UDC 62.503.5
SCOPUS CODE 1712
https://doi.org/10.36073/1512-0996-2019-4-56-65

პროგრამული უზრუნველყოფის ტესტირების ძირითადი მეთოდები

რუსუდან ქუთათელაძე ბიზნესის ადმინისტრირების დეპარტამენტი, საქართველოს ტექნიკური უნივერსიტეტი, საქართველო, 0160, თბილისი, მ. კოსტავას 77
E-mail: r.kutateladze@gtu.ge

ანა კობიაშვილი ეკონომიკური ინფორმატიკის დეპარტამენტი, საქართველოს ტექნიკური უნივერსიტეტი, საქართველო, 0160, თბილისი, მ. კოსტავას 77
E-mail: anakobia@hotmail.com

თამაზ ვეკუა ეკონომიკური ინფორმატიკის დეპარტამენტი, საქართველოს ტექნიკური უნივერსიტეტი, საქართველო, 0160, თბილისი, მ. კოსტავას 77
E-mail: tazo.vekua@gmail.com

რეცენზენტები:

რ. სამხარაძე, სტუ-ის ინფორმატიკისა და მართვის სისტემების ფაკულტეტის პროფესორი
E-mail: rsamkharadze@mail.ru

მ. კიკნაძე, სტუ-ის ინფორმატიკისა და მართვის სისტემების ფაკულტეტის პროფესორი
E-mail: m.kiknadze@gtu.ge

ანოტაცია. პროგრამული უზრუნველყოფის გამართული მუშაობა ძალიან მნიშვნელოვანია. სხვადასხვა აპლიკაციის ფუნქციონირებისას გაჩენილმა შეცდომებმა და უწესივრობებმა შესაძლებელია გამოიწვიოს სერიოზული ფინანსური დანაკარგები. ამიტომ უაღრესად აქტუალურია პროგრამული უზრუნველყოფის ხარისხიანი ტესტირება. აღწერილია პროგრამული უზრუნველყოფის ტესტირების დოკუმენტაცია, მისი ძირითადი კომპონენტები, მოყვანილია პროგრამული უზრუნველყოფის შექმნისა და მასზე მუშაობის ეტაპების აღწერა, განხილულია ტესტირების ორი ძირითადი ტიპი, კერძოდ, მანუალური და ავტომატიზებული ტესტირება. მოყვანილია მაგალითი, რომელიც შესრულებულია დაპროგრამების ენაზე JAVA, JUNIT ფრეიმვორკის გამოყენებით. განხილულია JUNIT ფრეიმვორკის საშუალებით ტესტირებისთვის ყველაზე საჭირო კლასის Assert-ის მეთოდები შესაბამის პრაქტიკულ მაგალითთან ერთად. ნაშრომში ასევე მოკლედ არის გადმოცემული პროგრამული უზრუნველყოფის ტესტირების სხვადასხვა მეთოდი და ზოგადად ტესტირების სასიცოცხლო ციკლი.

საკვანძო სიტყვები: ალგორითმი; პროგრამული უზრუნველყოფის ტესტირება; ტესტერი; ფრეიმვორკი.

შესავალი

პროგრამული უზრუნველყოფის ტესტირება არის კვლევა, რომელიც ტარდება იმისათვის, რომ დაინტერესებულ მხარეს ჰქონდეს ინფორმაცია პროგრამული პროდუქტის ხარისხზე. დღესდღეობით ყველა საჯარო და კერძო კომპანიაში გამოიყენება სხვადასხვა პროგრამული აპლიკაცია; სწორედ მათი სწორად და გამართულად მუშაობისთვის არის საჭირო პროგრამული უზრუნველყოფის ტესტირება. აპლიკაციის ტესტირების მიზანია იპოვოს მასში რაც შეიძლება მეტი შეცდომა, რათა ამ შეცდომების აღმოფხვრის შემდეგ პროგრამამ გამართულად იმუშაოს.

ახალი პროგრამული უზრუნველყოფა როდესაც იქმნება, ტესტირება უნდა იყოს მისი სასიცოცხლო ციკლის განუყოფელი ნაწილი, რათა საბოლოო ჯამში მივიღოთ ზუსტად ისეთი პროდუქტი, რომელიც შეესაბამება დამკვეთის მოთხოვნებს და მაქსიმალურად უშეცდომო იქნება.

ტესტირების შედეგად აღმოჩენილი შეცდომები და სხვა დეფექტები ანგარიშის სახით დეველოპერების გუნდს ეცნობება და მათი აღმოფხვრა მოხდება. მას შემდეგ, რაც შეცდომა გასწორდება, განმეორებითი ტესტირება მოხდება. პროგრამაში შეცდომის გასწორების შემდეგ უფრო სერიოზული შეცდომა შეიძლება გაჩნდეს ან/და საერთოდ ახალი შეცდომა წარმოიშვას. ასევე პროგრამული უზრუნველყოფის ტესტირების შედეგად ხშირად შესაძლებელია მივიღოთ ობიექტური და დამოუკიდებელი ინფორმაცია პროგრამის ხარისხისა და რისკების შესახებ.

ზოგჯერ ტესტირების შემდეგ შესაძლოა ვერ მოხდეს იმის იდენტიფიცირება, რომ პროდუქტი სწორად მუშაობს ყველა პირობებში, მაგრამ შესაძლოა დადგინდეს, რომ ის გარკვეულ პირობებში არ მუშაობს.

პროგრამაში ტესტირების ადრე დაწყება ამცირებს ღირებულებას, პროგრამის გადაკეთებისათვის საჭირო დროსა და შეცდომების რაოდენობას. შესაბამისად, კლიენტი იღებს მაღალი ხარისხის პროგრამულ პროდუქტს.

ძირითადი ნაწილი

ტესტირების სასიცოცხლო ციკლის ცნება საკმაოდ განსხვავებულია; სხვადასხვა კომპანიაში შეიძლება არსებობდეს განსხვავებული სასიცოცხლო ციკლი, თუმცა მისი გარკვეული სტანდარტული სქემა არსებობს. ეს სქემა შედგება შემდეგი ბიჯებისაგან:

მოთხოვნების ანალიზი: ტესტირების პროცესი მოთხოვნების გაანალიზებით უნდა დაიწყოს. ამ დროს ტესტერებმა უნდა გადაწყვიტონ, პროგრამის რომელი ასპექტები შემოწმდეს და რა პარამეტრებით იმუშაოს ამ ტესტებმა.

ტესტირების დაგეგმვა: საჭიროა შემუშავდეს ტესტირების სტრატეგია, შეიქმნას ტესტირების გეგმა. ეს პროცესი განსაკუთრებით მნიშვნელოვანია მაშინ, როცა ბევრი მოქმედებაა შესასრულებელი.

ტესტ-დეველოპმენტი: იქმნება ტესტ-პროცედურების, ტესტ-სცენარების, ტესტ-ქეისებისა და ტესტ-სკრიპტების ნაკრები, რომლებიც გამოიყენება სატესტო პროგრამაში.

ტესტების შესრულება: ტესტერები შესრულებაზე გაუშვებენ ტესტებს და რაიმე შენიშვნის შემთხვევაში აცნობებენ დეველოპერებს.

ტესტირების ანგარიში: ტესტირების დასრულების შემდეგ, ტესტერები ადგენენ ანგარიშს იმის

შესახებ, მზად არის თუ არა პროგრამა სამუშაოდ.

ტესტირების შედეგის ანალიზი: დეველოპერები ბიზნესის წარმომადგენლებთან ერთად განიხილავენ, თუ რა უნდა გამოსწორდეს ან რომელ შეცდომას არ უნდა მიექცეს ყურადღება.

რეგრესიული ტესტირება: ჩვეულებრივ პატარა პროგრამა უნდა გვქონდეს, რომელიც უზრუნველყოფს იმას, რომ პროგრამაში ახალი მოდულის დამატებისას პროგრამის ყველა დანარჩენი ნაწილი უშეცდომოდ მუშაობდეს.

ტესტის დახურვა: მას შემდეგ, რაც მივიღებთ კრიტერიუმების შესაბამის მდგომარეობას, უნდა განისაზღვროს, თუ რა ღონისძიებებს უნდა მივმართოთ, რომ შევინარჩუნოთ ძირეული შედეგები. პროექტთან დაკავშირებული მიღებული გაკვეთილები, შედეგები, ჟურნალები, დოკუმენტები დაარქივდება და ინახება, ხოლო მომავალში გამოიყენება, როგორც საცნობარო მასალა სამომავლო პროექტებისათვის.

პროგრამულ ტესტირებაზე თავიანთი შესაძლებლობების შესაბამისად შემდეგი სპეციალისტები მუშაობენ: პროგრამების ტესტერები, პროგრამების დეველოპერები, პროექტის მენეჯერები და მომხმარებლები.

პროგრამული უზრუნველყოფის ტესტირებას აქვს სამი ძირითადი მიზანი: უწესივრობების პოვნა, გადამოწმება, ვალიდაცია.

ტესტირების დოკუმენტაცია მოიცავს დოკუმენტაციის არტიფაქტებს, რომლებიც უნდა იყოს შედგენილი ტესტირების დაწყებამდე ან ტესტირების მიმდინარეობისას. ტესტირების დოკუმენტაციაში შედის:

- ტესტირების გეგმა (Test Plan);
- ტესტირების სცენარი (Test Scenario);
- ტესტების ნაკრები (Test Case);
- ტრასირების მატრიცა (Traceability Matrix).

ტესტირების გეგმა აღწერს სტრატეგიას, რესურსებს, ტესტირების გარემოს, რომელიც გამოყენებული იქნება პროგრამული უზრუნველყოფის ტესტირების დროს. ტესტირების გეგმის შედგენა ხდება ხარისხის უზრუნველყოფის სამსახურის მიერ. ტესტირების გეგმა მოიცავს: დოკუმენტის შესავალ ნაწილს, აპლიკაციის ტესტირების პროცესის მსვლელობის სავარაუდო ვერსიას, ტესტ-სცენარების სიას, გასატესტი ფუნქციების სიას, ტესტირებისათვის გამოყენებული მიდგომის მითითებას, გასატესტი ელემენტების სიას, პროგრამის ტესტირებისათვის საჭირო რესურსების ნუსხას, დავალებების ცხრილს და იმ რისკების ჩამონათვალს, რომლებმაც შეიძლება თავი იჩინოს ტესტირების დროს.

მოცემული ტესტირების კონკრეტულ მაგალითზე და კონკრეტული ტესტ-ქეისის შემთხვევაში, ტესტერს უნდა ჰქონდეს კრიტერიუმების კომპლექტი, რათა ტესტირების შესრულების შემდეგ გადაწყვიტოს, ტესტი წარმატებით იქნა შესრულებული თუ არა. ტესტირების გეგმის ამ ნაწილში უნდა იყოს შეტანილი მსგავსი კრიტერიუმების აღწერა. თუ ტესტი არ იქნა წარმატებით შესრულებული, ტესტი მარცხს განიცდის. ეს ხდება მაშინ, როდესაც პროგრამის შედეგი არ ემთხვევა იმას, რაც განსაზღვრულ პირობებში ტესტირების შედეგად იყო მოსალოდნელი. რეალურ შედეგსა და მოსალოდნელ შედეგს შორის სხვაობა შეიძლება განსაზღვრული იყოს ერთი ან რამდენიმე დეფექტით. დეფექტის გავლენა შეიძლება მინიმალურად აისახოს პროგრამის მომხმარებელზე ან დეფექტმა შეიძლება

მოახდინოს სისტემის შეჩერება. შესაფასებლად გამოიყენება სკალები, მაგალითად, სკალაზე შეფასებით 1-დან – 4-მდე, დეფექტმა 1-ქულიანი შეფასებით შეიძლება გამოიწვიოს სისტემის შეჩერება, ხოლო 4 ქულით შეფასებულმა დეფექტმა შეიძლება მინიმალური გავლენა მოახდინოს პროგრამის მომხმარებელზე.

მაგალითისათვის წარმოვიდგინოთ, რომ პროგრამულად ხდება ანგარიშის გენერირება Word-ის დოკუმენტში და მისი ამობეჭდვა; პროგრამის ტესტირების დროს გამოჩნდა, რომ ანგარიში და მასში შემავალი ტექსტი კორექტულია, მაგრამ დოკუმენტში აბზაცები არ არის გამოყოფილი. ასეთი შეცდომა შეიძლება შეფასდეს ზემოთ მოყვანილი სკალების მაგალითის მიხედვით 3-4 ქულით. 1-ქულიანი შეფასება ამ შემთხვევაში შეიძლება მიიღოს ისეთი სახის დეფექტმა, როგორიცაა, მაგალითად, მომხმარებლის მიერ აღნიშნული ანგარიშის დასაგენერირებელ ღილაკზე ხელის დაჭერის შეცდომა, როცა ვეღარ ხერხდება ოპერაციის გაგრძელება. ასეთ დროს სისტემა გამოუსადეგარი ხდება. აღნიშნული შეფასების სისტემა ძალიან ეხმარება დეველოპერს დეფექტებისთვის პრიორიტეტების მინიჭებაში და მათი თანამიმდევრულად შესრულების დაგეგმვაში.

ტესტირების დამგეგმავს შეუძლია სკალირების ხერხი გამოიყენოს, რათა დაადგინოს პროგრამის შეცდომის მისაღები დონე. ტესტირების დროს დეფექტების რაოდენობა და მათი სირთულეები თუ გადააჭარბებს მისაღებ დონეს, შეიძლება ითქვას, რომ პროგრამული უზრუნველყოფის ტესტირებამ წარმატებით ვერ ჩაიარა.

ტესტირების სცენარი არის პირობა, რომელიც მიუთითებს, თუ რომელი ნაწილი უნდა იყოს გატესტილი აპლიკაციაში. მისი საბოლოო მიზანია პროცესის სრული ტესტირების უზრუნველყოფა. ტესტირების სცენარი და ტესტების ნაკრები გამოიყენება როგორც სინონიმები, მაგრამ სხვაობა მათ შორის არის ის, რომ ტესტირების სცენარი მოიცავს რამდენიმე ბიჯს, ხოლო ტესტირების ნაკრები – კონკრეტულად ერთს. ერთი სცენარიდან შეიძლება გამოდიოდეს ტესტების რამდენიმე ნაკრები.

ტესტების ნაკრები შედგება რამდენიმე ბიჯისაგან – პირობებისაგან, რომლებიც გამოყენებულია ტესტირების დავალების შესრულებისას. მისი მთავარი მიზანია დადგინდეს, პროგრამული პროდუქტი ფუნქციონირებს სწორად თუ არასწორად. არსებობს ნაკრების რამდენიმე ტიპი: ფუნქციური, ნეგატიური, ლოგიკური და ა. შ. მთავარი კომპონენტები, რომლებსაც შეიცავს ყველა ნაკრები, არის: ნაკრების იდენტიფიკატორი, პროდუქტის მოდული, პროდუქტის ვერსია, ცვლილების ისტორია, მიზანი, ვარაუდები, წინაპირობები, ბიჯები, სავარაუდო შედეგი, არსებული შედეგი, შემდგომი პირობები.

ტრასირების მატრიცა არის ცხრილი, რომელიც გამოიყენება პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის თვალყურის სადევნებლად.

პროგრამული უზრუნველყოფის ტესტირებაში არსებობს ტრასირების მატრიცების ტიპები: პირდაპირი ტრასირება, უკუმიმართულების ტრასირება, ორმაგი მიმართულების მქონე ტრასირება (პირდაპირი და უკუმიმართულებით).

არსებობს ტესტირების ორი ძირითადი ტიპი: მანუალური და ავტომატური.

მანუალური ტესტირება არის ტესტირების ტიპი, რომლის დროსაც ხდება პროგრამის ხელით ტესტირება და არ გამოიყენება ავტომატიზებული საშუა-

ლებები ან სკრიპტები. მანუალური ტესტირება გამოიყენება იმ შემთხვევაში, თუ პროგრამის ინტერფეისი საკმაოდ ხშირად იცვლება ან თუ პროგრამა პატარა მოცულობისაა, რადგან ასეთ შემთხვევაში ავტომატური ტესტის შექმნას უფრო დიდი დრო დასჭირდება, ვიდრე მანუალურ ტესტირებას.

ავტომატური ტესტირება არის ტიპი, რომელშიც ტესტერი წერს სკრიპტს ან იყენებს სხვა გარე პროგრამას მოცემული პროგრამის ტესტირებისათვის. ავტომატიზებული ტესტირება გამოიყენება იმისათვის, რომ ის ტესტ-სცენარები ბევრად უფრო სწრაფად გაეშვას, რომლებიც შესრულებული იყო მანუალური ტესტირების დროს. ავტომატიზებული ტესტირება ზოგავს დროს მანუალურ ტესტირებასთან შედარებით ანუ ასეთი ტესტირების დადებითი მხარეა სისწრაფე.

არსებობს ტესტირების რამდენიმე მეთოდი: თეთრი ყუთის, შავი ყუთის, რუხი ყუთისა და ვიზუალური.

შავი ყუთის ტესტირების დროს ტესტირება ხდება მომხმარებლის ინტერფეისის დონეზე. ტესტერს არ აქვს წვდომა შიგა ლოგიკაზე და კოდის არქიტექტურაზე. მან არ იცის, ესა თუ ის შედეგი რა ოპერაციით შესრულების შედეგად მიიღება კოდის დონეზე.

თეთრი ყუთის ტესტირება არის შიგა ლოგიკისა და კოდის სტრუქტურის დეტალური გამოკვლევა. იმისათვის, რომ ტესტერმა შეასრულოს აღნიშნული ტიპის ტესტირება, მან უნდა იცოდეს კოდის მუშაობის რეჟიმი, მან აუცილებლად უნდა ჩაიხედოს პროგრამის კოდში და ნახოს, ესა თუ ის შეცდომა კონკრეტულად კოდის რომელ ნაწილში ხდება. ტესტერი ირჩევს მეთოდში შემავალ მნიშვნელობებს და ამით განსაზღვრავს, თუ რა შედეგს დააბრუნებს კონკრეტული ბლოკი. ტესტირების ამ ტიპისათვის საჭიროა გამოცდილი ტესტერი, რომელსაც აქვს ასევე კოდის სტრუქტურის აღქმისა თუ კოდის წერის უნარი.

რუხი ყუთის ტესტირება არის ტექნიკა, რომელშიც ხდება აპლიკაციის გატესტვა იმ შეზღუდული ცოდნის ხარჯზე, რომელიც არსებობს შიგნით – პროგრამის ლოგიკასა და სტრუქტურაში. ტესტერს ამ შემთხვევაში აქვს წვდომა პროგრამის დიზაინსა და მონაცემთა ბაზებზე. ამ ცოდნის მქონე ტესტერს უმარტივდება ტესტების მონაცემებისა და სცენარების მომზადება, ტესტ-სცენარების შედგენისას. იცის რა პროგრამული უზრუნველყოფის მუშაობის ძირითადი ლოგიკა, ტესტერი უკეთესად და მარტივად აკეთებს არჩევანს.

ვიზუალური ტესტირების მიზანია დაანახოს დეველოპერს, რა დროს ხდება პროგრამის შესრულებისას შეცდომის წარმოშობა. ვიზუალური ტესტირება მოითხოვს ჩაიწეროს ტესტირების პროცესის ყველა ეტაპი, რათა თვალნათლივ ჩანდეს, თუ რა დროს ხდება შეცდომა. ვიზუალური ტესტირების დადებითი მხარე ის არის, რომ ტესტერს შეუძლია მაგალითის ხარჯზე დეველოპერს აჩვენოს პროგრამაში მომხდარი შეცდომა, რაც მას უმარტივებს შეცდომის აღმოფხვრას. ასევე ტესტირების აღნიშნული ტიპი ამარტივებს კომუნიკაციას ტესტერსა და დეველოპერს შორის.

პროგრამულ უზრუნველყოფაში არსებობს მრავალფეროვანი აპლიკაციები, რომლებიც იყოფა გარკვეულ კლასებად, მაგალითად, Desktop აპლიკაციები, რომლებიც ძირითადად გამოიყენება გამოთვლითი სამუშაოების შესასრულებლად. ტესტები

ხშირად ჯგუფდება იმის მიხედვით, პროგრამაში თუ სად არის დამატებული ან ტესტირების დონეების მიხედვით. პროგრამული უზრუნველყოფის დეველოპმენტის დროს გამოიყენება სამი ძირითადი დონე: კომპონენტების ტესტირება, ინტეგრაციული ტესტირება და სისტემური ტესტირება, ხოლო დანარჩენი დონეები გამოიყენება ტესტირების მიზნის მიხედვით. ტესტირების ყოველ დონეს ან ყოველ ტესტს უნდა ჰქონდეს შესასვლელი სპეციალური კრიტერიუმით და ასევე გამოსასვლელი კონკრეტული კრიტერიუმებით.

კომპონენტების ტესტირების მიზანია მოხდეს პროგრამის კოდის კონკრეტული ნაწილის ფუნქციონირების შემოწმება იმ მოთხოვნებთან შესაბამისობის დასადგენად, რისთვისაც შეიქმნა კომპონენტი. ობიექტზე ორიენტირებულ დაპროგრამებაში ასეთი სახის ტესტირება დაყვანილია კლასის დონეზე, ხოლო მინიმალური დონეა კონსტრუქტორების ტესტირება. კომპონენტების ტესტირებებს, როგორც წესი, წერენ დეველოპერები, რათა დარწმუნდნენ, რომ აღნიშნული მეთოდი ისე მუშაობს, როგორც მათ ჩაიფიქრეს და განახორციელეს. კონკრეტულ მეთოდს შეიძლება ჰქონდეს რამდენიმე ტესტი ყველა იმ შესაძლო პარამეტით ტესტირებისათვის, რომლებიც შეიძლება გადაეცეს პროგრამაში მეთოდს.

ინტეგრაციული ტესტირება ითვალისწინებს პროგრამული უზრუნველყოფის ყველა იმ კომპონენტისა და მოდულის ტესტირებას, რომლებიც არის ახალი, შეცვლილი ან რომელზეც იმოქმედა სხვა ცვლილებამ. იმ დროს, როდესაც სისტემის ტესტირება ცდილობს მინიმალურ დონეზე დაიყვანოს პროგრამის ტესტირებისას გარეშე ფაქტორები, ინტეგრაციული ტესტირება მოითხოვს სხვა სისტემებისა და ინტერფეისების ჩართულობას. ინტეგრაციულ ტესტირებას აქვს კიდევ ტესტირების შიგა ტიპები, რომლებიც პროგრამის ტიპზე დამოკიდებულების მიხედვით შეიძლება იქნეს ან არ იქნეს გამოყენებული; ესენია: ტესტირება შეთავსებადობაზე, ტესტირება მწარმოებლურობაზე, ტესტირება სტრესზე და ტესტირება დატვირთვაზე.

- ტესტირება შეთავსებადობაზე არის ტესტირების ტიპი, რომელიც მოიცავს პროგრამის მუშაობას სხვადასხვა სისტემაზე. მაგალითად, ვებპროგრამის ტესტირება ხდება სხვადასხვა ინტერნეტბრაუზერზე.
- ტესტირება მწარმოებლურობაზე გამოიყენება პროგრამის მასშტაბურობის ტესტირებისათვის, მაგალითად, როდესაც აპლიკაციაში დამატებულია ბევრი მომხმარებელი ან მონაცემთა რაოდენობა იზრდება. ძირითადი მიდგომა ისაა, რომ შეაგროვოს დროითი პარამეტრების მნიშვნელობები კრიტიკული ბიზნესპროცესის დროს, როცა მაქსიმალურად დაბალ დონეზე არის დატვირთული, ხოლო შემდეგ შეაგროვოს იგივე მნიშვნელობები უკვე მაქსიმალურად მაღალი დატვირთვისას.
- ტესტირება სტრესზე გამოიყენება პროგრამის მაღალ დონეზე დატვირთვის ტესტირებისათვის. ხდება პროგრამის გაშვება მისი დატვირთულობის მაქსიმალურ ლიმიტზე, რათა მოხდეს იმ წერტილის დადგენა, სადაც ხდება შეცდომა.
- ტესტირება დატვირთვაზე არის სტრესის ტესტირების საწინააღმდეგო; ამ დროს ხდება პროგრამის შესაძლებლობების შემოწმება

ნორმალური დატვირთვით. იზომება მისი რეაგირების დრო კრიტიკული ოპერაციების შესრულებისას, რათა დადგინდეს, ის არის თუ არა ბიზნესის მოთხოვნების ფარგლებში და აკმაყოფილებს თუ არა მომსახურების დონეს.

სისტემური ტესტირება არის ტესტირება, რომელიც ამოწმებს მთლიან ინტეგრირებულ სისტემას იმაში დასარწმუნებლად, რომ სისტემა მუშაობს მისადმი წაყენებული მოთხოვნების შესაბამისად. მაგალითად, სისტემის ტესტირებისას ხდება Login ინტერფეისის ტესტირება, შემდეგ ახალი ჩანაწერების განხორციელება, მათი რედაქტირება. ასევე პრინტერზე ბეჭდვა, შემდეგ ჩანაწერების წაშლა და ბოლოს სრულდება Log off ფუნქცია. ეს იგივეა, რომ მოხდეს პროგრამის უხეშად ტესტირება და მისი მხოლოდ ძირითადი ფუნქციები შემოწმდეს. შეცდომის დაფიქსირების და მისი აღმოფხვრის შემდეგ ყველა ტესტი თავიდან უნდა იქნეს გაშვებული იმაში დასარწმუნებლად, რომ შეცდომის აღმოფხვრამ პროგრამის რაიმე სხვა ნაწილი არ დააზიანა.

ტესტირება უსაფრთხოებაზე არის ტესტირება, რომლის დროსაც ხდება პროგრამის შემოწმება უსაფრთხოებაზე ანუ პროგრამის შემოწმება ჰაკერული შემოტევებისაგან თავდასაცავად და მოწმდება, თუ რამდენად უსაფრთხოა იგი და რამდენად შესაძლებელია მისგან ინფორმაციის მოპარვა ან მსგავსი უკანონო ქმედება.

ტესტირება მისაღებობაზე არის ტესტირება, რომელსაც ასრულებს დამკვეთი იმის განსაზღვრისათვის, თუ რამდენად შეესაბამება პროგრამა მის მოთხოვნებს. ამ ეტაპის გავლის შემდეგ გადაწყდება, პროგრამა უნდა იქნეს თუ არა მიღებული.

რეგრესიული ტესტირების მიზანია, გადაამოწმოს, რომ პროგრამაში ახალმა დამატებულმა მოდულმა თუ კომპონენტმა ხომ არ გამოიწვია სისტემის ძველი ნაწილის გაფუჭება. ამისათვის ხდება ყველა არსებული ტესტის თავიდან გაშვება ახალ ტესტთან ერთად, რისთვისაც ხშირად იყენებენ ავტომატურ ტესტირებას, რათა მოხდეს ძველი ტესტების სწრაფად და მრავალჯერადად გაშვება.

განვიხილოთ ერთი კონკრეტული მეთოდის ტესტირების მაგალითი. მაგალითში გამოყენებულია დაპროგრამების ენა Java და ტესტირების ფრეიმვორკი JUNIT.

ვთქვათ, გვაქვს შემდეგი მეთოდი:

```
public class StringMethods {
    public String Concatenate(String a, String b) {
        return a + b;
    }
}
```

Unit Tests იმპლემენტირებულია კლასების სახით, რომელსაც აქვს ტესტ-მეთოდები. ხშირ შემთხვევაში ყოველი ტესტ-მეთოდი ახდენს ერთი კლასის ერთი მეთოდის ტესტირებას. ზოგ შემთხვევაში ერთი ტესტ-მეთოდი შეიძლება ახდენდეს რამდენიმე მეთოდის ტესტირებას კლასში, ხოლო ზოგჯერ თუ გასატესტი მეთოდი მოცულობით დიდია, შეიძლება დაიყოს და რამდენიმე ტესტ-მეთოდით გაიტესტოს.

ქვემოთ მოცემულია ტესტ-მეთოდი:

```
Org.Junit.Test;
Import statc org.junit.Assert. *;

public class StringMethodsTest{
```

```
@Test
public void ConcatenateTest(){
StringMethods stringMethods = new StringMethods();
String result = stringMethods.Concatenate("Saxeli", "Gvari")
 assertEquals("SaxeliGvari", result)
}
}
```

StringMethodsTest კლასი არის ტესტ-კლასი ერთი მეთოდით ConcatenateTest(), ტესტ-მეთოდი იწყება ანოტაციით @Test, რომელიც ტესტის გამშვებ პროგრამას ატყობინებს, რომ კლასში გამოყენებულია ტესტ-მეთოდი.

ConcatenateTest მეთოდის შიგნით შექმნილია StringMethods კლასის ობიექტი, რომლის საშუალებითაც შემდეგ ვიძახებთ მეთოდს Concatenate().

ასევე ტესტ-მეთოდის ტანში არის მეთოდი assertEquals(); ტესტირებას სწორედ ეს მეთოდი აკეთებს. ჩვენს მაგალითში ვადარებთ Concatenate მეთოდის გამოსავალს მოსალოდნელ შედეგს ანუ "SaxeliGvari" სტრიქონს ვადარებთ მოსალოდნელ შედეგს, მეთოდი Concatenate-დან დაბრუნებულ პასუხს, რომელიც ინახება ცვლადში Result.

თუ მოცემული ორი ცვლადი ერთმანეთის ტოლი აღმოჩნდება, მეთოდი assertEquals დააბრუნებს True-ს და ტესტირება წარმატებით შესრულდება, ხოლო თუ ორი ცვლადი არ უდრის ერთმანეთს, ადგილი ექნება გამონაკლის შემთხვევას და ტესტირება მარცხით დამთავრდება; შესაბამისად გაჩერდება გაშვებული ტესტირების პროცესი.

შესაძლებელია ტესტ-კლასში გვქონდეს იმდენი ტესტ-მეთოდი, რამდენიც მოგვესურვება. ტესტის შემსრულებელი პროგრამა იპოვნის მათ ანოტაციის მიხედვით და გაუშვებს სათითაოდ.

დასკვნა

პროგრამული უზრუნველყოფის ტესტირება უაღრესად მნიშვნელოვანია პროდუქტის მაღალი ხარისხის გარანტიისათვის. ტესტირება პროგრამული პროდუქტის სასიცოცხლო ციკლის განუყოფელი ნაწილია; ის უზრუნველყოფს დამკვეთის მოთხოვნების შესაბამისი პროდუქტის მიღებას, რომელიც იქნება მაქსიმალურად უშეცდომო. ტესტირება ასევე საშუალებას აძლევს ბიზნესის წარმომადგენლებს შეაფასონ ბიზნეს-რისკები, რომლებიც განხორციელდა პროგრამულ უზრუნველყოფაში, და დაადგინონ, რამდენად გამოსადეგია ეს პროგრამული საშუალებები მოხმარებლისათვის.

ლიტერატურა

1. Surguladze G., Gulitashvili M., Kiviladze N. Webapplications testing, validation and verification. Technical University. Tbilisi. 2015, 199p. (in Georgian).
2. URL: https://en.wikipedia.org/wiki/Software_testing
3. URL: https://www.tutorialspoint.com/software_testing/index.htm
4. URL: http://testingbasicinterviewquestions.blogspot.com/2015/03/system-testing-example-complete-guide.html
5. URL: https://www.guru99.com/test-scenario.html

6. URL: https://blog.testlodge.com/how-to-write-test-cases-for-software-with-sample/
7. URL: https://www.guru99.com/traceability-matrix.html
8. URL: http://www.cs.ubbcluj.ro/~cretu/VVSS2014_2015/Bibliography/Springer.Verlag.Practical.Software.Testing.eBook-KB.pdf

UDC 62.503.5
SCOPUS CODE 1712

Basic methods of software testing

Rusudan Kutateladze Department of Business Administration, Georgian Technical University, 77 M. Kostava str, 0160 Tbilisi, Georgia
E-mail: r.kutateladze@gtu.ge

Ana Kobiashvili Department of Economic Informatics, Georgian Technical University, 77 M. Kostava str, 0160 Tbilisi, Georgia
E-mail: anakobia@hotmail.com

Tamaz Vekua Department of Economic Informatics, Georgian Technical University, 77 M. Kostava str, 0160 Tbilisi, Georgia
E-mail: tazo.vekua@gmail.com

Reviewers:
R. Samkharadze, Professor, Faculty of Informatics and Control Systems, GTU
E-mail: rsamkharadze@mail.ru
M. Kiknadze, Professor, Faculty of Informatics and Control Systems. GTU
E-mail: m.kiknadze@gtu.ge

Abstract. Software performance is very important. Errors and malfunctions during functioning of various applications may result in serious financial losses. That is why the qualitative testing of software is actual. The article describes software testing documentation, its core components, the software development and stages of its work. Two basic types of testing, particularly, manual and automated testing, are reviewed. An example is drawn up which is performed on the programming language JAVA, using JUNIT framework. The most essential Assert class methods for the testing with framework JUNIT, along with relevant practical example, are discussed. The paper also briefly examines various methods of software testing and the cycle of testing in general.

Key words: Algorithm; framework; software testing; tester.

UDC 62.503.5
SCOPUS CODE 1712

Основные методы тестирования программного обеспечения

Русудан Кутателадзе Департамент бизнес-администрирования, Грузинский технический университет, Грузия, 0160, Тбилиси, ул. М. Костава, 77
E-mail: r.kutateladze@gtu.ge

Ана Кобиашвили Департамент экономической информатики, Грузинский технический университет, Грузия, 0160, Тбилиси, ул. М. Костава, 77
E-mail: anakobia@hotmail.com

Тамаз Векуа Департамент экономической информатики, Грузинский технический университет, Грузия, 0160, Тбилиси, ул. М. Костава, 77
E-mail: anakobia@hotmail.com

Рецензенты:

Р.Самхарадзе, профессор факультета информатики и систем управления ГТУ
E-mail: rsamkharadze@mail.ru
М.Кикнадзе, профессор факультета информатики и систем управления ГТУ
E-mail: m.kiknadze@gtu.ge

Аннотация. Очень важна исправная работа программного обеспечения. Ошибки и неисправности в работе различных приложений могут привести к серьезным финансовым потерям. Именно поэтому чрезвычайно актуально качественное тестирование программного обеспечения. В статье описывается документация по тестированию программного обеспечения, ее основные компоненты, разработка программного обеспечения и этапы работы, рассматриваются два основных типа тестирования: мануальное и автоматическое тестирование. Приведен пример на языке программирования JAVA с использованием фреймворка JUNIT. Рассматриваются наиболее важные методы оценки Assert, применяемые при тестировании с использованием фреймворка JUNIT, а также соответствующий практический пример. В статье также кратко рассматриваются различные методы тестирования программного обеспечения и цикл тестирования в целом.

Ключевые слова: алгоритм; тестер; тестирование программного обеспечения; фреймворк.

განხილვის თარიღი 28.05.2019
შემოსვლის თარიღი 14.05.2019
ხელმოწერილია დასაბეჭდად 17.12.2019