახ.7.35. Class1 - > BankAccount

შემდეგ BankAccount.cs-ის ტექსტი რედაქტორის არეში შევცვალოთ ჩვენი დასატესტი პროგრამის კოდით.

ეს საწყისი ტექსტი, მაგალითად, მოცემულია 7.5 ლისტინგში.

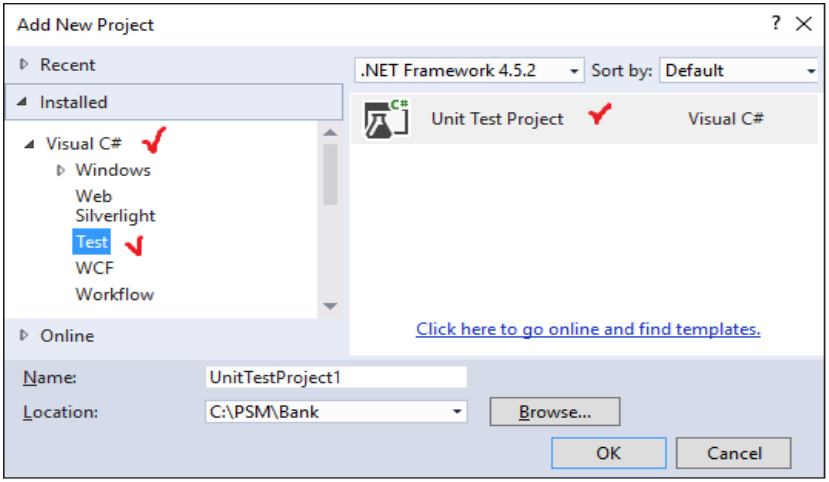
//-- ლისტინგი_7.5 --- BankAccount.cs --------

```
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);
   }
  } }
```

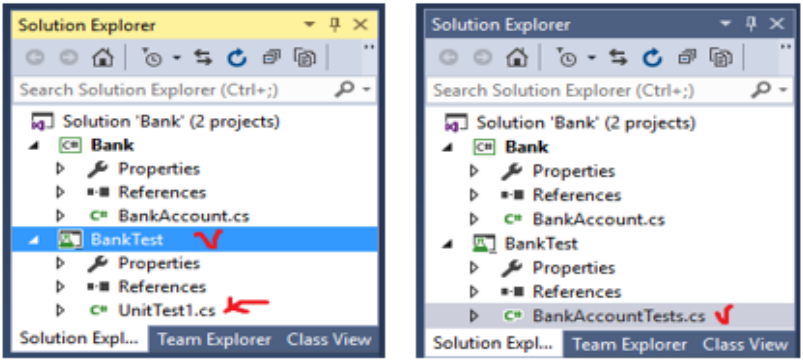
7.2.2. Unit ტესტ-ფაილის პროექტის აგება

დავამატოთ ახალი პროექტი. Visual Studio-ში საჭიროა ავირჩიოთ: **File -> New -> Project.**



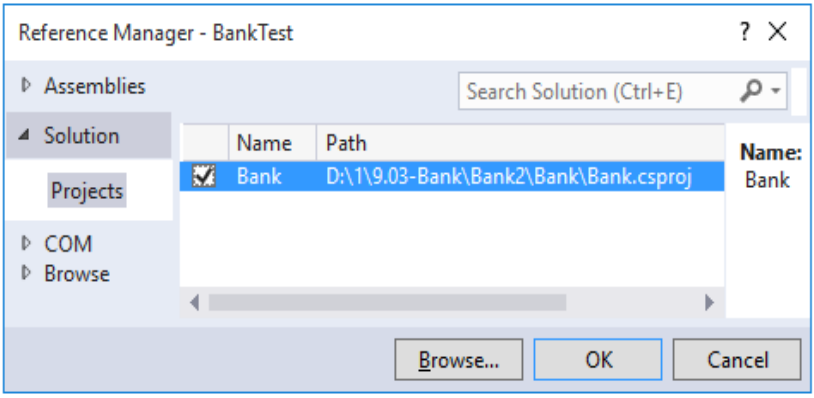
ნახ.7.36. Unit Test პროექტის შექმნა

მივიღებთ 7.37 ნახაზზე ნაჩვენებ სურათს BankTest პროექტით. მარჯვენა სურათზე შეცვლილია UnitTest კლასის სახელი BankAccountTests-ით.



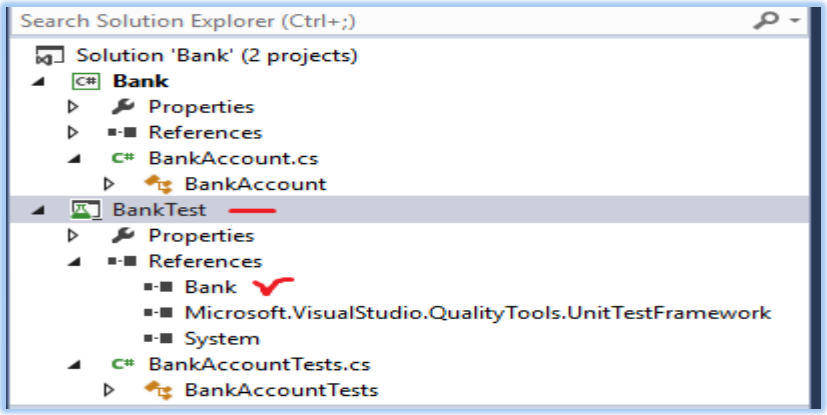
ნახ.7.37

BankTests პროექტში დავამატოთ reference **Bank** solution-იდან. ამისათვის **BankTests**-ზე მაუსის მარჯვენა ღილაკით ავირჩიოთ **Add Reference** და მივიღებთ 7.38 ნახაზზე ნაჩვენებ ფანჯარას. აქ Solution სტრიქონში ვირჩევთ Projects და Bank-ის ჩეკბოქსს მოვნიშნავთ.



ნახ.7.38

მივიღებთ 7.39 ნახაზზე ნაჩვენებ შედეგს.



ნახ.7.39

ჩავამატოთ BankAccountTests პროგრამაში სახელსივრცე Bank-ის პროექტიდან: using Bank;

ამგვარად, BankAccountTests.cs ფაილს ექნება 7.6 ლისტინგზე ნაჩვენები სახე.

//-- ლისტინგი_7.6 ----- BankAccountTests.cs -----

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Bank;

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.

1. 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 მეთოდისათვის, რათა მოხდეს საბოლოო ბალანსის ვერიფიკაცია - არის ის, რასაც ჩვენ ველოდებით.

ტესტ-მეთოდის მოთხოვნები ასეთია:

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

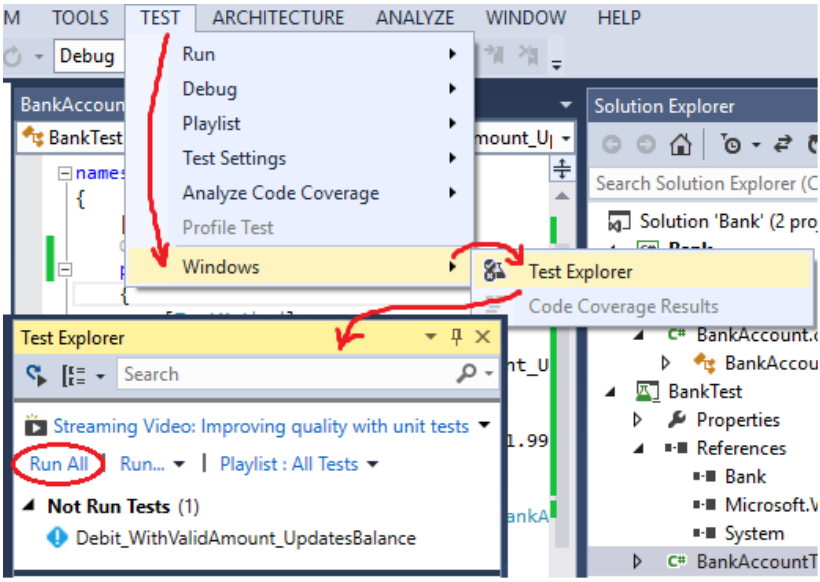
7.2.3. ტესტის კოდის ამუშავება და შედეგის ფორმირება

მთლიანი ტესტის კოდი მოცემულია 7.7 ლისტინგში.

```
//-- ლისტინგი_7.7  ----- BankAccountyTests.cs ----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Bank;
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.7.40).

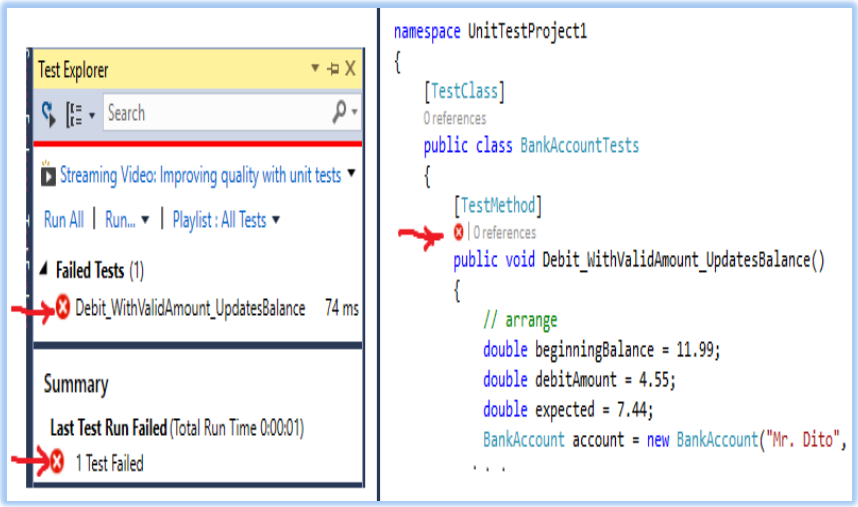


ნახ.7.40

აქ ვირჩევთ Run All - სა და ვიღებთ შედეგს (ნახ.7.41).

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

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



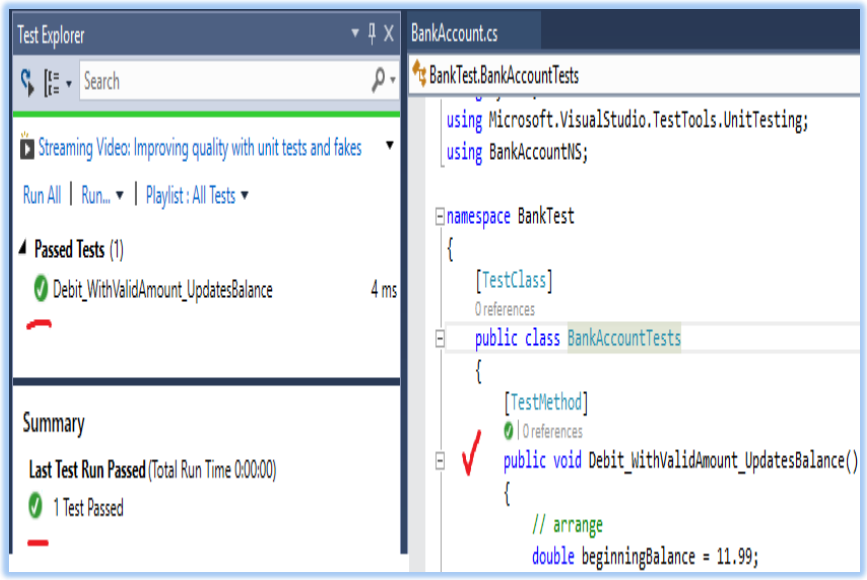
ნახ.7.41

შემდეგი ეტაპი კოდის გასწორება და ხელახალი ტესტირებაა. დასატესტ პროგრამაში შევცვალოთ სტრიქონში „ + „ ნიშანი „ – „ -ით (ნახ.7.42).

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

ნახ.7.42

ტესტის თავიდან ამუშავებით ვიღებთ წარმატებულ შედეგს ანუ მიიღება მწვანე ფერის სიმბოლოები (ნახ.7.43).



ნახ.7.43. სწორი შედეგი

7.3. პროგრამული კოდების ხარისხის შეფასება

განხილულია პროგრამული პროექტების შემუშავების პროცესში აპლიკაციის მოდულების ხარისხის შეფასების მეთოდები.

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 [194,195].

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 [193]:

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

სადაც,

- HV – Halstead Volume, გამოთვლითი სირთულე. მეტრიკის სიდიდე პირდაპირპროპორციულად 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:

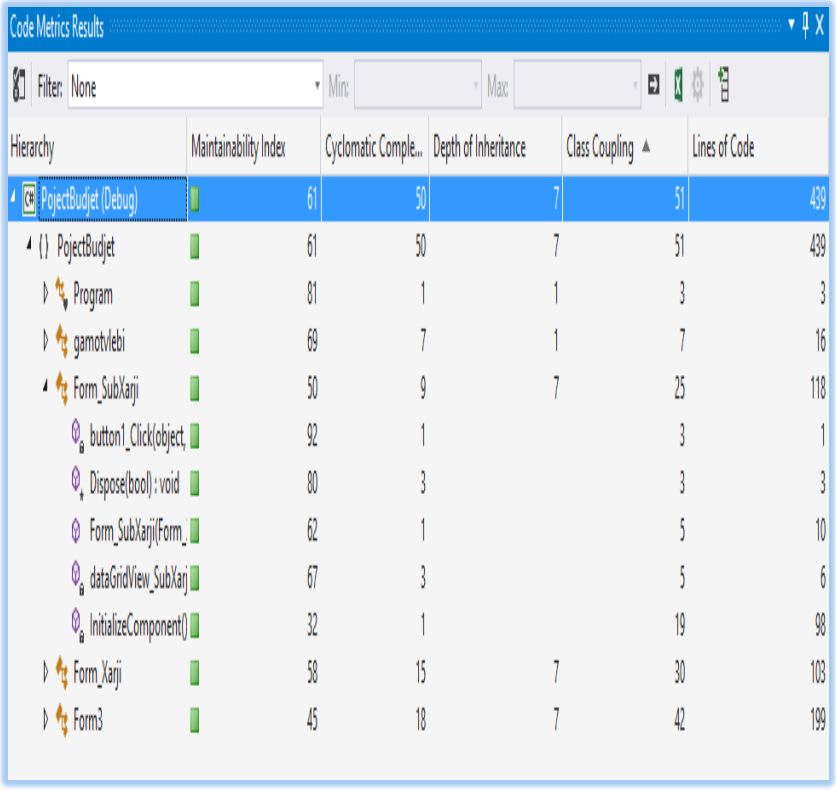
ძირითად მენიუში ღილაკით Analyze – Calculate Code Metrics - for Solution. ასევე, შესაძლებელია ნაწილში Solution Explorer, solution- Calculate Code Metrics გამოძახება მაუსის მარჯვენა ღილაკით.

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.7.44).

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.



ნახ.7.44

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.

ლიტერატურა:

1. Sommerville I. Software Engineering 10th Edition. Copyright Pearson Education, Inc., publishing as Addison-Wesley. 2016

2. Softwarequalität. Internet resource: https://de.wikipedia.org/wiki/Softwarequalität

3. Software quality. Internet resource: https://en.wikipedia.org/wiki/Software_quality

4. Tandard IEC 9126. Internet resource: https://de.wikipedia-.org/wiki/ISO/IEC_9126

5. Goal Question Metric. Internet resource: https://de.wikipedia.org/wiki/Goal_Question_Metric

6. ISO/IEC 15504-5 oder SPICE (Software Process Improvement and Capability Determination) ist ein internationaler Standard. Internet resource: https://de.wikipedia.org/wiki/ISO/IEC_15504

7. Capability Maturity Model Integration (CMMI). Internet resource: https://ru.wikipedia.org/wiki/CMMI

8. Rational Unified Process (RUP). Internet resource: https://de.wikipedia.org/wiki/Rational_Unified_Process

9. V-Model. Internet resource:. https://de.wikipedia.org/wiki/V-Modell

10. სურგულაძე გ. კომპიუტერული პროგრამირების მეთოდები და მეთოდოლოგიები. სტუ. „IT-კონსალტინგის ცენტრი“, თბ., 2019. -202 გვ. https://gtu.ge/book/Surg_ProgMethod_2019.pdf

11. Software Quality Management. Internet resource: https://en.-wikipedia.org/wiki/Software_quality_management

12. Maxim, B.R. Software Quality Management. University of Michigan. Dearborn. 2014. Retrieved 7 December 2017

13. PRINCE2. Internet resource: https://en.wikipedia.org/wiki/-PRINCE2

14. Hinde D. PRINCE2 Study Guide. John Wiley & Sons. ISBN 978-1-119-97097-2. https://books.google.ge/books?id=lVVf-qoDZi UC&pg=-PT16&redir_esc=y#v=onepage&q&f=false

15. Koskela, L., Howell, G. The underlying theory of project management is obsolete, Proceedings of the PMI Research Conf. 2002, 293-302. https://usir.salford.ac.uk/id/eprint/9400/1/2002 _The_underlying_theory_of_project_management_is_obsolete.pdf

16. Continuous Integration (CI). Internet resource: https://en. wikipedia.org/wiki/Continuous_integration

17. Test Driven Development (TDD). Internet resource: https:// en.wikipedia.org/wiki/Test-driven_development

18. სურგულაძე გ., ურუშაძე გ. საინფორმაციო სისტემების მენეჯმენტის საერთაშორისო გამოცდილება (BSI, ITIL, COBIT). სტუ, „ტექნიკური უნივერსიტეტი“. თბ., 2014. -320 გვ. http://gtu.ge/book/-gia_sueguladze/sainfo_sistemebi_BSI_ITIL_COBIT.pdf

19. COBIT. Internet resource: https://en.wikipedia.org/wiki/ COBIT

20. PO8 Manage Quality. Process Description. Plan and Organise A QMS is developed and maintained that includes proven development and acquisition . https://www.isaca.org/resources

21. Software Quality Assurance. Internet resource: https://en.wiki pedia.org/wiki/Software_quality_assurance

22. Softwaremetrik. Internet resource: https://de.wikipedia.org/wi ki/Softwaremetrik

23. Experience curve effects. Internet resource: https://en.wikipe-dia. org/wiki/Experience_curve_effects

24. ErrorQuotient (Fehlerquotient). Internet resource: https://de.-wikipedia.org/wiki/Fehlerquotient

25. Mean Time Between Failures. Internet resource: https://de.wi-kipedia.org/wiki/Mean_Time_Between_Failures

26. Computational complexity. Internet resource: https://en.wikipedia.org/wiki/Computational_complexity

27. Code coverage. Internet resource: https://en.wikipedia.org-/wiki/Code_coverage

28. Milestone Trend Analysis. Internet resource: https://www.project-management-knowhow.com/milestone_trend_analysis.html

29. Lines of Code. Internet resource: https://de.wikipedia.org/wiki/Lines_of_Code

30. Estimation Techniques - Function Points. Internet resourcr: https://www.tutorialspoint.com/estimation_techniques/estimation_techniques_function_points.htm

31. COCOMO. Internet resource: https://en.wikipedia.org/wiki/COCOMO

32. Cyclomatic complexity. Internet resource: https://en.wikipedia.org/wiki/Cyclomatic_complexity

33. Halstead complexity measures. Internet resource: https://en.-wikipedia.org/wiki/Halstead_complexity_measures

34. Measurement of Halstead Metrics with Testwell CMT++ and CMTJava (Complexity Measures Tool.). Internet resource: https://www.verifysoft.com/en_halstead_metrics.html

35. Savoia A. How to Read and Improve the C.R.A.P Index of your code. 2011. Internet resource: https://opnsrce.github.io/how-to-read-and-improve-the-c-r-a-p-index-of-your-code

36. Rana Khudhair Abbas Ahmed. Information Security Metrics and its Relation to Risk Management. American Journal of Data Mining and Knowledge Discovery. Apr.2017. https://www.researchgate.net/-publication/311884003_Overview_of_Security_Metrics

37. გულიტაშვილი მ. Web აპლიკაციების ავტომატური ტესტირება Selenium ტექნოლოგია. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 2(13), 2012. გვ.199-202.

38. სურგულაძე გ., გულიტაშვილი მ., კივილაძე ნ. Web-აპლიკაციების ტესტირება, ვალიდაცია და ვერიფიკაცია. მონოგრ., სტუ. „IT-კონსალტინგის ცენტრი“, თბ., 2015, -205 გვ.

39. Burns D. Selenium 1.0 Testing Tools, Birmingham 2010

40. გულიტაშვილი მ., სურგულაძე გ., ჩერქეზიშვილი გ. პროგრამული უზრუნველყოფის ტესტირების ტიპები და სცენარები. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(14), 2013. გვ.95-99

42. Guru99, new kind of learning experience http://www.guru99.com.

43. 7 Software Testing Principles: Learn with Examples. https://www.guru99.com/software-testing-seven-principles.html

44. Chaudhary S.D. Defect Clustering and Pesticide Paradox. 2015. Internet resource: https://www.pitsolutions.ch/blog/defect-clustering-and-pesticide-paradox/

45. 7 Principles of Software Testing: Defect Clustering and Pareto Principle. 2019. Internet resource: https://www.softwaretestinghelp.com /7-principles-of-software-testing/

46. Software-bug. Internet resource: https://www.techopedia.com/definition/24864/software-bug-

47. Bug-Fix. Internet resource: https://www.techopedia.com/definition/18105/bug-fix

48. Selenium_(software). Internet resource: http://en.wikipedia.-org/wiki/Selenium_(software).

49. სურგულაძე გ., ხვედელიძე ნ., მაისურაძე გ. კორპორაციული პროგრამული აპლიკაციების ხარისხის მართვის, დიზაინის და უსაფრთხოების ზოგიერთი ასპექტები. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N2(29), 2019. გვ.183-189

50. ჩოგოვაძე გ., ფრანგიშვილი ა., სურგულაძე გ., შონია ო. მართვის ავტომატიზებული სისტემები TO მენეჯმენტის საინფორმაციო სისტემები: თანამედროვე მეტამორფოზა. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N2(29), 2019. გვ.7-18

51. სურგულაძე გ., ოხანაშვილი მ., სურგულაძე გ. მარკეტინგის ბიზნესის პროცესების ერთიანი და სიმულაციური მოდელირება. სტუ, „ტექნიკური. უნივ.", თბ., 2009. -170 გვ.,

52. სურგულაძე გ., ქრისტესიაშვილი ხ., სურგულაძე გ. საწარმოთა რესურსების მართვის ბიზნესის პროცესების მოდელირება და კვლევა). სტუ, „ტექნიკური. უნივ.", თბ., 2015. -216 გვ.

53. სურგულაძე გ., გულუა დ. განაწილებული სისტემების ობიექტ-ორიენტირებული მოდელირება უნიფიცირებული პეტრის ქსელებით. სტუ. თბ., 2004.

54. სურგულაძე გ., ბულია ი. კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. მონოგრ., სტუ. თბ., 2012. https://gtu.ge/book/GiaSurg_Book_2012.pdf

55. Котляров В.П., Коликова Т.В. Основы информационных технологий: Основы тестирования программного обеспечения. Интернет-Университет БИНОМ. – М., Информационные Технологий. Лаб. знаний. www.intuit.ru. 2005.

56. Куликов, С.С. Тестирование программного обеспечения. Базовый курс. Минск: „Четыре четверти", 2017. - 312 с.

57. What is Grey Box Testing? Techniques, Example. Guru99. Internet resource: https://www.guru99.com/grey-box-testing.html

58. Unit Testing Tutorial: What is, Types, Tools, Example. Internet resource: https://www.guru99.com/unit-testing-guide.html

59. Integration Testing. Internet resource: http://softwaretesting-fundamentals.com/integration-testing/

60. What is version control. Internet resource (2019): https://www.atlassian.com/git/tutorials/what-is-version-control.

61. Sukhraj R. 10 Best CMS Platforms for Digital Marketing in 2019. 2017. Internet resource: https://www.impactbnd.com/blog/top-10-cms-platforms-for-digital-marketing

62. Deep Dive: SharePoint as CMS and Intranet. Internet resource (2019): https://www.2plus2.com/Resources/Deep-Dive/SharePoint-as-CMS-and-Intranet.aspx

63. Pattigulla M. Measure Your Code Using Code Metrics. 2017. Internet resource: https://www.c-sharpcorner.com/article/measure-your-code-using-code-metrics/

64. Web-hosting service. Internet resource: https://en.wikipedia. org/wiki/Web_hosting_service

65. Domain Name System. Internet resource: https://en.wikipedia.org/wiki/ Domain_Name_System

66. What is DNS. Internet resource: https://cdome.comodo. com/what-is-dns.php

67. What is HTTP. Internet resource: https://www.w3schools.com-/whatis/whatis_http.asp

68. Secure Sockets Layer. Internet resource: https://en.wikipedia.org/wiki/SSL

69. VPN Guide for Newbies 2020. Internet resource: https://www.vpnmentor.com/blog/vpns-101-vpnmentors-vpn-guide-new bies/

70. Virtual Private Network. https://ru.wikipedia.org/wiki/VPN

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

72. Оптимизация программного кода. (2019). Internet resource: https://techrocks.ru/2019 /01/25/code-optimization-tips/

73. Profiler (Programmierung). Internet resource: https://en.wikipedia.org/wiki/Profiling_ (computer_programming)

74. Program optimization or Software optimization. Intrnet resource: https://en.wikipedia.org/wiki/Program_optimization

75. ჩოგოვაძე გ., ფრანგიშვილი ა., სურგულაძე გ. მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. მონოგრ., ISBN 978-9941-20-790-7. სტუ, „ტექნიკური უნივერსიტეტი“, თბ., 2017. -1001 გვ.

76. Оптимизация программного кода. 2019. Ред.techrocks.ru. Internet resource: https://techrocks.ru/2019/01/25/code-optimization-tips/

77. სურგულაძე გ., დოლიძე ს. მომხმარებლის ინტერფეისის დაპროგრამება (AngularJS, ReactJS). სტუ. „IT კონსალტინგ ცენტრი“, თბ., 2019. -106 გვ.|

78. Parallel algorithm. Internet resource: https://en.wikipedia.org /wiki/Parallel_algorithm

79. Reisig W. Elements of Distributed Algorithms. Modeling and Analysis with Petri Nets. Springer Berlin Heidelberg. -324 p. 1998

80. რეისიგი ვ. (გერმანია), სურგულაძე გ., გულუა დ. ვიზუალური ობიექტ-ორიენტირებული დაპროგრამების მეთოდები (BorlandC++Builder, PetriNet). ISBN 99928-943-9-3. სტუ, „ტექნიკ.უნივერსიტეტი", თბ., 2002, -202 გვ.

81. Parallel computing. Internet resource: https://en.wikipedia.org/wiki/Parallel_computing

82. What is concurrent programming? 2020. Internet resource: https://www.educative.io/edpresso/what-is-concurrent-programming.

83. Rozenberg G., Agha G. Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets. 2001. https://www.researchgate.net/publication/242504671_Concurrent_Object-Oriented_Prog¬ram-ming_and_Petri_Nets_Advances_in_Petri_Nets

84. Koirala Sh. Concurrency vs Parallelism 2018. Internet resource: https://www.codeproject.com/Articles/1267¬757/Concurrency-vs-Parallelism

85. Narahari Y. Petri Nets: Overview and Foundations. Dep. of Computer Science and Automation, Indian Institute of Science, Bangalore. 1999. https://www.ias.ac.in/article/fulltext/reso/004/08/0058-0069

86. სურგულაძე გ., გულუა დ., თურქია ე. ბიზნეს-პროცესების მოდელირება პეტრის ქსელებით. სტუ. „ტექნიკური უნივერსიტეტი“, თბ., - 2014. -148 გვ. https://gtu.ge/book/gia_suegula-dze/SurgEka_PetNet1.pdf

87. Lazy evaluation or call-by-need. Internest resource: https://en.wikipedia.org/wiki/Lazy_evaluation

88. Eager evaluation. Internet resource: https://en.wikipedia .org/wiki/Eager_evaluation

89. Horner's Method. Internet resource: https://en.wikipedia.org /wiki/Horner%27s_method

90. Memeti S., Pllana S., Binotto A., Kołodziej J., Brandic I. Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review. Springer. 2018. https://link.springer.com/article/10.1007/s00607-018-0614-9

91. Castañón J. 10 Machine Learning Methods that Every Data Scientist Should Know. Medium. 2019. Internet resource: https://towards-datascience.com/10-machine-learning-methods-that-every-data-scientist -should-know-3cc96e0eeee9

92. Genetic Algorithm. Internet resource: https://en.wikipedia.org/ wiki/Genetic_algorithm

93. Genetic programming. Internet resource: https://en.wikipedia .org/wiki/Genetic_programming

94. Simulated Annealing. Internet resource: https://en.wikipedia .org/wiki/Simulated_annealing

95. Differential evolution. Internet resource: https://en.wikipedia.org/wiki/Differential_evolution

96. Ant colony optimization algorithms. Internet resource: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

97. Bees algorithm. https://en.wikipedia.org/wiki/Bees_algorithm

98. Particle swarm optimization. Internet resource: https://en.wikipedia.org/wiki/Particle_swarm_optimization

99. Zhang Y., Wang Sh., Ji G. A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. 2015. https://www.hindawi.com/journals/mpe/2015/931256/

100. Local search (optimization). Internet resource: https://en.wikipedia.org/wiki/Local_search_(optimization)

101. Tabu search. Internet resource: https://en.wikipedia.org/wiki/Tabu_search

102. Zong Woo Geem. Harmony Search algorithm. Introduction to Harmony Search. https://sites.google.com/a/hydroteq.com/www/

103. Askarzadeh A., Rashedi E. Harmony Search Algorithm. Kerman Graduate University of Advanced Technology, Iran. 2017. doi: 10.4018/978-1-5225-2322-2.ch001 https://www.researchgate.net/publication/314523255_Harmony_Search_Algorithm

104. Applications of artificial intelligence. Internet resource: https://en.wikipedia.org/wiki/Applications_of_artificial_intelligence.

105. Shitut N. Most Popular Regression Algorithms in Machine Learning. Internet resource: https://translate.google.com/

106. რეგრესიული ანალიზი. ლექსიკონი-ცნობარი სოცმეცნიერებებში. 2016. http://dictionary.css.ge/content/analysis-regression

107. Linear regression. Internet resource: https://en.wikipedia.org/wiki/Linear_regression

108. Logistic regression. Internet resource: https://en.wikipedia.org/wiki/Logistic_regression

109. Decision tree learning. Internet resource: https://en.wikipedia.org/wiki/Decision_tree_learning

110. Ray S. Understanding Support Vector Machine(SVM) algorithm from examples (along with code). 2017. Internet resource: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

111. Willems K. Python Machine Learning: Scikit-Learn Tutorial. 2019. Internet rtesource: https://www.datacamp.com/community/tutorials/machine-learning-python

112. Bayesian Methods for Machine Learning. MEDIUM. 2019. Internet resource: https://towardsdatascience.com/bayesian-methods-for-machine-learning/home.

113. Wong P. Predicting vs. Explaining. And Why Data Science Needs More “Half-Bayesians”. https://towardsdatascience.com/predicting-vs-explaining-69b516f90796

114. Brownlee J. How to Implement Bayesian Optimization from Scratch in Python. Internet resource: 2019. https://machinelearningmastery.com/what-is-bayesian-optimization/

115. k-nearest neighbors algorithm. Internet resource: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

116. Machine Learning Made in a Minute. Internet resource: http://accord-framework.net/

117. Kriegel H., Schubert E.,Zimek A. The (black) art of runtime evaluation: Are we comparing algorithms or implementations. Knowledge and Information Systems. 52 (2), Munich. Gernany 2016. 341–378. doi:10.1007/s10115-016-1004-2. ISSN 0219-1377.

118. Classification Algorithms - Random Forest. Internet resource: https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_random_forest.htm

119. Artificial neural network. Internet resource: https://en .wikipedia.org/wiki/Artificial_neural_network

120. Pelk H. Machine Learning, Neural Networks and Algorithms. https://chatbotsmagazine.com/machine-learning-neural-networks-and-algorithms-5c0711eb8f9a

121. Deep learning. Internet resource: https://en.wikipedia.org/wiki/Deep_learning

122. Schulz H., Behnke S. Deep Learning. KI - Künstliche Intelligenz. 26 (4), 2012, pp. 357–363. doi:10.1007/s13218-012-0198-z. ISSN 1610-1987. https://www.researchgate.net/publication/230690795_Deep_Learning_Layer-wise_Learning_of_Feature_Hierarchies

123. Deng L., Yu D. Deep Learning: Methods and Applications. Microsoft Research. USA, 2014. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/DeepLearning-NowPublishing-Vol7-SIG-039.pdf

124. Reactive programming. Internet resource: https://en.wikipedia.org/wiki/Reactive_programming

125. Пацианский М. Основы React (текстовый учебник, 2-е изд.). 2018. https://legacy.gitbook.com/book/maxfarseer/react-cour-se-ru-v2/details

126. React (web framework). Internet resoutce: https://en.wikipedia.org/wiki/React_(web_framework)

127. სურგულაძე გ., პეტრიაშვილი ლ. ვიზუალური დაპროგრამება C# ენის ბაზაზე ინფორმაციული სისტემებისათვის. MsVisual Studio.NET 2019 პლატფორმაზე. სტუ. „IT კონსალტ. ცენტრი“. თბ., 2019., -200 გვ. http://gtu.ge/book/GiaSurg_Csh_IS.pdf

128. სურგულაძე გ., ქრისტესიაშვილი ხ., სურგულაძე გ. საწარმოთა რესურსების მართვის ბიზნესის პროცესების მოდელირება და კვლევა). სტუ, „ტექნიკური. უნივ.", თბ., 2015

129. სურგულაძე გ., გულუა დ. განაწილებული სისტემების ობიექტ-ორიენტირებული მოდელირება უნიფიცირებული პეტრის ქსელებით. სტუ. თბ., 2004

130. Microservices. Internet resoutce: https://de.wikipedia.org/wiki/Microservices

131. სურგულაძე გ., მღებრიშვილი გ. საფინანსო ორგანიზაციის მართვა მიკროსერვისული არქიტექურის გამოყენებით. სტუ-ს შრ.კრებ. „მას" N2(29), 2019. თბ., გვ.117-123

132. Lewis J., Fowler M. Microservices - a definition of this new architectural term. 2014. https://martinfowler.com/articles/microservices.html

133. DevOps. Internet resoutce: https://en.wikipedia.org/wiki /DevOps

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

135. Mala, D.J. Integrating the Internet of Things Into Software Engineering Practices. Advances in Systems Analysis, Software Engineering, and High Performance Computing. IGI Global. p. 16. 2019. ISBN 978-1-5225-7791-1

136. Source Code Control System. Internet resource: https://de.wikipedia.org/wiki/Source_Code_Control_System

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

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

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

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

141. Infrastructure as code. Internet resource: https://en.wikipedia.org/wiki/Infrastructure_as_code

142. Stäbler R. Kotlin for Swift Developers. Medium programming. 2018. https://medium.com/@raphaelstaebler/kotlin-for-swift-developers-40e846fb7813

143. Barrett L. Virtual Reality with React360. 2018. https://hackernoon.com/vir¬tual-reality-with-react-360-ce24b611f0f5

144. სურგულაძე გ., წაწიშვილი დ. ვირტუალური რეალობა და თანამედროვე საინფორმაციო ტექნოლოგიები. ISBN 978-9941-8-0626-1. სტუ, „IT კონსალტინგ ცენტრი“. თბ., 2018. http://gtu.ge/book/Surgul_VirtualReality.pdf

145. ReactJS Tutorial. Internet resource: https://www.tutorialspoint.com/reactjs/

146. Redux ბიბლიოთეკა. Internet resource: https://en.wikipedia.org/wiki/Redux _(Java¬Script_lib¬rary)

147. Hooking. Internet resource: https://en.wikipedia.org/wiki/Hooking

148. Okeh K. How to Get Started With React Hooks: Controlled Forms. 2019. https://medium.freecodecamp.org/how-to-get-started-with-react-hooks-controlled -forms-826c99943b92

149. Kennedy J., Satran M. Hooks. Microsoft. 2018. Internet resource: https://docs. microsoft .com/en-us /windows/desktop/winmsg/hooks

150. Dolodze S. React Hooks: Memoization. 2019. A MEDIUM Co. San Francisco, California, US. https://medium.com/@sdolidze/react-hooks-memoiza¬tion-99a9a91c8853

151. Dolodze S. The Iceberg of React Hooks. 2019. A MEDIUM Co. San Francisco, California, US. https://medium.com/@sdolidze/the-iceberg-of-react-hooks-af0b588f43fb

152. Boilerplate. https://en.wikipedia.org/wiki/Boilerplate_code

153. Design Principles (ReactJS). Copyright © 2020 Facebook Inc. https://reactjs.org/docs/design-principles.html

154. Selenium Automates Browsers. Selenium Official Page. Internet resource: http://docs.seleniumhq.org

155. Hunt D., Newhook P., Kumar T., Selenium Documentation from Selenium Project, 2012

156. გულიტაშვილი მ., სურგულაძე გ. Web-აპლიკაციების ავტომატური ტესტირება. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(12), 2012. გვ.41-44

157. გულიტაშვილი მ. Web აპლიკაციების ავტომატური ტესტირება Selenium ტექნოლოგია. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 2(13), 2012. გვ.199-202

158. Sing Li Automate web application UI testing with Selenium. 2010. http://www.developerfusion.com/article/84484/light-up-your-development-with-selenium-tests

159. Ashfaque A. Software Testing as a Service, 2010.

160. Burns D. Selenium 1.0, Testing Tools. Birmingham. 2010

161. JavaScript. Internet resource: https://en.wikipedia.org/wiki/JavaScript

162. ბოტჰე კ., სურგულაძე გ., დოლიძე თ., შონია ო., სურგულაძე გ. თანამედროვე პროგრამული პლატფორმები და ენები (Unix, Linux, Windows, C++, Java). სტუ, თბილისი, 2003

163. Selenium (software). Internet resource. https://en.wikipedia.org/wiki/Selenium_(software)

164. Бек К. Шаблоны реализации корпоративных приложений. Экстремальное программирование: Пер. с англ. М.: Вильямс, 2008.

165. Stewart S. WebDriver. The 2nd Annual Google Test Automation Conference (GTAC) in our New York office on August 23 and 23. https://www.youtube.com/watch?v=tGu1ud7hk5I

166. სურგულაძე გ., გულიტაშვილი მ., რამიშვილი ა. Web სისტემების ტესტირების ავტომატიზაცია Open Source ტექნოლოგიების გამოყენებით. ინსო–2013, VI საერთაშ.სამეცნ.კონფ. „ინტერნეტი და საზოგადოებია“. ქუთაისი. 2013. გვ. 15-19.

167. Selenium Grid Tutorial: Hub & Node (with Example). Internet resource: https://www.guru99.com/introduction-to-selenium-grid.html

168. Apache Ant (Another Neat Tool). Internet resource: https://en.wikipedia.org/wiki/Apache_Ant

169. Molina D. Grid2. 2019. Internet resource: https://github .com/SeleniumHQ/selenium/wiki/Grid2

170. Get Started With Selenium 3 and Selenium Grid. 2016. Internet resource: https://www.browseemall.com/Blog/index.php/2016/-11/03/get-started-with-selenium-3-and-selenium-grid/

171. Stewart S. Selenium Grid 4. 2019. Internet resource: https://github.com/SeleniumHQ/selenium/wiki/Selenium-Grid-4

172. Renuka Hingmire, What is the difference between Selenium 3 and Selenium 4? 2020. Internet resource: https://www.quora.com/What-is-the-difference-between-Selenium-3-and-Selenium-4

173. TestNG Tutorial: Annotations, Framework, Examples in Selenium. INternet resource: https://www.guru99.com/all-about-testng-and-selenium.html

174. Freeman A. Pro ASP.NET MVC 4. Apress, 2013.

175. Sanderson S. Pro ASP.NET MVC Framework (Expert's Voice in .NET) – Apress, 2009

176. Msdn Library. Internet resource: https://msdn.microsoft.com/ en-us/magazine/dd942838.aspx.

177. Code First vs Database First. Internet resource: https://entity-framework.net/code-first-vs-database-first

178. თურქია ე. ბიზნეს–პროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. სტუ. თბ., 2010.

179. სურგულაძე გ., თურქია ე., ქაჩლიშვილი თ., ფხაკაძე ც. საფინანსო კორპორაციის ბიზნეს-პროცესების მენეჯმენტი ITIL მეთოდოლოგიის საფუძველზე (რეპორტების ავტომატიზაცია). სტუ-ს შრ.კრ. „მას“. 2, 18, თბ., 2014, გვ.51-55.

180. სურგულაძე გ., კივილაძე ნ. Web-აპლიკაციაში რეპორტების ინტეგრაციის მეთოდები. VII-საერთაშ.სამეცნ.-პრაქტ.კონფ. „ინტერნეტი და საზოგადოება“ (INSO 2015). ქუთაისი. 3-5.07. 2014

181. Misner S. Microsoft SQL Server 2012. Reporting Services (Developer Reference).

182. MSDN. microsoft.com. library/reportviewer controls (visual studio). 2013

183. technet.microsoft.com. Library / integrating reporting services into your application. 2013

184. Booch G., Jacobson I., rambaugh J. Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara, 1995

185. Halpin T. ORM-2 Graphical Notation. Neumont Univer., 2004. http://www.orm.net/ pdf/ORM2_TechReport1.pdf

186. Surguladze G., Turkia E., Topuria N., Lominadze T., Giutashvili M. Automation of Business-Processes of an Election System. IV-Intern.Conf. “Problems of Cybernetics and Informatics“ (PCI’ 2012). Baku, Azerbaijan, 2012. pp. 16-19

187. ვედეკინდი ჰ., სურგულაძე გ., თოფურია ნ. განაწილებული ოფის-სისტემების მონაცემთა ბაზების დაპროექტება და რეალიზაცია UML-ტექნოლოგიით. მონოგრ., სტუ, თბ., 2005

188. სურგულაძე გ., თოფურია ნ., ბაკურია კ., ლომიძე მ. საინფორმაციო სისტემის დაპროექტება ობიექტ-როლური მოდელირების და სერვის-ორიენტირებული არქიტექტურის ბაზაზე. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(17), თბილისი, 2014, გვ. 32-44

189. Collins M.J. Beginning WF: Windows Workflow in .NET 3.0. ISBN-978-1-4302-2485-3 Copyright © 2010. USA. http://www.ebooks-it.net/ebook/beginning-wf

190. Уотсон К.,Нейгел К., Педерсен Я., ХаммерР., Джон Д., Скиннер М., Уайт Э. Visual C# 2008: базовый курс. : Пер. с англ. - М. : ООО "И.Д. Вильяме", 2009

191. Petzold Ch. Applications=Code+Markup. A Guide to the MicroSoft Windows Presentation Foundation. St-Petersburg. 2008

192. Anurag S. WCF: From a Beginner's perspective & a Tutorial. V, 4 Apr 2013.

193. Carnegie Mellon Software Engineering Institute. Internet resource: http://www.sei.cmu.edu/

194. Software quality assurance. Internet resource: https://en. wikipedia.org/wiki/-Software_¬quality_assurance

გადაეცა წარმოებას 15.12.2019 წ. ხელმოწერილია დასაბეჭდად 16.01.2020 წ. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 21,5. ტირაჟი 100 ეგზ.

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

ISBN 978-9941-8-0629-2