KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Eivilė Jankevičiūtė

Programinės įrangos testavimo proceso valdymo automatizavimas

Magistro darbas

Darbo vadovas:

doc. dr. Rita Butkienė

KAUNAS, 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Eivilė Jankevičiūtė

Programinės įrangos testavimo proceso valdymo automatizavimas

Magistro darbas

Recenzentas:	**Vadovas:**	**Atliko:**
Doc.dr. S.Maciulevičius	Doc.dr. R.Butkienė	IFM-2/4 gr.stud. E.Jankevičiūtė
2008-01-14	2008-01-14	2008-01-14

KAUNAS, 2008

Summary

Automation of Software Testing Process Management

Due to fast information technology spread, software quality becomes one of the most important issues. More and more attention is paid to software testing process and integration into whole software development process to increase quality. In this paper software development process problems are analyzed, highlighting the importance of software testing process. Software testing process management system model is proposed which enables integrate whole software testing process and efficiently manage testing process collecting information from other stages. Software testing process management problems are analyzed.

The field of this work is software testing process. The object of this work is automation of software testing process and integration of it into whole software development process.

After analyzing similar tolls for managing software testing process turned out that they do not meet requirements for automation of software testing process management. Also analysis has showed that none of analyzed similar solutions has high level using testing maturity model TMM.

The main aims of this work are to improve and to automate software process management and integrate it into software development process, and to create a testing process management system, which would meet desired requirements. During this work a solution was proposed that manages software testing process and after TMM has at least 3rd or even 4th level.

The overview of this solution was presented at the conference „IVUS‘12“ that was arranged in May, 2007.

Turinys

Paveikslų turinys

Lentelių turinys

Įvadas

XXI – ojo amžiaus pradžioje, kada gyvenimas neįsivaizduojamas be kompiuterių ir stengiamasi, kad kiekvienas įmonės verslo procesas būtų kompiuterizuojamas, per metus sukuriama milijonai vienokių ar kitokių, stambesnių ar smulkesnių, sudėtingesnių ar paprastesnių programinės įrangos (sutr. PĮ) produktų. Programinės įrangos kūrimas yra iteratyvus procesas, kurio metu visi procese dalyvaujantys asmenys privalo harmoningai ir betarpiškai bendradarbiauti [1]. Šis bendradarbiavimas yra esminis visame PĮ kūrimo procese, o jo efektyvumas yra didžiausias produkto sėkmingumo garantas. Šiame kontekste svarbiausias uždavinys – nagrinėjama problema – yra užtikrinti sukuriamos programinės įrangos kokybę. Dar 1980 metais **kokybė** tapo vienu iš pagrindinių programinės įrangos tikslų [5]. Pradėtas studijuoti programinės įrangos gyvavimo ciklas, programinės įrangos kūrėjai ir testuotojai pradėjo dirbti kartu. Programinės įrangos kokybė bei produktyvumas šiandien yra svarbiausi klausimai programinės įmonės pramonėje [9]. Ją užtikrina eilė procesų, kurie formaliai ar neformaliai būna kiekviename programinės įrangos kūrimo projekte. Išskiriami keturi pagrindiniai inžineriniai procesai: **reikalavimų analizė**, **projektavimas**, **programavimas** ir **testavimas**. Po šių procesų seka programinės įrangos **dokumentavimas**, o visus šiuos procesus apjungia **projekto valdymas**, kurį kontroliuoja projektų vadovas [2].

Programinės įrangos patikimumas yra didelė auganti problema. Nors yra vertinamas patikimas testavimas ir dažniausiai tai būna brangiausia PĮ kūrimo fazė visame procese, tačiau dauguma atvejų, tai vis dar yra problema. Paprastai testavimas yra paremtas intuicija ir testavimo procesas [2, 16] nėra formalizuotas. Testuotojas intuityviai parenka testavimo atvejus ir pagal juos testuoja sistemą, tačiau tai gali iššaukti klaidų ar spragų tikimybę, kaip žmogiškojo faktoriaus pasekmė. Tokiu atveju, projekto finansavimas, skirtas šiam moduliui, gali pasibaigti, efektyviai neišnaudojus visų galimybių.

Nagrinėjamas programinės įrangos kūrimo procesas turi būti integruojamas į vieną bendrą visumą, užtikrinant programinės įrangos kūrimo integralumą ir efektyvų kokybės valdymą.

Nagrinėjamam darbe sekančiame skyriuje bus atlikta programinės įrangos testavimo proceso analizė bei egzistuojančių įrankių įvertinimas. Vėliau bus pateikiama programinės įrangos kūrimo ir testavimo valdymo automatizavimo sistemos reikalavimų specifikacija ir analizė. Toliau apžvelgiamas programinės įrangos kūrimo ir testavimo valdymo sistemos projektas, po kurio seks realizacija bei eksperimentinis tyrimas. Darbo pabaigoje bus pateikiamos išsamios išvados bei naudotos literatūros sąrašas, o prieduose pateikiamas publikuotas straipsnis dalykinės srities tema bei papildoma medžiaga.

2. Programinės įrangos testavimo proceso analizė ir egzistuojančių įrankių įvertinimas

2.1. Analizės tikslas

Šiame skyriuje bus analizuojamos pagrindinės programinės įrangos testavimo funkcijos. Norint suprasti šias funkcijas, reikia ištirti programinės įrangos testavimo atvejų sudarymą, pačius testavimo procesus, klaidų identifikavimą bei kontrolę, programinės įrangos kokybės valdymą bei informacijos dokumentacijai pateikimą.

Nagrinėjant probleminę sritį, bus suformuluoti reikalavimai, keliami programinės įrangos testavimo procesui valdyti bei automatizuoti. Taip pat analizės metu bus nustatyti vertinimo kriterijai, pagal kuriuos bus įvertinti bei išanalizuoti jau egzistuojantys sprendimai.

Išnagrinėjus programinės įrangos testavimo procesą bei su juo susijusius etapus bus galima tiksliai apsibrėžti reikalavimus, kaip užtikrinti programinės įrangos testavimo proceso valdymą bei jo automatizavimą.

2.2. Tyrimo sritis, objektas, problema ir tikslas

Norint užtikrinti programinės įrangos testavimo proceso valdymą bei jo automatizavimą, reikia nagrinėti ne tik testavimo procesą, bet ir visą programinės įrangos kūrimo procesą.

Šiuo metu yra žinoma daug testavimo metodikų, įvairūs testavimo metodai papildo, perdengia vienas kitą. Tačiau pasaulyje neegzistuoja tokia testavimo metodologija, kurią pritaikius galima būtų pasakyti, kad sistema neturi klaidų.

Testavimo procesas yra vienas svarbiausių procesų programinės įrangos kūrime. Testavimą būtina pradėti dar projekto kūrimo stadijoje. Todėl programinės įrangos testavimo procesas turi būti kuo tampriau susiejamas su visais kitais programinės įrangos kūrimo procesais. Tam pasitarnauja vadinamasis „V“ modelis, o proceso brandumui užtikrinti yra naudojamas brandumo modelis TMM [4]. Šie modeliai bus plačiau pristatyti sekančiuose skyreliuose.

Išnagrinėjus programinės įrangos kūrimo bei testavimo procesus, bus galima pasirinkti priimtiniausią programinės įrangos testavimo metodą, pagal kurį turėtų vykti programinės įrangos testavimas. O išnagrinėjus programinės įrangos testavimo brandumo modelį, bus galima įvertinti testavimo procesą, pasitelkiant į pagalbą jau egzistuojančius įrankius. Po egzistuojančių įrankių

įvertinimo, bus galima išsikelti tikslus programinės įrangos testavimo proceso valdymui pagerinti bei automatizuoti.

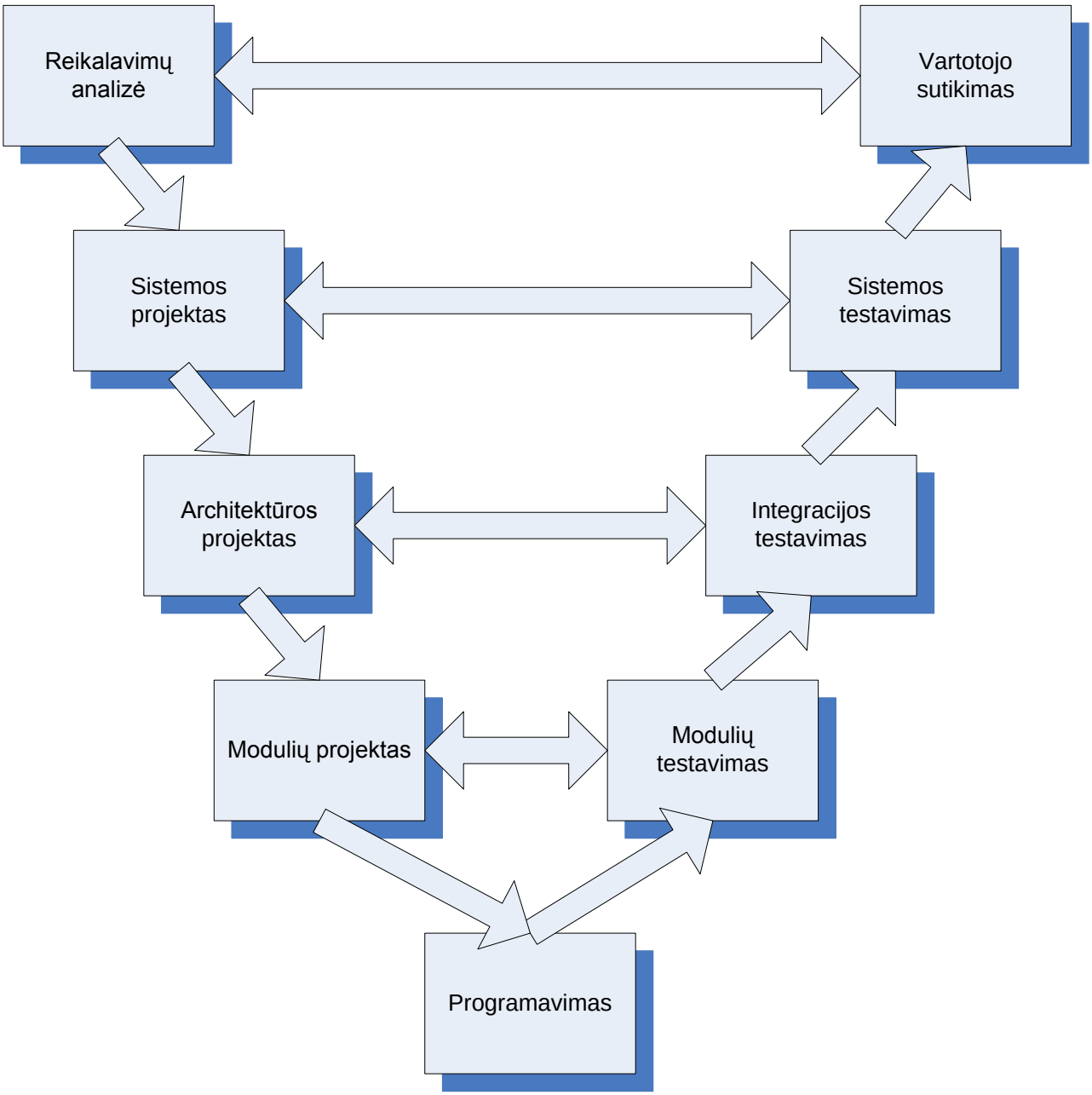
Taigi apibendrinus galima teigti, kad šio tyrimo tikslas yra pagerinti ir automatizuoti programinės įrangos testavimo proceso valdymą, integruoti jį į visą programinės įrangos kūrimo procesą bei sukurti sistemą, atitinkančią iškeltus reikalavimus bei realizuojančią reikiamas funkcijas.

2.3. Testavimo proceso analizė

Literatūros šaltiniuose, kuriuose kalbama apie programinės įrangos testavimo procesą, rekomenduojama, kad programinės įrangos kūrimo procesas vyktų pagal vadinamąjį „V“ modelį [16, 17]. Informacija, reikalinga sekančiam etapui, būtų gaunama ir tampriai susieta su informacija iš prieš tai vykusio proceso. Sekančiame skyrelyje pristatomas šis modelis.

2.3.1. Programinės įrangos kūrimo proceso „V“ modelio analizė

„V“ modelis – tai viena iš programinės įrangos kūrimo metodologijų. Šioje metodologijoje programinės įrangos kūrimas ir testavimas yra vykdomi lygiagrečiai, tuo pačiu metu ir yra vienodai svarbūs. Šį procesą grafiškai patogu vaizduoti „V“ raidės formos, todėl jis turi tokį pavadinimą. Kairėje pusėje vaizduojami programinės įrangos kūrimo proceso etapai, o dešinėje – testavimo proceso etapai bei šių etapų sąveikos (Pav.1). Šis modelis yra tarsi transformuotas bei išplėstas krioklio modelis, tik užuot vaizdavus visus procesus vieną paskui kitą krentančius žemyn viena linija, čia linija užsilenkia ties programavimo etapu ir vėl kyla į viršų. Lygiagrečiai vienoje eilėje esantys procesai sąveikauja tarpusavyje. Programinės įrangos testavimo procese kiekviename etape informacija gaunama iš lygiagrečiai vykstančio programinės įrangos kūrimo etapo. Taip programinės įrangos plėtojimas tampa efektyvesnis dėl dviejų priežasčių. Pirmoji – sutaupoma laiko lygiagrečiai vystant procesus. O antroji priežastis – nagrinėjant tą patį etapą vienu metu skirtingais pjūviais, jis yra geriau išpildomas. Taigi testavimo etapui yra reikalinga informacija, gaunama iš programavimo etapo. Testuotojas, turėdamas šią informaciją, sudaro testavimo atvejus, pagal kuriuos yra testuojama sistema. Testavimo atvejai sudaromi tiesiogiai iš programos specifikacijų, o iš testavimo atvejų bei papildomos informacijos apie projektą bei modulį sudaromas testavimo planas.



Pav. 1 Programinės įrangos kūrimo „V“ modelis

Kaip matoma diagramoje, visi procesai tarpusavyje susiję ir yra vieni nuo kitų priklausomi. Todėl nagrinėjamas programinės įrangos kūrimo procesas turi būti integruojamas į vieną bendrą sistemą, kuri užtikrintų programinės įrangos kūrimo integralumą ir efektyvų programinės įrangos kokybės valdymą [10].

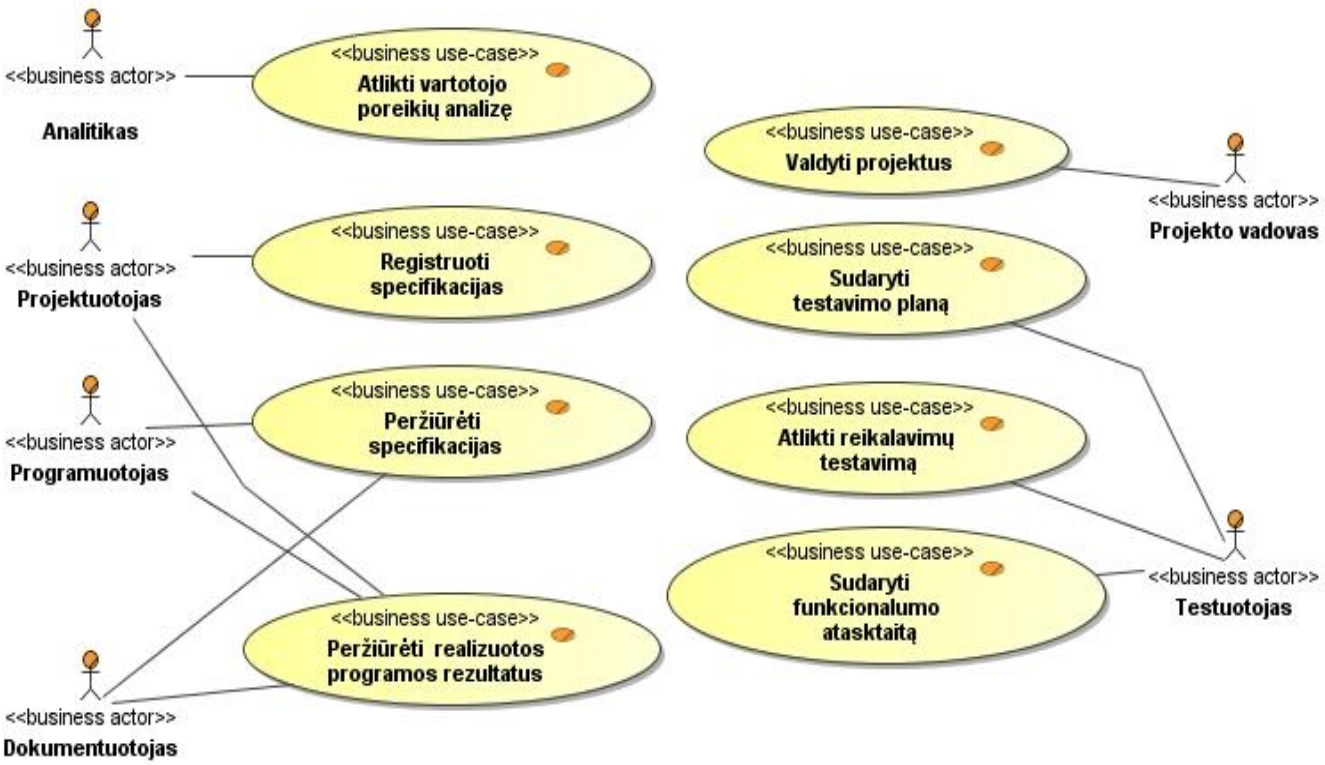
2.3.2. Aplinkos analizė

Šiame skyrelyje bus aptariamas programinės įrangos kūrimo procesas bei pristatomas jo veiklos uždavinių modelis.

Programinės įrangos kūrimo metu procese dalyvaujančių darbuotojų veiklos:

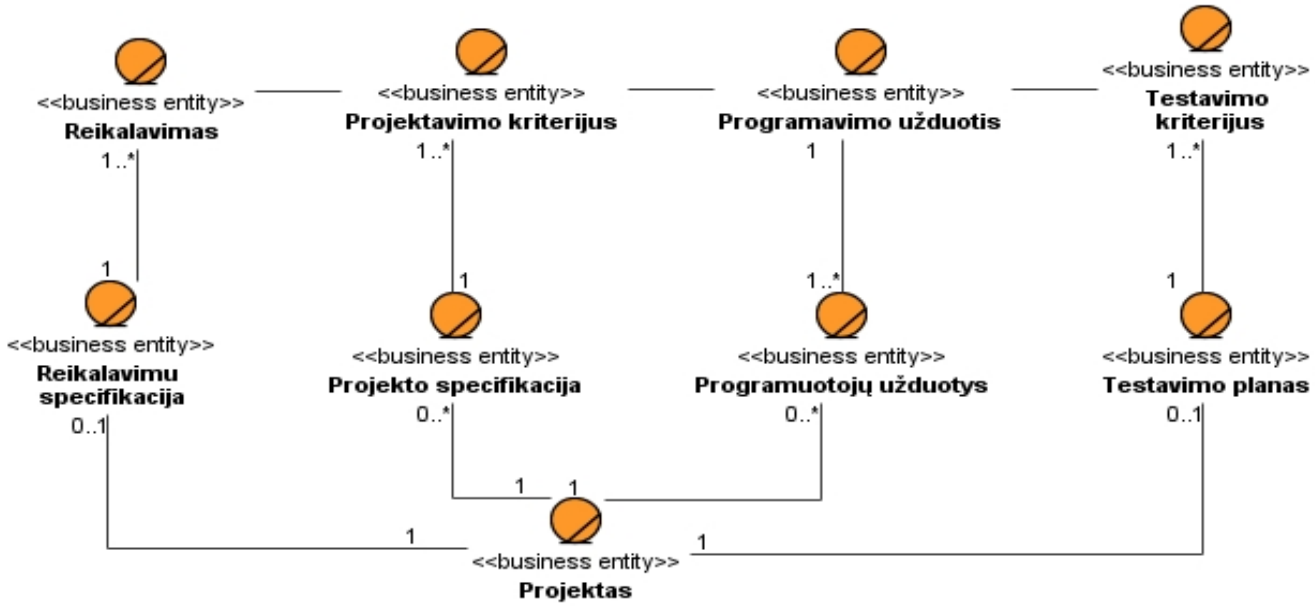
- Projektavimo metu projektuotojas nustato programinės įrangos specifikacijas.
- Pagal jas programuotojai sudaro programą.
- Testuotojas pagal sudarytas specifikacijas sudaro testavimo planą, pagal kurį testuoja programinę įrangą ir vėliau pateikia ataskaitą.
- Pagal sudarytą ataskaitą bei projektuotojų specifikacijas dokumentuotojas sudaro programos dokumentaciją bei vartotojo vadovą.

Tai matoma sekančiame paveiksle (pav. 2), kuriame vaizduojamas veiklos uždavinių modelis.



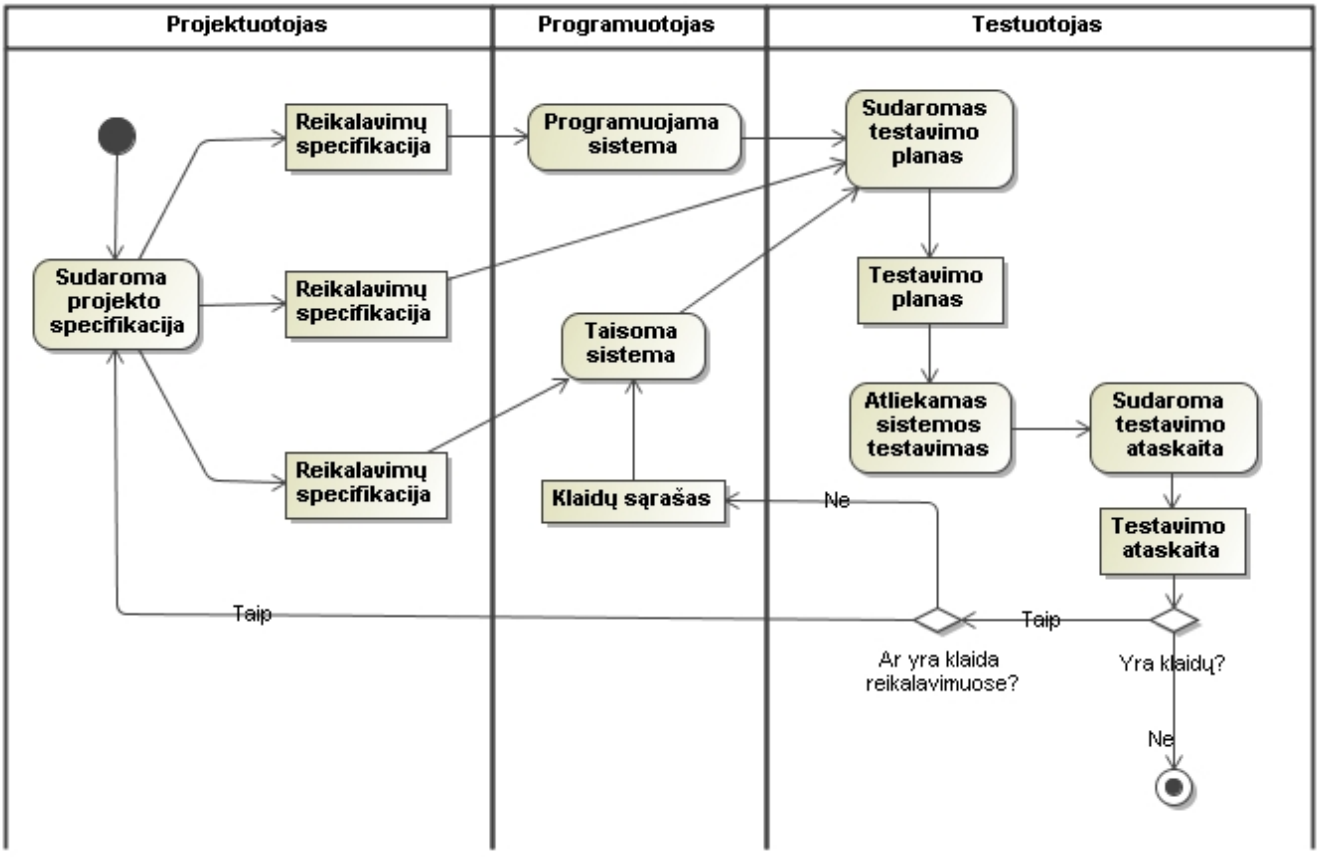
Pav. 2 Veiklos uždavinių modelis

Veiklos esybių modelyje (pav. 3) vaizduojamos projekte dalyvaujančios esybės ir ryšiai tarp jų. Projekto kūrimo metu sudaromas reikalavimų specifikacija ir testavimo planas. Testavimo planas pasako kas testuojama, tuo tarpu pati testavimo procedūra pasako kaip tai bus atliekama. Testavimo kriterijams yra priskirta vienas arba daugiau reikalavimų:



Pav. 3 Veiklos esybių modelis

Veiklos proceso modelyje (pav. 4) matomas visas programinės įrangos kūrimo procesas, suskaidytas į atskirus etapus, kurie kiekvienas priskirti tam tikrai būsimos sistemos vartotojų grupei:



Pav. 4 Veiklos proceso modelis

2.4. Vartotojų analizė

2.4.1. Vartotojų aibė, tipai ir savybės

Kuriamos sistemos vartotojai yra visi į programinės įrangos kūrimo procesą patenkantys įmonės darbuotojai. Šiame darbe gilinamasi į programinės įrangos testavimo procesą.

Pagrindinis sistemos vartotojas yra PĮ testuotojas, kuris pagal gautas specifikacijas sudaro testavimo planus, testuoja sistemą ir pateikia analizės ataskaitą. Galimi sistemos vartotojai:

- Projekto vadovas;
- Analitikas;
- Projektuotojas;
- Programuotojas;
- Testuotojas;
- Dokumentuotojas.

Kiekvienas vartotojas, prisijungęs prie sistemos mato su jo valdomu procesu susijusią informaciją. Tačiau šiame darbe planuojama realizuoti su testavimo procesu susijusių dalių modulius. Nerealizuotos sistemos dalys bus paliekamos kaip sistemos išplėtimo galimybė.

2.4.2. Vartotojų tikslai ir problemos

Būsimi sistemos vartotojai – programinės įrangos gyvavimo ciklo aktoriai – sistemą vartos programinės įrangos kūrimo proceso, o ypač testavimo proceso valdymo automatizavimui. Sistema bus kaip pagalbinė ir pirmoji priemonė, padedanti valdyti šiuos procesus ir leidžianti turėti visą reikalingą informaciją vienoje vietoje. Pagrindinis būsimų vartotojų tikslas yra sugebėti suvaldyti testavimo procesą ir užtikrinti programinės įrangos kokybę.

Kokybės užtikrinimas yra probleminė sritis, su kuria susiduria programinės įrangos gamintojai, kiekvieną dieną pasaulyje sukuriantys vis naujų programinės įrangos produktų, kurie norint kad patenkintų potencialų vartotoją, turi būti galutinai ištestuoti, kad atitiktų vartotojų poreikius. Taigi sprendžiamos problemos yra šios:

- programinės įrangos patikimumo bei kokybės užtikrinimas,
- testavimo proceso apjungimas į programinės įrangos kūrimo procesų visumą,
- šio proceso automatizavimas.

Tokioms problemoms spręsti labai patogu pasitelkti būsimą sistemą, automatizuojančią minėtų procesų valdymą bei apjungiančią informaciją, reikalingą šiems procesams, į vieną visumą.

2.4.3. Reikalavimų būsimai sistemai nustatymas

Norint užtikrinti programinės įrangos kūrimo integralumą ir efektyvų programinės įrangos kokybės valdymą, programinės įrangos kūrimo procesas turi būti integruojamas į vieną bendrą sistemą. Tam tikslui turi būti sukurta informacinė sistema, kuri apimtų tokius modulius su savom funkcijom, kaip:

- Projekto valdymo dalis:

Naujų projektų pradėjimas, PĮ plano apibrėžimas ir vystymas, proceso kontrolė ir stebėjimas.

- Sistemos analizės dalis:

Suprasti vartotojo poreikius, dokumentuoti ir tvirtinti reikalavimus.

- Sistemos projektavimo dalis:

Sudaryti sistemos projektą, specifikuoti reikalavimus.

- Realizacijos dalis:

Suprasti reikalavimus, pagal pateiktą specifikaciją realizuoti programą, kuri atitiktų keliamus reikalavimus.

- **Testavimo dalis:**

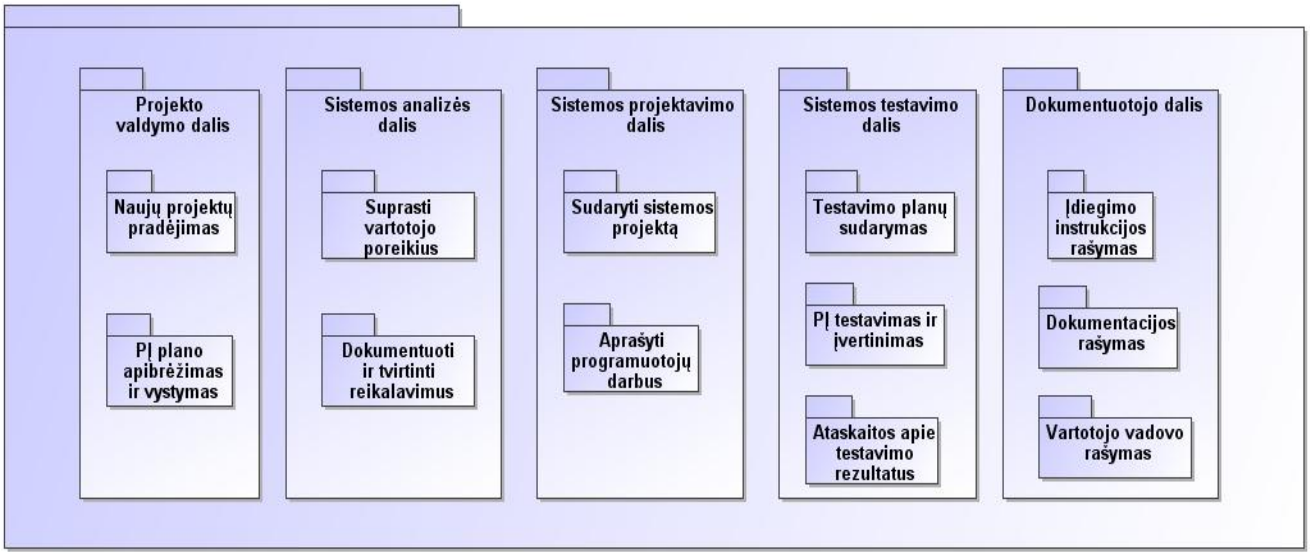
Pagal pateiktas specifikacijas sudaryti testavimo planus, pagal juos ištestuoti ir įvertinti programinę įrangą, pateikti ataskaitas apie testavimo rezultatus.

- Dokumentavimo dalis:

Suprasti programinės įrangos funkcionalumą, suprasti vartotojo poreikius, rašyti dokumentus PĮ vartotojams (įdiegimo instrukcija, dokumentacija, vartotojo vadovas).

Programinės įrangos **testavimo dalis** su visomis tą dalį apimančiomis funkcijomis turi būti realizuota praktiškai, paliekant integraciją su kitais moduliais ir pačių modulių realizavimą kaip programos išplėtimo galimybes. Taip pat kaip išplėtimo galimybė gali būti testavimo planų sudarymo dalies automatizavimas. Šios sistemos praktinės realizacijos numatytos funkcijos turėtų aprėpti klaidų valdymo sistemos (angl. *bug tracking system*) funkcijas bei turėti kitų, o visa tai turėtų būti suintegruota į bendrą sistemą, apimančią su testavimo procesu susijusius programinės įrangos kūrimo proceso etapus.

Principinė programinės įrangos kūrimo etapų schema vaizduojama sekančiame paveiksle (Pav. 5).



Pav. 5 Programinės įrangos kūrimo etapų schema

Vadovaujantis šia schema, galima sukurti sistemą, padedančią ne tik valdyti bei automatizuoti programinės įrangos testavimo procesą, užtikrinti programinės įrangos kokybę, bet ir testavimo procesą padaryti brandžiu (angl. *Testing Maturity model*) [7, 8]. Kuriamos programinės įrangos kokybė turi būti užtikrinama ne tik klaidų nebuvimu, bet ir viso numatyto sistemos funkcionalumo nepriekaištingu veikimu, patogumu vartoti bei pagalbos vartotojui suteikimu. Visi šie faktoriai turi būti išpildyti, norint užtikrinti programinės įrangos kokybę. Tam yra reikalinga informacija, kuri turi būti suintegruota ir patogiai pasiekiama vienoje sistemoje, kad būtų galima nuodugniai vystyti produktą. Pagal testavimo brandumo modelį (TMM), kokybė turi būti išpildyta atsižvelgiant į aukščiau išvardintus faktorius. Programinės įrangos proceso brandumo modelis TMM nusako taisykles, kaip aiškiai apibrėžti, valdyti, vertinti, reguliuoti kūrimo procesą ir daryti jį efektyvų.

2.5. Problemos sprendimo metodų apžvalga

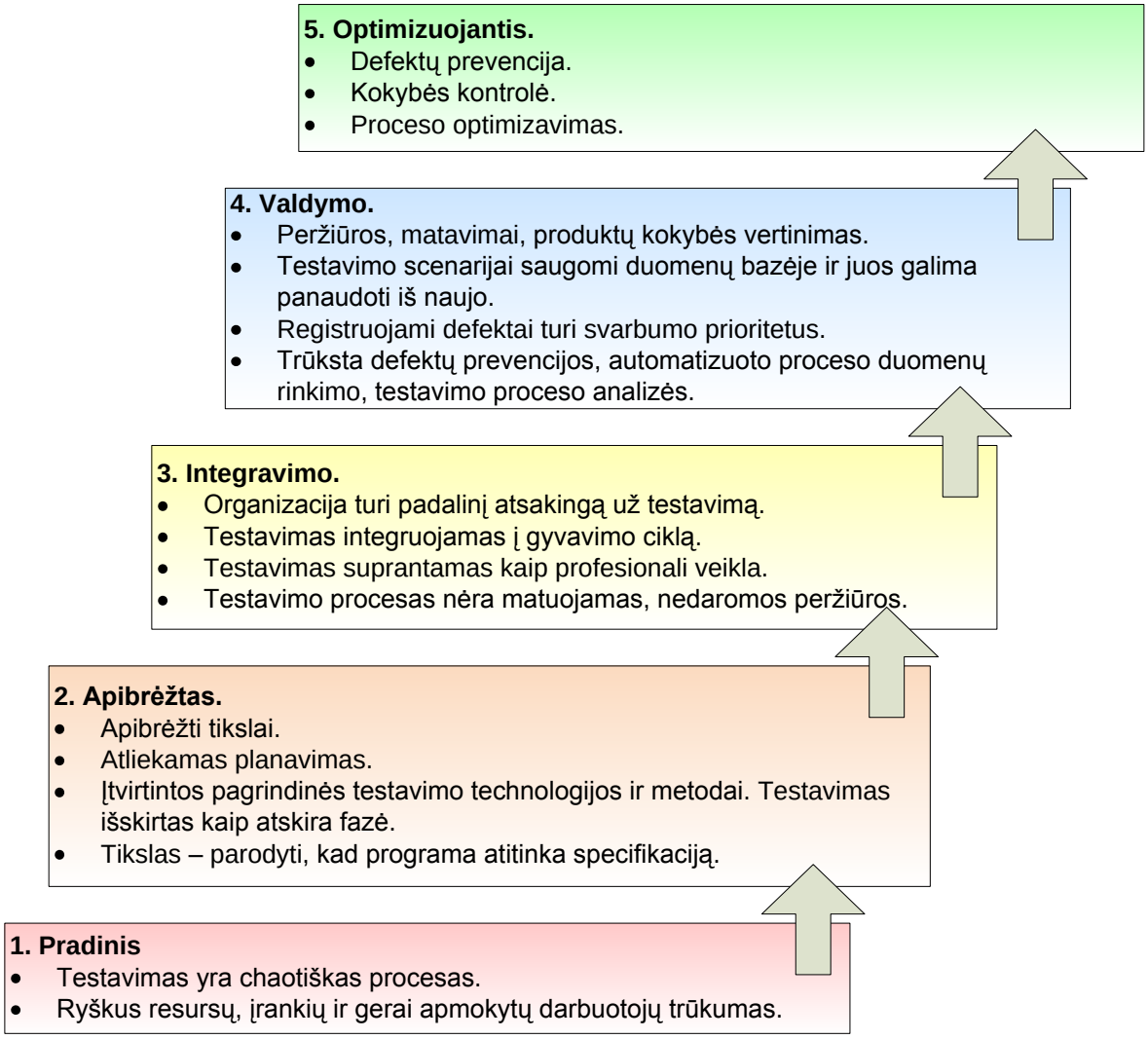
Kaip jau minėta, rekomenduojama kad programinės įrangos testavimo procesas vyktų pagal vadinamąjį „V“ modelį. Tokiu atveju, pats testavimo procesas būtų glaudžiai susietas su kitais programinės įrangos kūrimo procesais. Informacija, reikalinga sekančiam etapui vykdyti, būtų gaunama iš prieš tai buvusio programinės įrangos kūrimo etapo.

Norint įvertinti programinės įrangos testavimo proceso brandumo lygį, taikomas Testavimo brandumo modelis (TMM, ***angl.*** *Testing Maturity Model*), nusakantis programinės įrangos testavimo brandumo lygį bei įvertinantis kuriamos programinės įrangos brandumą.

TMM yra charakterizuojamas penkiais testavimo brandumo lygmenimis: tikslų, potikslių, veiklų, užduočių ir atsakomybių [6,7,8]. Pagrindiniai TMM modelio lygiai yra:

1. Pradinis;
2. Fazių apibrėžimas;
3. Integracija;
4. Valdymas ir įvertinimas;
5. Optimizavimas, defektų atsiradimo mažinimas ir PĮ kokybės valdymas.

Šie lygiai vaizduojami paveiksle pav.6.



Pav. 6 TMM modelio lygiai

Pradinio lygmens savybės: nėra jokių brandumo tikslų, testavimas vyksta chaotiškai, testai kuriami pasibaigus kodavimui, testavimas neatskirtas nuo derinimo, testavimo tikslas – parodyti, kad programinė įranga veikia, PĮ išleidžiama be kokybės užtikrinimo.

Fazės apibrėžimas: testavimas atskirtas nuo derinimo ir apibrėžtas kaip fazė sekanti kodavimą. Testo planavimas gali vykti ir po kodavimo dėl proceso nebrandumo. Pagrindinis brandumo tikslas yra parodyti, kad PĮ atitinka specifikaciją. Kadangi testo planavimas atliekamas vėlai PĮ gyvavimo cikle, tai defektai dažnai atsiranda dėl klaidų reikalavimų bei projektavimo fazėse. Testavimas po kodavimo vis dar laikomas pirmine testavimo veikla.

Integracija: testavimas integruotas į PĮ gyvavimo ciklą. Testavimo planavimas prasideda reikalavimų fazėje bei tęsiasi viso PĮ gyvavimo ciklo metu. Testo tikslai yra nustatomi atsižvelgiant į reikalavimus paremtus vartotojo poreikiais ir jais remiamasi kuriant testinius atvejus. Testavimas pripažįstamas profesionalia veikla. Organizuojami techniniai testavimo apmokymai.

Valdymas ir matavimas: testavimas yra matuojamas ir vertinamas procesas. Peržiūra visose PĮ kūrimo fazėse yra pripažįstama kaip testavimo ir kokybės kontrolės veikla. Testuojami PĮ produktų tokie kokybės kriterijai, kaip patikimumas, panaudojamumas, palaikomumas. Testiniai atvejai iš visų projektų yra surenkami į duomenų bazę bei pakartotinai panaudojami. Defektai yra registruojami ir jiems priskiriamas svarbumo lygis.

Optimizacija, defektų prevencija ir kokybės kontrolė: laikoma, kad testavimo proceso kaštus bei efektyvumą galima įvertinti ir stebėti, dėl ankstesniuose 4-iuose TMM lygiuose pasiektų tikslų:

- Pastoviai vyksta procesai, kurie tobulina testavimą;
- Praktikuojama defektų prevencija bei kokybės kontrolė;
- Nustatyta procedūra testavimo įrankių pasirinkimui bei įvertinimui.

Brandumo potiksliai yra sudaromi iš veiklų ir užduočių su atsakomybėmis (sutr. angl. ATR). ATR atvaizduoja problemos sprendimą ir įgyvendinimą tam tikru lygmeniu. Veiklos ir užduotys yra apibrėžiamos veiksmais, kurie turi būti atlikti tame lygmenyje, kad padidinti testavimo galimybes. Atsakomybės, kurios yra priskiriamos šios veikloms ir užduotims, yra skirstomos į 3 grupes ir atvaizduoja pagrindinius dalyvius testavimo procese: vadovas, programuotojai ir testuotojai, vartotojai ir klientai. TMM modelyje jie vadinami "trim kritiniais vaizdais". Vadovo vaizdas apima įsipareigojimus ir galimybes atlikti veiklas ir užduotis, kurios padidintų testavimo galimybes. Programuotojų ir testuotojų vaizdas apima techninius veiksmus ir užduotis, kurios taikomos ir sukuria brandų testavimo procesą. Vartotojų ir klientų vaizdas yra apibrėžtas kaip bendradarbiavimo ir palaikymo vaizdas. Programuotojai ir testuotojai dirba su klientų ir vartotojų grupėmis, atlikdami užduotis ir veiklas, susijusias su programinės įrangos patikimumu, kokybe ir vartotojų poreikiais. Programinės įrangos patikimumas dažnai yra apibrėžiamas kaip jos naudojimosi be sutrikimų ir bendro jos naudojimosi laikų santykis.

Vadovaujantis šiuo modeliu, turi būti sukurta sistema, apimanti testavimo procesą bei su juo susijusių procesų informaciją.

2.6. Panašių sistemų (Lietuvos ir tarptautiniu mastu) analizė

Įvairūs testavimui specializuoti produktai dažniausiai apima tik tam tikrą siaurą sritį ir įmonėse dažniausiai naudojami tam tikrai informacijai bei duomenis apdoroti kaip pagalbinės priemonės. Šiame skyriuje palygintos kelios programos, dažniausiai naudojamos nedidelėse bei vidutinėse įmonėse: *Mantis* (klaidų sekimo programa), *Bugzilla* (internetinė klaidų registravimo sistema), *Test Link* (testų rašymo programa). Šios sistemos yra atrinktos analizei dėl to, kad kiekviena jų atskirai geriausiai atspindi testavimo procesui reikalingas savybes. Versijavimo valdymo sistemos (tokios kaip *Win CVS* ar *Tortoise CVS*) yra atmetamos, nes jos labiau skirtos programavimo procesui valdyti bei automatizuoti.

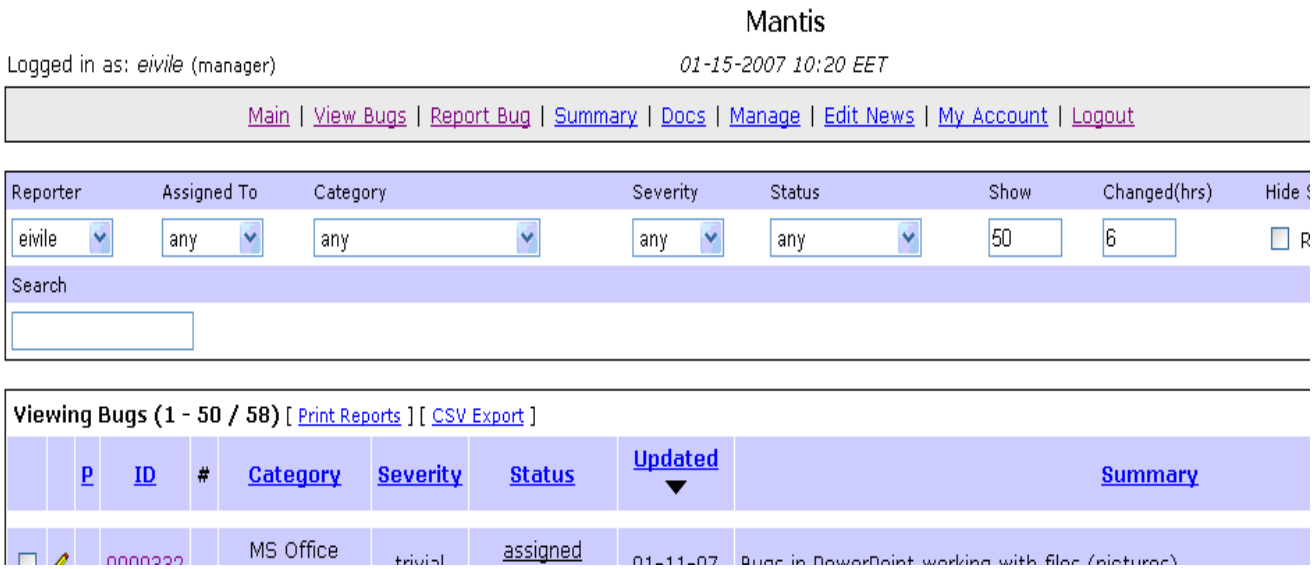
Kadangi siekiama automatizuoti bei integruoti programinės įrangos testavimo proceso valdymą, išsikeliami kriterijai, kuriuos išpildžius rezultatas būtų pasiektas. Analizėje naudojami palyginimo kriterijai:

- Galimybė sudaryti testavimo atvejus;
- Galimybė sekti bei valdyti klaidas;
- Galimybė generuoti diagramas apie produkto kokybę (klaidų skaičių bei svarbumą);
- Galimybė kaupti užregistruotų klaidų statistiką;
- Vartotojo sąsajos patogumas, lankstumas;
- Produkto kainos dydis;
- TMM brandumo lygmuo.

Toliau išvardinami egzistuojantys sprendimai, atrinkti šiai analizei atlikti bei jų savybės.

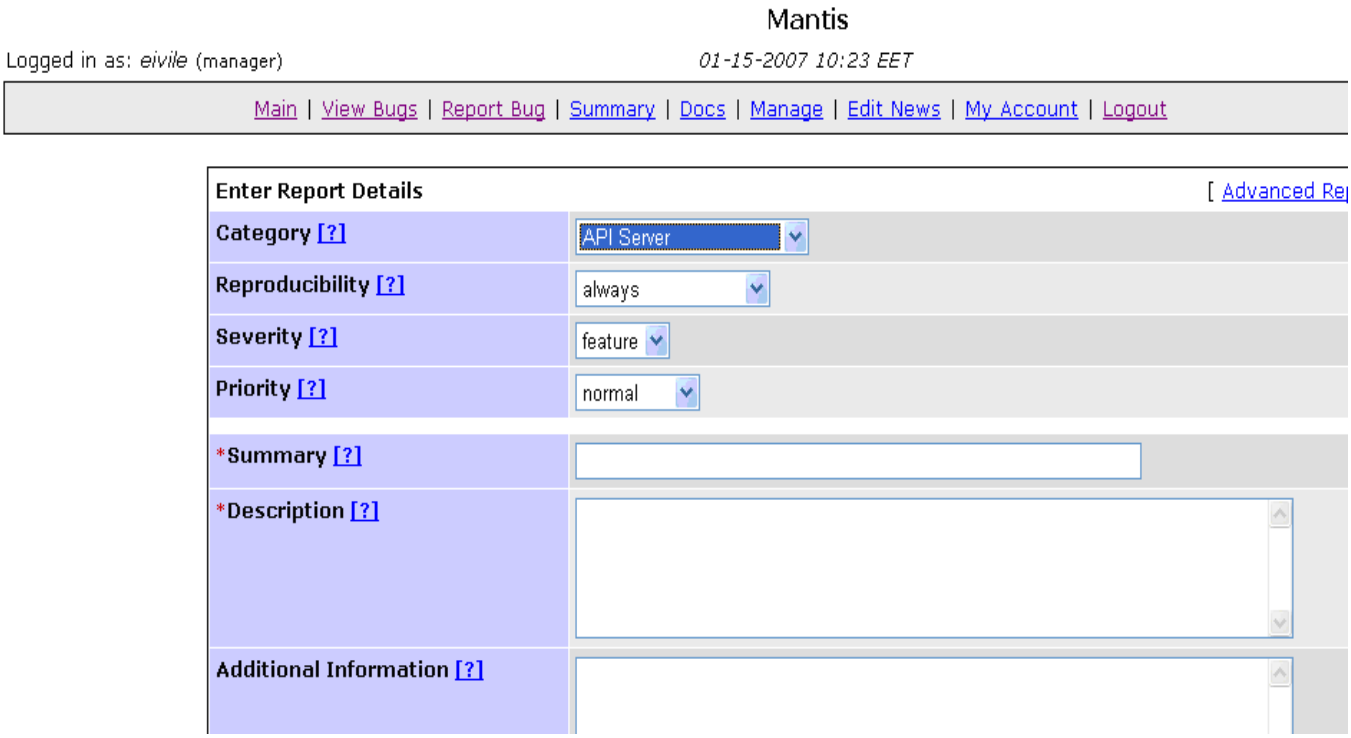
***Mantis* – klaidų sekimo internetinė programa** [12]. Ji leidžia užregistruoti surastas klaidas, joms priskirti statusą, svarbumo lygį, kategoriją, atkartojamumą, nukreipti konkrečiam darbuotojui, pridėti žinutę (angl. *bugnote*), prisegti failą, susieti klaidas, filtruoti pagal tam tikrus kriterijus. Vartotojo sąsaja yra patogi, maloni ir aiški. Tačiau šioje sistemoje negalima rašyti testavimo atvejų, kurti testavimo planų, rašyti ataskaitų, generuoti diagramų apie produkto kokybę. Joje galima matyti tik statistiką apie visas užregistruotas klaidas įvairiais pjūviais. Tačiau šis produktas yra nemokamas.

Prisijungus prie sistemos, galima matyti visas sistemoje įvestas klaidas (Pav. 7), jas peržiūrėti naudojant įvairius filtrus, tokius kaip pagal klaidos reporterį, pagal tai, kam klaida yra priskirta, pagal kategoriją, kuriai ji priklauso, pagal klaidos svarbumą ar statusą:



Pav. 7 Sistemoje matomos įvestos klaidos

Pranešti (įvesti) klaidą į sistemą galima paspaudus nuorodą “Report Bug” (Pav. 8). Tuomet atsiverčia klaidos registravimo langas, kuriame galima įvesti trumpą klaidos aprašymą bei apibūdinimą, taip pat visus aukščiau minėtus kriterijus, papildomą informaciją, prisegti failą:



Pav. 8 Klaidos registravimo langas

***Bugzilla* – internetinė klaidų registravimo sistema** [13]. Kaip ir Mantis, ši sistema leidžia registruoti klaidas, joms priskiriant statusą, svarbumo lygį, kategoriją, atkartojamumą, nukreipti konkrečiam darbuotojui, pridėti žinutę, prisegti failą, susieti klaidas, matyti statistiką. Ji taip pat neturi testavimo planų valdymo funkcionalumo. Bet šis produktas taip pat yra nemokamas.

***Test Link* – testų rašymo programa** [14]. Ji leidžia rašyti arba įkelti programos testus, kurti testavimo atvejus, rašyti testavimo žingsnius. Tačiau šios programos vartotojo sąsaja nėra labai patogi, turi nemažai trūkumų. Taip pat joje negalima atlikti klaidų sekimo kontrolės. Bet produktas - nemokamas.

Apibendrinus galima pasakyti, kad visos trys nagrinėtos programos yra specializuotos, tam tikrai siaurai sričiai skirtos, puikiai išpildančios įmonės reikalavimus programos, kurių vienintelis trūkumas – integracijos su kitais procesais ar jų dalimis stygius. Tačiau, jei nėra vieno integruoto produkto, kuris leistų valdyti visą testavimo procesą ir bendradarbiauti šio proceso aktoriams su kitų procesų aktoriais, visas procesas tampa nenuoseklus, nekontroliuojamas, trumpiau tariant – neefektyvus. Visi išvardinti įrankiai tinka antrajam testavimo brandumo lygmeniui pagal TMM modelį.

Apžvelgus keletą specializuotų programų buvo prieita išvados, kad įmonėje vieno produkto kūrimui naudoti daug atskirų programų yra nepatogu. Taip neišvengiama klaidų, atsirandančių dėl produktų nesuderinamumo bei nepatogumo darbuotojams, dirbantiems prie projekto, atsirandančio dėl nestruktūrizuotos informacijos nesuderinamumo. Dėl to atsiranda poreikis turėti integruotą informaciją, apimančią viso programinės įrangos kūrimo proceso eigą. Vienas iš tokių realizuotų produktų yra Microsoft Team System.

***Microsoft Team System* – tai visos gyvavimo grandinės (*lifecycle*) valdymui, užduočių (*tasks*), klaidų (*bugs*), versijavimo kontroliavimui skirtas produktas** [15]. Jame yra galimybė aprašyti savo programinės įrangos kūrimo procesus, vesti projekto valdymą, valdyti programos kodo versijavimą bei atlikti testavimą. Visual Studio Team System galima sugeneruoti modulinius testus, atlikti internetinių aplikacijų testavimą ir vykdyti apkrovimo testus [11]. Tačiau *Team System* yra visiškai integruotas su *Visual Studio* projektais, todėl yra sunkiai pritaikomas kitiems projektams. Jis reikalauja daug žinių ir yra sunkiai suprantamas. Be to, produktas yra mokamas, netgi labai brangus. Šis produktas tinka trečiajam testavimo brandumo lygmeniui pagal TMM modelį.

Taigi reziumuojant panašių sprendimų analizę, pateikiama apibendrinanti lentelė:

Lentelė 1 Panašių sistemų analizė

Lyginimo kriterijai	„Mantis„	„Bugzilla“	„Test Link“	„Microsoft Team System“
Galimybė sudaryti testavimo atvejus	Nerealizuota	Nerealizuota	Realizuota	Realizuota
Galimybė sekti bei valdyti klaidas	Realizuota	Realizuota	Nerealizuota	Realizuota
Galimybė generuoti diagramas apie produkto kokybę (klaidų skaičių bei svarbumą)	Nerealizuota	Nerealizuota	Nerealizuota	Realizuota
Galimybė kaupti užregistruotų klaidų statistiką	Realizuota	Realizuota	Nerealizuota	Realizuota
Vartotojo sąsajos patogumas, lankstumas	Patogi	Patogi	Turi trūkumų	Reikalauja daug žinių ir yra sunkiai suprantamas
Produkto kainos dydis	Nemokamas	Nemokamas	Nemokamas	Mokamas, netgi labai brangus
TMM brandumo lygmuo	2	2	2	3

Atlikus panašių sistemų analizę, galima teigti, jog norint testavimo procesą integruoti į programinės įrangos kūrimo procesų visumą bei visą gyvavimo ciklą, taip pat norint pasiekti trečiąjį TMM brandumo lygmenį, tektų naudotis keliais įrankiais vienu metu arba naudotis galingu bei brangiu produktu, skirtu viso programinės įrangos kūrimo proceso valdymui.

2.7. Galimų įgyvendinimo priemonių variantų analizė

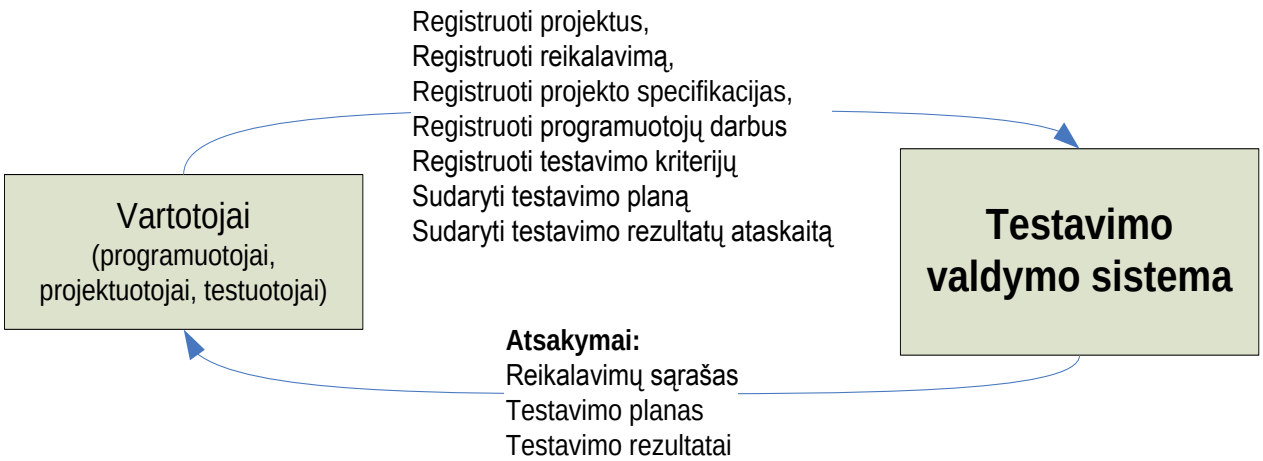
Atlikus programinės įrangos testavimo proceso analizę bei jau egzistuojančių sistemų, skirtų vienaip ar kitaip valdyti šį procesą, apžvalgą, prieita išvados, jog testavimo procesui integruoti į programinės įrangos kūrimo procesų visumą bei automatizuotai valdyti šį procesą, galima:

- Naudotis net keliais jau egzistuojančiais įrankiais, padedančiais valdyti testavimo procesą. Tačiau tai nėra patogu, kadangi reikia persijungti nuo vienos sistemos prie kitos.
- Naudotis įrankiais, integruojančiais testavimo procesą į bendrą programinės įrangos kūrimo visumą, tačiau apimančiais žymiai platesnius procesus, pernelyg galingais bei brangiais.
- Sukurti specializuotą sistemą, padedančią valdyti bei automatizuoti programinės įrangos testavimo procesą bei integruoti jį į visą programinės įrangos kūrimo gyvavimo ciklą.

Pirmasis variantas atmetamas, nes jis yra nepatogus. Antrasis variantas automatiškai atkrinta dėl brangumo, taip pat dėl to, kad nėra tikslo naudoti sistemos, kuri yra per daug galinga vien testavimo procesui valdyti, kuri yra skirta visam programinės įrangos kūrimo procesui vykdyti. Atmetus pirmuosius du variantus, apsistojama ties trečiuoju – specializuotos sistemos, skirtos programinės įrangos testavimo valdymui automatizuoti.

2.8. Siekiamos sistemos apibrėžimas

Pav. 9 vaizduojama realizuojamos sistemos kontekstinė diagrama, atskleidžianti pagrindines realizuojamos klaidų valdymo sistemos funkcijas bei rezultatus:



Pav. 9 Sistemos kontekstinė diagrama

2.9. Darbo tikslas ir siekiami privalumai

Išanalizavus alternatyvius produktus, paaiškėjo, kad pirmieji neapima viso testavimo proceso, o paskutinis *Microsoft* produktas yra be galo didelis, plačiai apimantis visą programinės įrangos kūrimo procesą ir yra labai brangus. Taigi yra poreikis produktui, kuris apimtų daugelį naudojamų programinės įrangos testavimo proceso etapų ir leistų integraliai testuoti programinę įrangą.

PĮ kūrimo procesą pagerintų sistema, apjungianti projekto valdymo, analizės, projektavimo, realizacijos, testavimo bei dokumentavimo etapus.

Šiame modelyje kiekvieno etapo dalyvis turi savo individualias užduotis, kurios turi sąryšį su kitais etapais. Projektų vadovas gali pradėti naujus projektus, suskirstyti užduotis visiems darbuotojams, valdyti PĮ kūrimo procesą. Analitikai gali specifikuoti reikalavimus, kuriuos projektuoja projektuotojai. Pagal suprojektuotus modelius programuotojai gali projektą realizuoti. Visa tai testuoja testuotojas, kuris iš reikalavimų gali sudaryti testavimo atvejus, o ištestavęs gali pateikti ataskaitas apie projekto kūrimo eigą ir jo kokybę. Iš reikalavimų, ataskaitų ir rezultatų dokumentuotojai gali dokumentuoti produktą. Tačiau ši sistemos integracija yra paliekama kaip išplėtimo galimybė, o realizuojamas testavimo etapas.

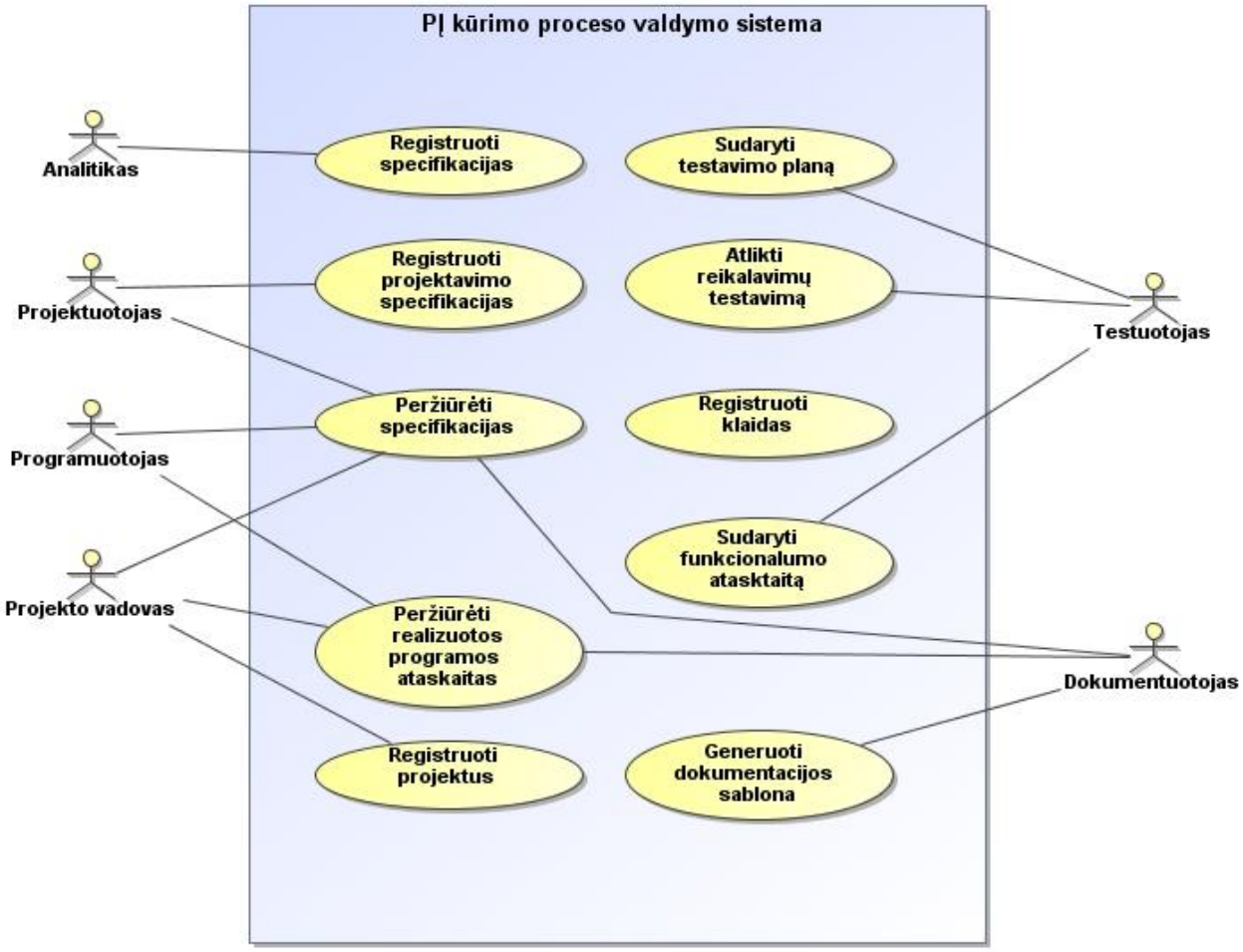
Projektavimo ir realizacijos dalyje turėtų būti registruojamos užduotys, iš kurių būtų galima sudaryti testavimo atvejus. Kiekviena užduotis gali būti susieta su vienu ar daugiau testavimo atvejų, arba gali būti parašomi tokie testavimo atvejai, kurie apimtų kelias užduotis. Pagal testavimo atvejus gali būti registruojamos klaidos. Norint ištaisyti tas klaidas, sukuriamos naujos užduotys, pagal kurias kuriami nauji testavimo atvejai. Klaidos gali iš naujo įtakoti reikalavimus tokiu atveju, jeigu jie yra nekorektiški. Toks modelis gali būti naudojamas ir projekto palaikymui.

Lyginant siūlomą sprendimą su analizuotais, galima pastebėti, kad *Mantis* ir *Bugzilla* apima tik klaidų registravimą, o *Test link* – tik testavimo atvejų sudarymą.

Siekiamas sistemos privalumas turi būti integralumas, leidžiantis tiksliai, betarpiškai ir vieningai dirbti siekiant rezultato. Taip suprojektuota sistema leistų visiškai stebėti sąsajas tarp visų etapų: t.y., galima būtų matyti klaidų įtaką darbams, planuoti kaštus. Galima tokios sistemos rinka yra nedidelės programinės įrangos kūrimo ir plėtojimo įmonės, savo veiklos eigoje naudojančios testavimo procesą. Naudojantis šia sistema, būtų galima dirbti bet kokioje aplinkoje, priešingai negu *Microsoft Team System*, kuriai yra reikalinga *Visual Studio* aplinka.

2.10. Kompiuterizuojamos sistemos funkcijos

Kompiuterizuojamos sistemos uždavinių modelis (panaudojimo atvejų modelis) atskleidžia klaidų valdymo sistemos kompiuterizuojamas funkcijas (pav. 10). Projektuotojas registruoja ir peržiūri specifikacijas, programuotojas bei dokumentuotojas peržiūri specifikacijas bei realizuotos programos rezultatus, o testuotojas pagal gautas specifikacijas sudaro testavimo planą, atlieka reikalavimų testavimą bei sudaro funkcionalumo ataskaitą:



Pav. 10 Kompiuterizuojamos sistemos uždavinių modelis

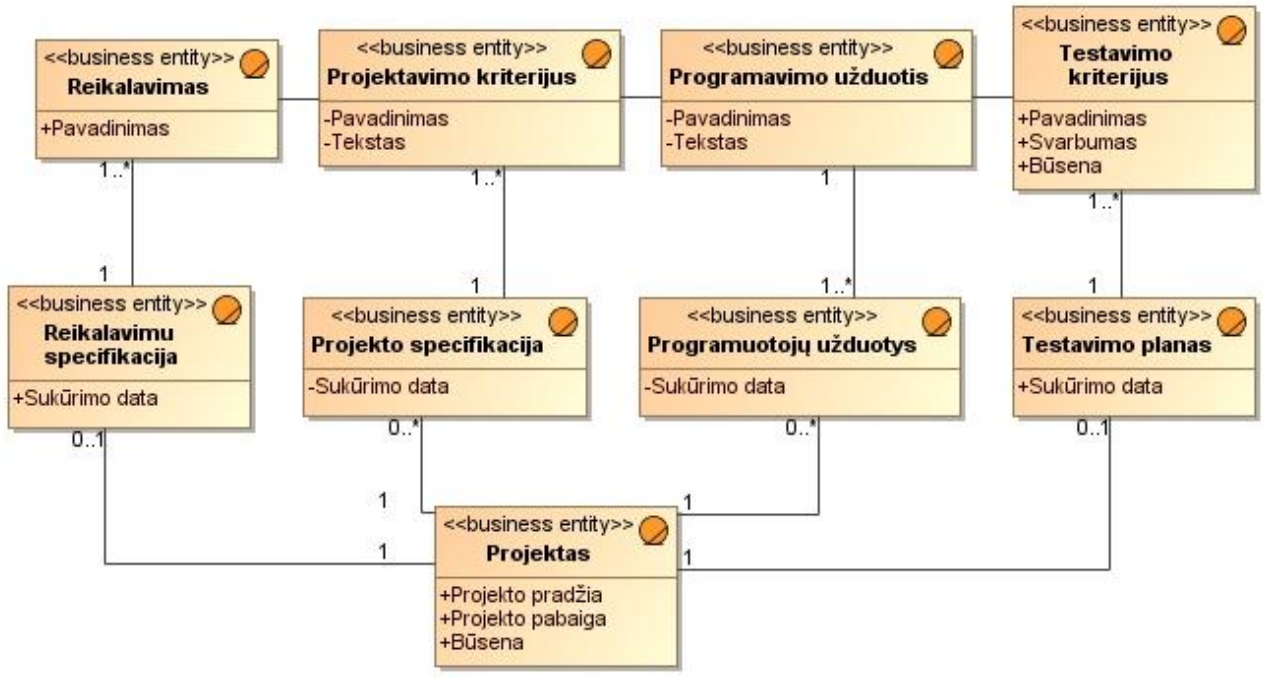
2.11. Reikalavimai duomenims

Projekto vykdymo metu yra sudaroma keletas specifikacijų:

- reikalavimų specifikacija,
- projekto specifikacija,
- programavimo užduočių specifikacija,
- testavimo planai.

Šios specifikacijos turi būti saugomos sistemoje, todėl joms sudaromos klasės.

Sistemos reikalavimus duomenims apibūdina duomenų klasių diagrama, pateikta Pav.11.



Pav. 11 Duomenų klasių diagrama

2.12. Nefunkciniai reikalavimai ir apribojimai

Šiame skyrelyje bus aptariami nefunkciniai sistemos reikalavimai bei apribojimai, bus aptariama netiesioginės sistemos funkcijos. Šioje lentelėje (Lentelė 2) pateikti kuriamos sistemos nefunkciniai reikalavimai ir jų apribojimai:

Lentelė 2 Nefunkciniai reikalavimai ir apribojimai

Nefunkciniai reikalavimai	Apribojimai
Suprantamumas (vartotojo pastangų prasme) (angl. *Understandability*)	Sistemos viena pagrindinių savybių – patogumas vartotojui. Sistema turi pasižymėti suprantama grafine sąsaja, kad vartotojas gebėtų išmokti ja dirbti be specialistų pagalbos per vieną savaitę.

Nefunkciniai reikalavimai	Apribojimai
Realizacija	Sistema turi būti lengvai pritaikoma įvairiems programinės įrangos kūrimo gyvavimo ciklams (rekomenduojamas „V“ modelis, tačiau gali būti ir kiti), programavimo technologijoms bei kalboms (neturi būti skirtumo, kokia programavimo technologija ar kalba pasirinkta), taip pat įvairioms testavimo metodikoms (tokioms kaip vienetų testavimas, integracijos testavimas, sistemos testavimas, vartotojo sutikimas ir t.t.)
Praplečiamumas, integralumas	Turi būti galimybė programuotojams praplėsti sistemą neperprogramuojant visos sistemos, o tik prijungiant prie jos kitus modulius.

2.13. Rizikos faktorių analizė

Sistemos rizikos faktoriai:

- Netiksliai apibrėžtas programinės įrangos kūrimo proceso gyvavimo ciklas.
- Chaotiškas, neapibrėžtas programinės įrangos testavimo procesas.
- Programinei įrangai testuoti naudojamas automatinis testavimas.

Esant šiems rizikos faktoriams, sistemos naudojimas gali neatitikti ar ne pilnai atitikti jai keliamų reikalavimų bei nepateisinti vartotojo lūkesčių.

2.14. Rezultato kokybės kriterijai

Pagrindinis rezultato kokybės kriterijus yra Testavimo brandumo modelio brandumo lygmuo. Atlikus programinės įrangos testavimo proceso analizę bei iškėlus būsimos sistemos reikalavimus, nuspręsta, jog būsima sistema turėtų apeliuoti mažiausiai į trečiąjį programinės įrangos testavimo brandumo modelio brandumo lygmenį.

TMM trečiojo lygmens (integracijos) savybės: testavimas integruotas į PĮ gyvavimo ciklą, testo tikslai yra nustatomi atsižvelgiant į reikalavimus paremtus vartotojo poreikiais ir jais remiamasi kuriant testinius atvejus. Testavimas pripažįstamas profesionalia veikla.

TMM ketvirtojo lygmens (valdymo ir matavimo) savybės: testavimas yra matuojamas ir vertinamas procesas. Peržiūra visose PĮ kūrimo fazėse yra pripažįstama kaip testavimo ir kokybės kontrolės veikla. Testuojami PĮ produktų tokie kokybės kriterijai, kaip patikimumas, panaudojamumas, palaikomumas. Testiniai atvejai iš visų projektų yra surenkami į duomenų bazę bei pakartotinai panaudojami. Defektai yra registruojami ir jiems priskiriamas svarbumo lygis.

2.15. Analizės išvados

Darbo metu buvo išanalizuota programinės įrangos kūrimo ir testavimo proceso specifika ir prieita tokių išvadų:

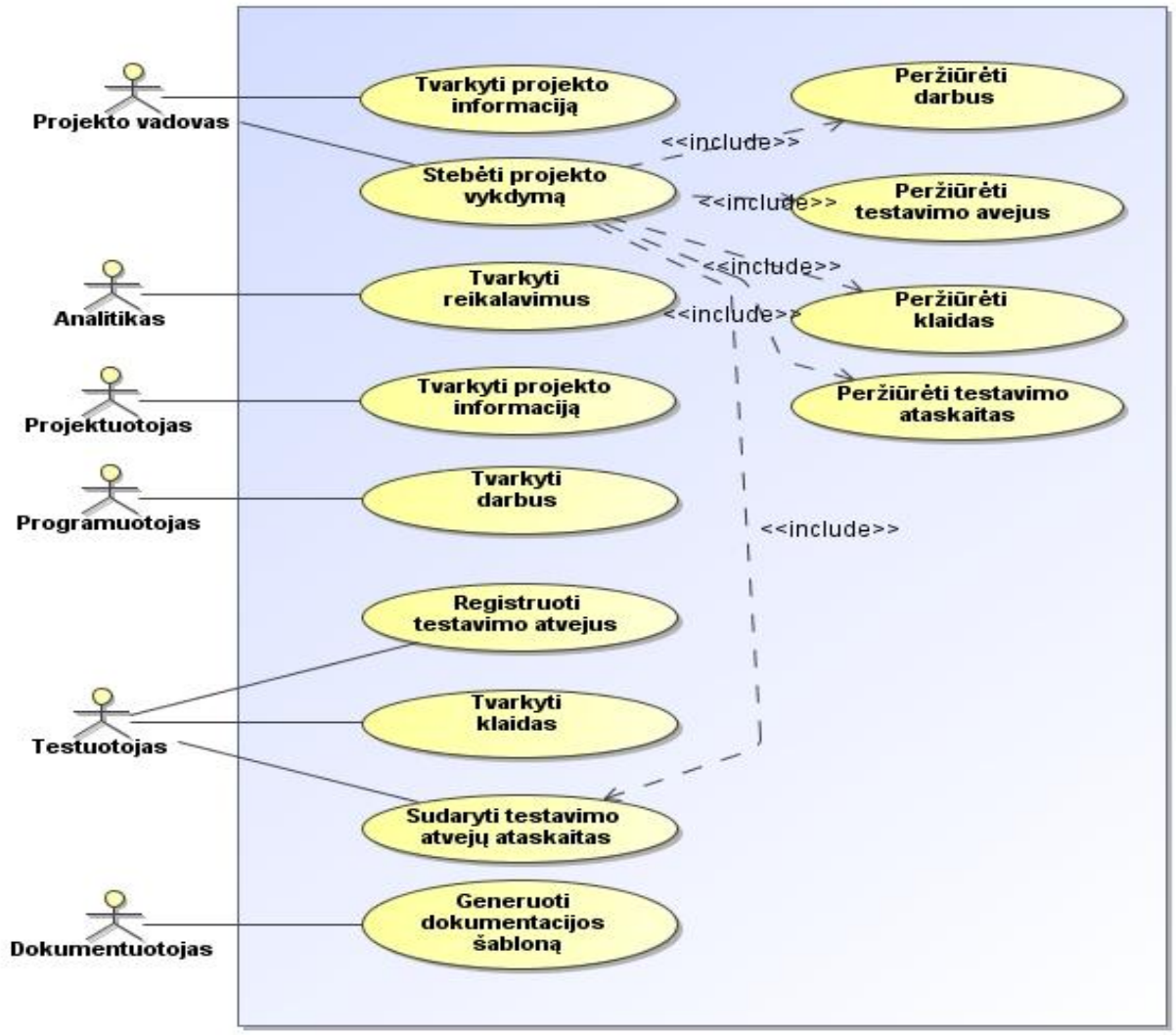
1. Egzistuojančios klaidų valdymo sistemos neužtikrina visų programinės įrangos kūrimo procesų integralumo, nes dažniausiai apima tik testavimo atvejus ir jų rezultatus, tačiau nesusieja jų su reikalavimais ir projekto darbais.
2. Norint užtikrinti efektyvų testavimo ir viso programinės įrangos kūrimo procesą yra poreikis sukurti sistemą, apjungiančią testavimo procesą su kitais programinės įrangos kūrimo procesais į vieną visumą.
3. Norint testavimo procesą integruoti į visą programinės įrangos kūrimo procesą, reikia sukurti sistemą, apjungiančią testavimo atvejų ryšius su reikalavimais ir PĮ kūrimo metu atliekamais darbais.

3. Programinės įrangos testavimo valdymo automatizavimo sistemos reikalavimų specifikacija ir analizė

Šiame skyriuje bus sudaromi reikalavimai kuriamai programinės įrangos testavimo valdymo automatizavimo sistemai. Bus nustatomi panaudojimo atvejai, sudaromos klasių diagramos, sekų diagramomis pavaizduojamas vartotojo bendravimas su sistema. Proceso projektavimui bei vaizdavimui bus naudojamas RUP modelis bei UML projektavimo įrankis „Magic Draw UML“.

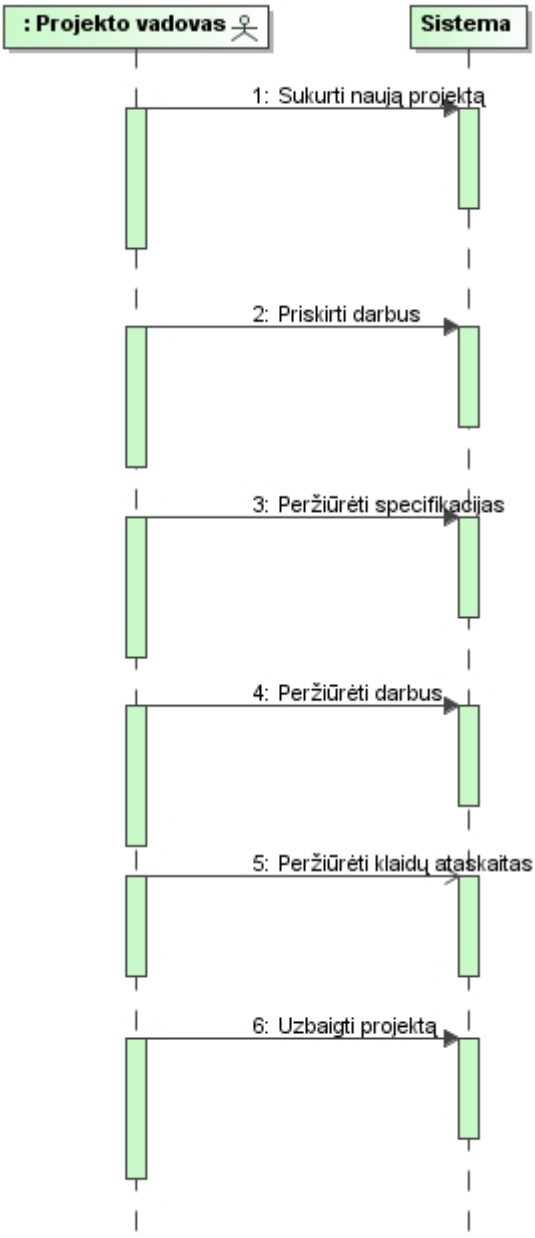
3.1. Reikalavimų specifikacija

Vykdant projektą, reikia kad projektų vadovas įvestų informaciją apie projektą, analitikas reikalavimus, testuotojas klaidas. Visi šie reikalavimai atvaizduojami panaudojimo atvejų diagramoje:



Pav. 12 Kompiuterizuojamos veiklos panaudojimo atvejai

Visa vartotojo ir sistemos veiksmų seka pateikta sekančiose reikalavimų etapo sekų diagramose:

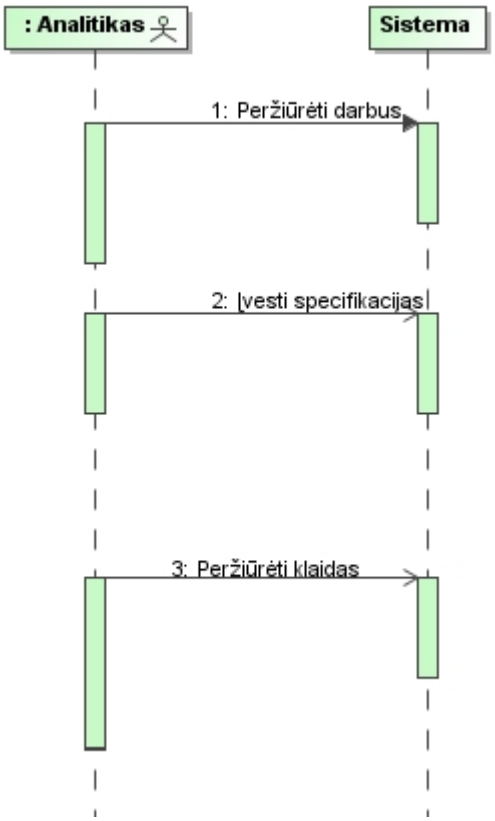


Pav. 13 Projekto vadovo ir sistemos sekų diagrama

Projekto vadovas su sistema galės daryti tokius veiksmus:

- Sukurti naują projektą;
- Priskirti darbus;
- Peržiūrėti specifikacijas;
- Peržiūrėti darbus;
- Peržiūrėti klaidų ataskaitas;

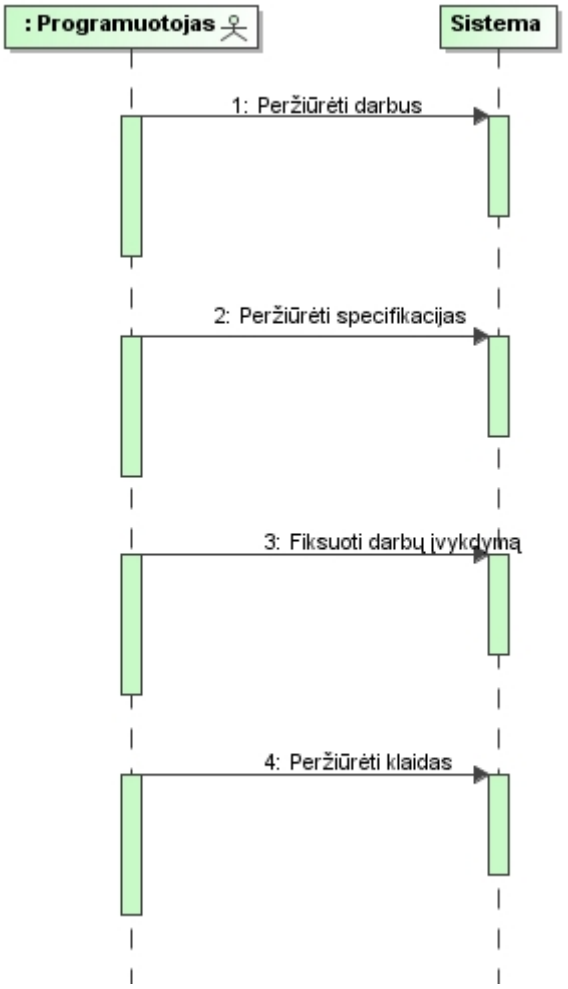
- Užbaigti projektą.



Pav. 14 Analitiko ir sistemos sekų diagrama

Analitikas su sistema galės daryti tokius veiksmus:

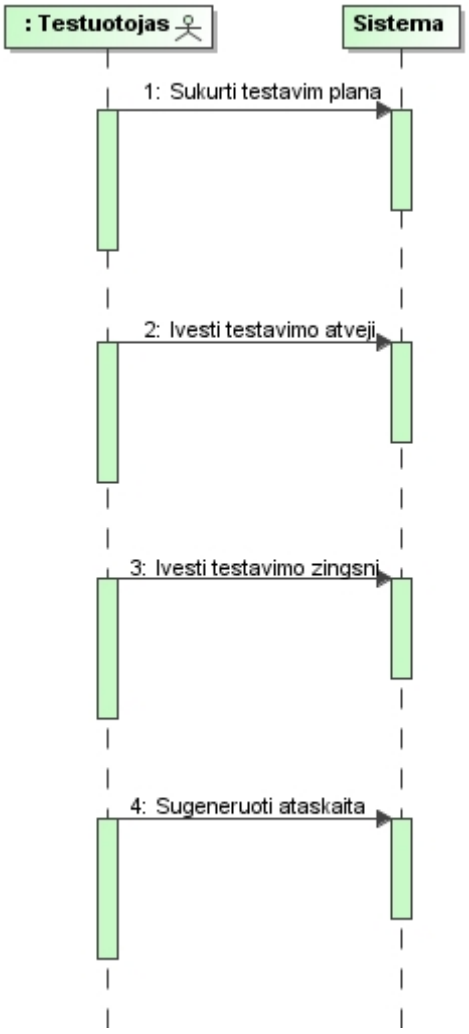
- Peržiūrėti darbus;
- Įvesti specifikacijas;
- Peržiūrėti klaidas.



Pav. 15 Programuotojo ir sistemos sekų diagrama

Programuotojas su sistema galės daryti tokius veiksmus:

- Peržiūrėti darbus;
- Peržiūrėti specifikacijas;
- Fiksuoti darbų vykdymą;
- Peržiūrėti klaidas.



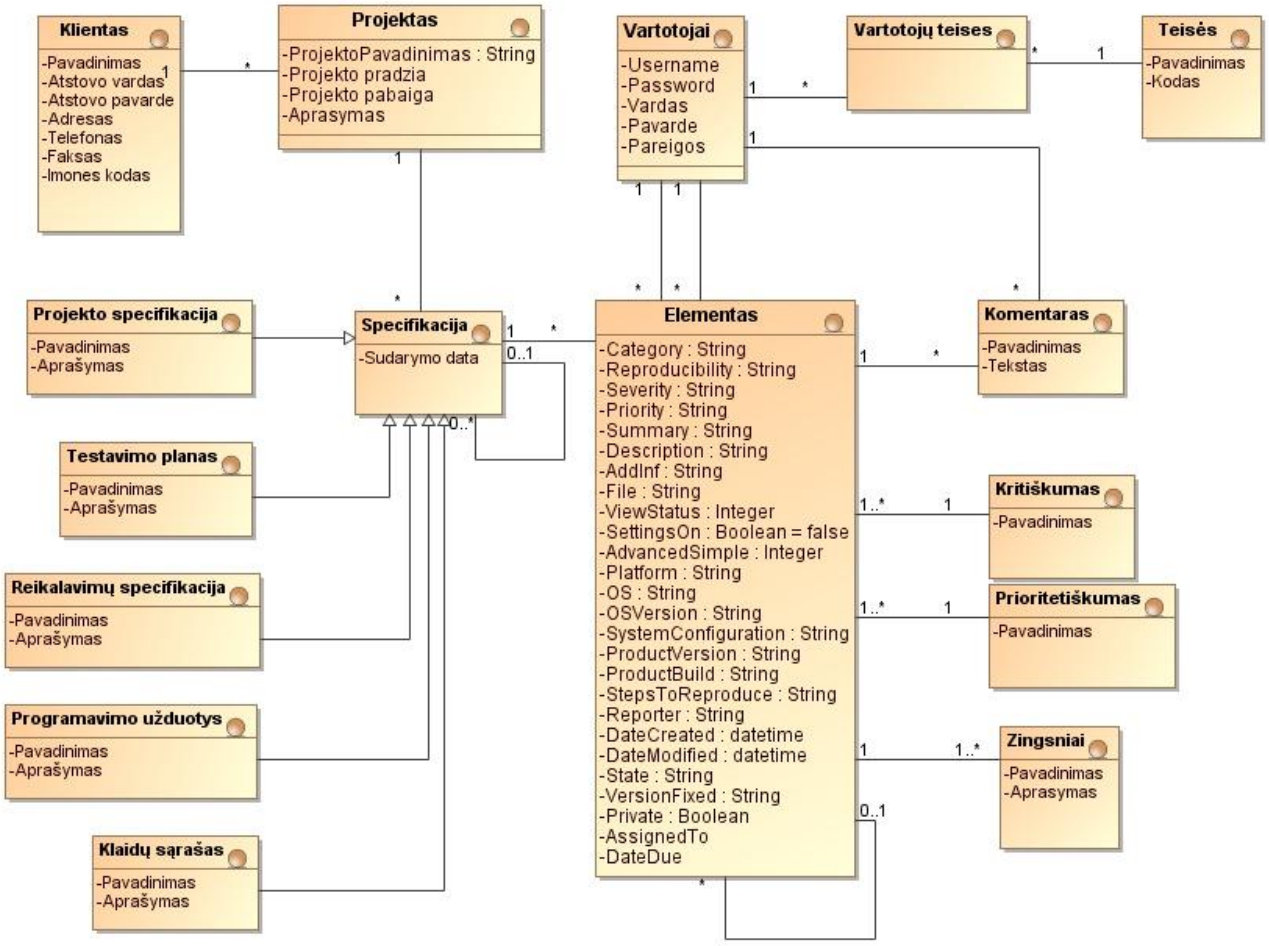
Pav. 16 Testuotojo ir sistemos sekų diagrama

Testuotojas su sistema galės daryti tokius veiksmus:

- Sukurti testavimo planą;
- Įvesti testavimo atvejį;
- Įvesti testavimo žingsnį;
- Sugeneruoti ataskaitą.

3.2. Dalykinės srities modelis

Koncepcinė klasių diagrama (Pav. 17) vaizduoja sistemos klases.



Pav. 17 Koncepcinė klasių diagrama

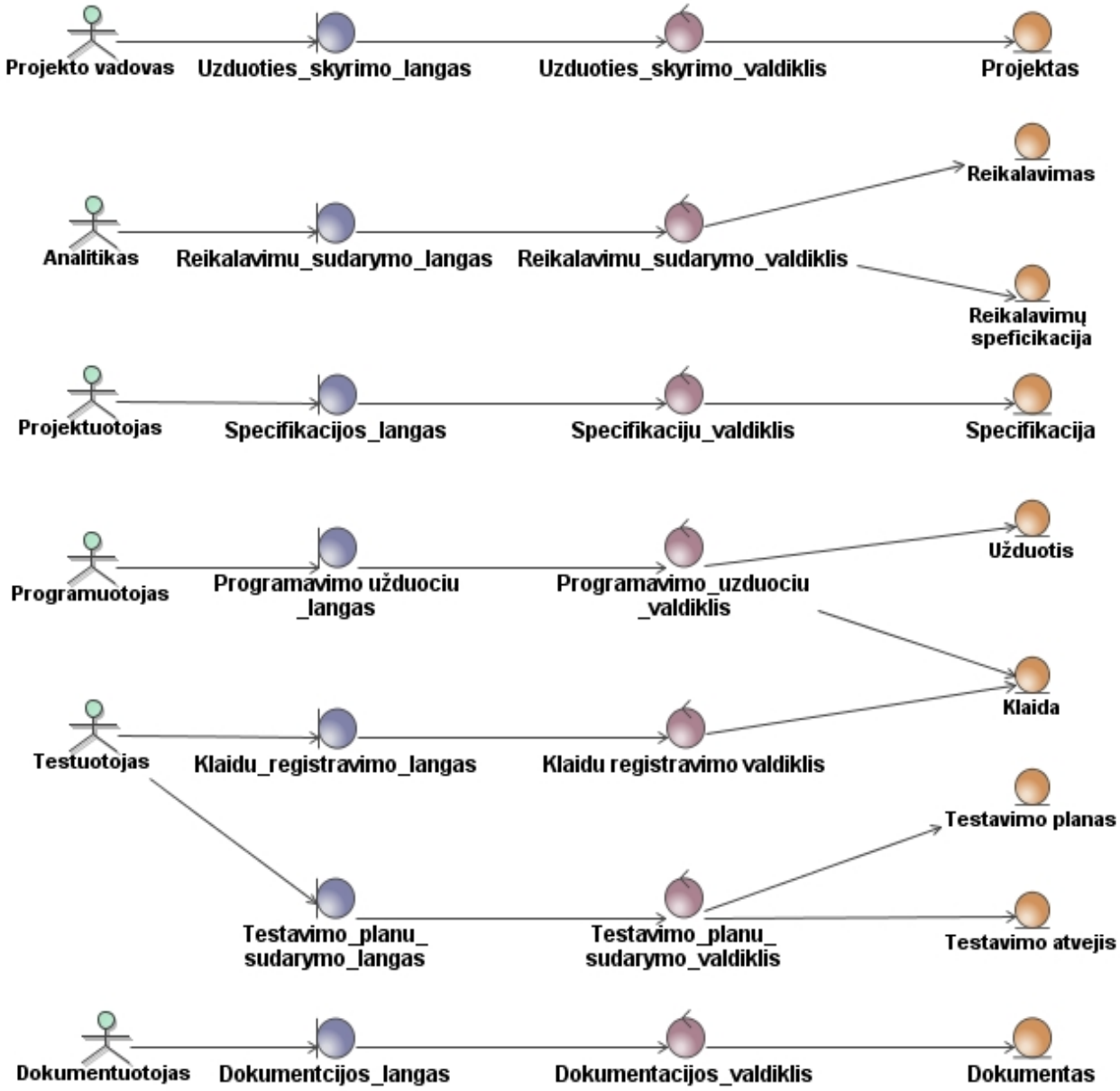
Programinės įrangos kūrimo procese, priklausomai nuo pasirinkto programinės įrangos gyvavimo ciklo, kiekviename etape yra sudaromos specifikacijos. Norint sukurti universalią sistemą, kuri galėtų būti pritaikoma esant bet kokiam gyvavimo ciklui, visos specifikacijos ir jų elementai bus saugomi vienoje klasėje. Specifikacijos gali būti tokios:

- Reikalavimų specifikacija;
- Projektavimo specifikacija;
- Programavimo specifikacijos;
- Testavimo planai;
- Klaidų sąrašai;

Taip pat priklausomai nuo gyvavimo ciklo, gali būti sukuriami nauji specifikacijų tipai. Todėl esant tokiai duomenų struktūrai, sistemai suteikiamos didesnes lankstumo, plečiamumo ir pritaikomumo galimybės.

3.3. Reikalavimų analizė

Analizės diagramoje vaizduojamas sistemos objektų tarpusavio sąveikos (Pav. 18).



Pav. 18 Analizės diagrama

Šioje diagramoje vaizduojama, kaip aktorius sąveikauja su lango klasėmis, kurios aptarnauja vartotojo užklausas. Langas kreipiasi į modulio valdiklių klases, kurios dirba su esybių klasėmis.

4. Programinės įrangos kūrimo ir testavimo valdymo sistemos projektas

4.1. Sprendimo pagrindimas ir esmės išdėstymas

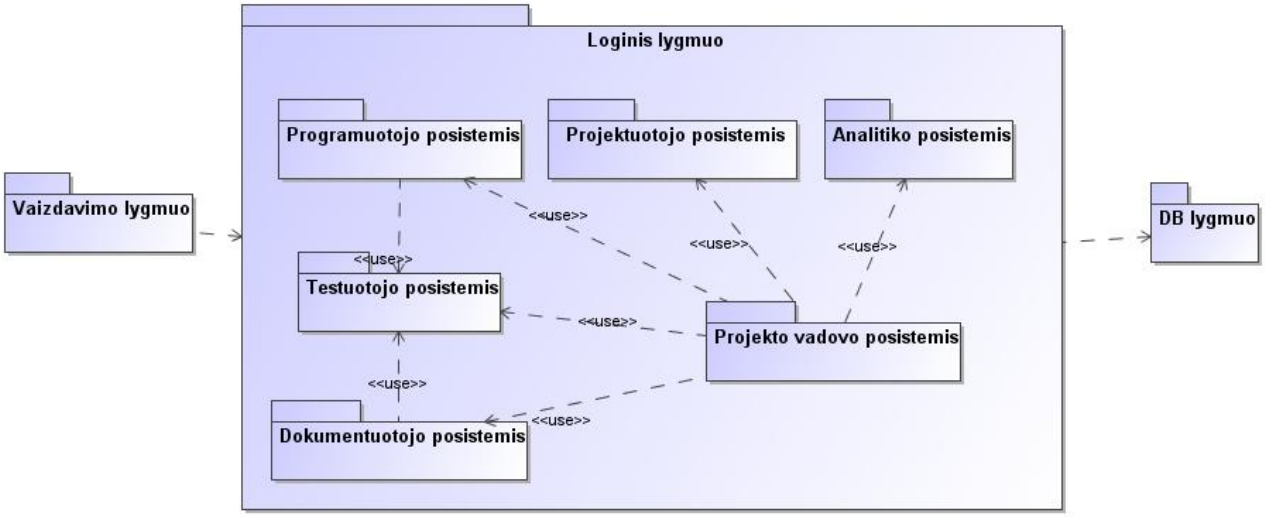
Visos darbo analizės metu buvo keliami reikalavimai programinės įrangos testavimo proceso valdymui automatizuoti. Darbo eigoje susiformavo būsimo sprendimo koncepcija bei reikalavimai realizuojamos sistemos funkcionalumui. Pagrindiniai reikalavimai:

- Integruoti testavimo procesą į programinės įrangos kūrimo procesų visumą, kad tai taptų neatsiejama programinės įrangos gyvavimo ciklo dalimi.
- Automatizuoti testavimo proceso valdymą, susiejant informaciją iš skirtingų programinės įrangos kūrimo etapų ir padarant ją lengvai pasiekiamą.

4.2. Sistemos architektūra – statinės struktūros modelis

4.2.1. Loginė visos sistemos architektūra

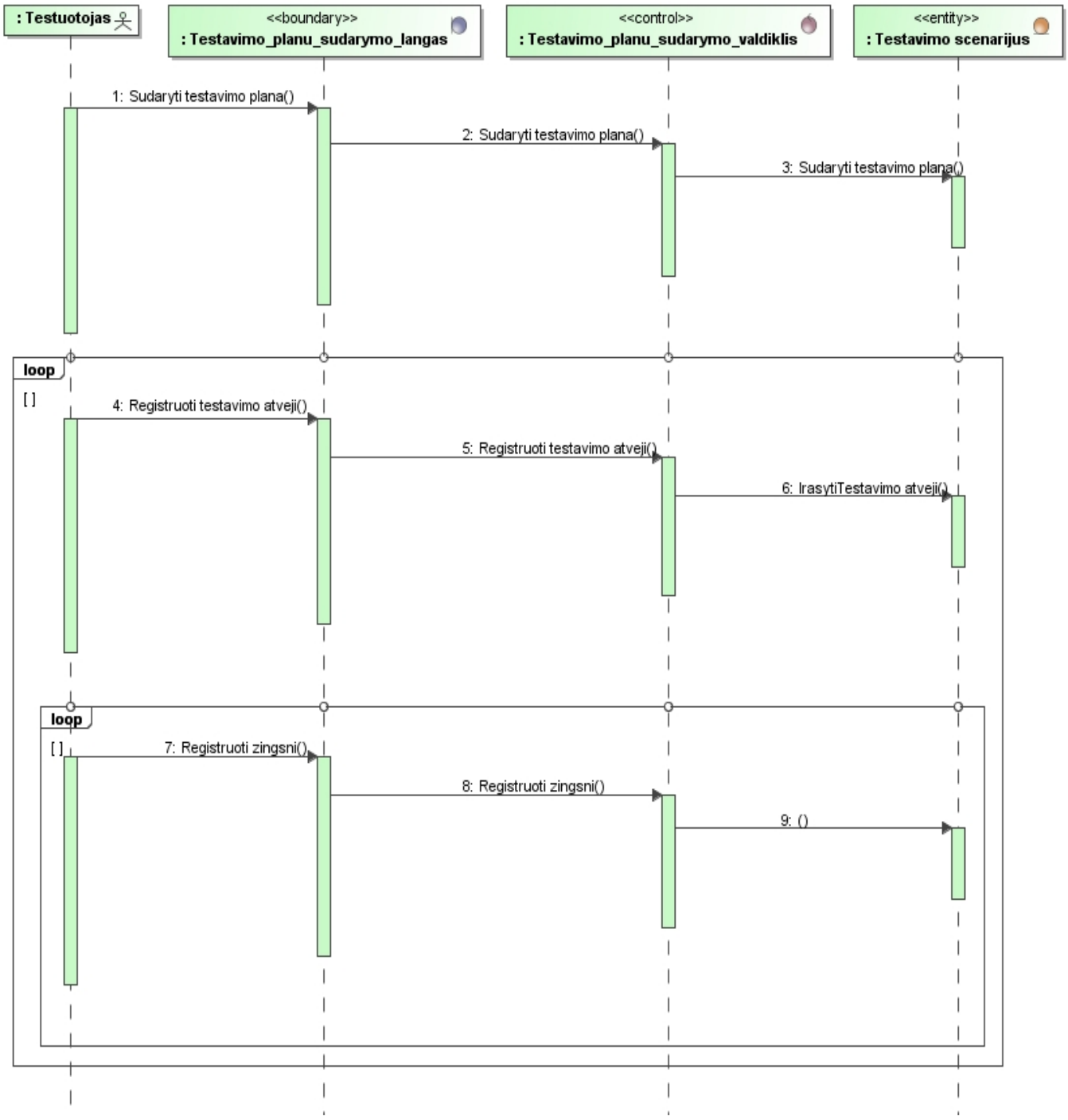
Sistema sudaryta iš trijų loginių lygmenų. Loginis lygmuo sudarytas iš posistemių, kurie atlieka tam tikrą visos sistemos apimties dalį. Loginė trijų lygių sistemos architektūra pateikta sekančiame paveiksle (Pav. 19).



Pav. 19 Sistemos loginė architektūra

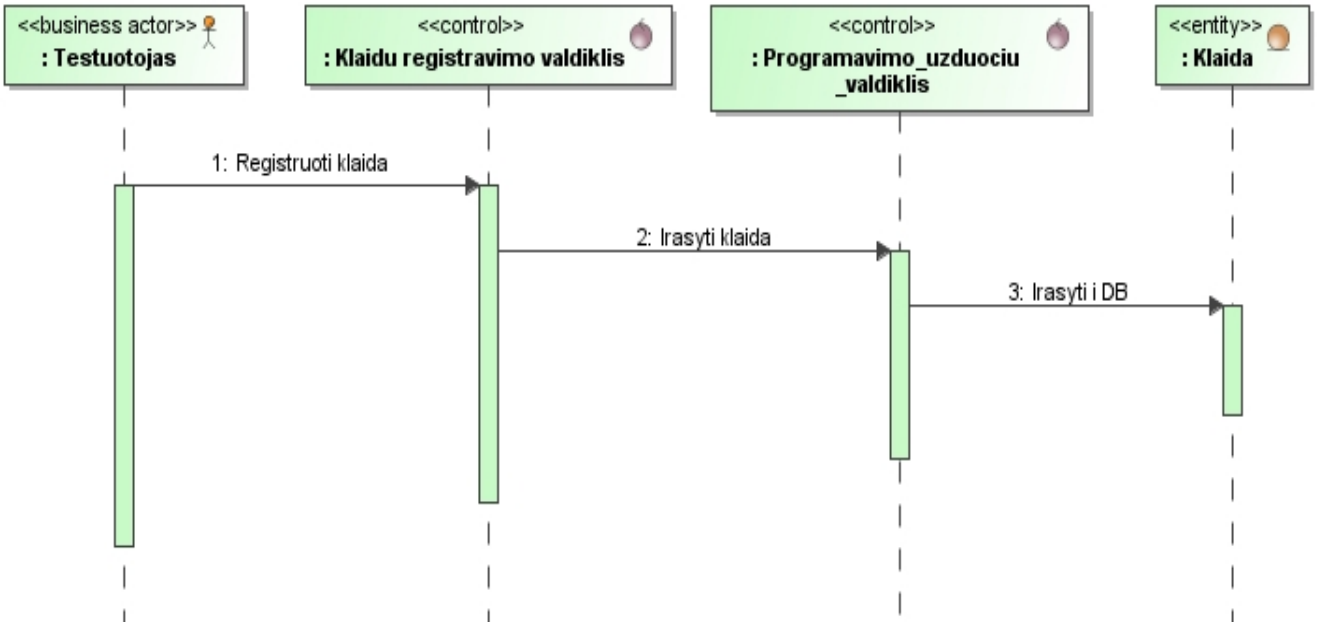
4.3. Sistemos elgsenos modelis

Šiame skyrelyje pateiktos testuotojo sekų diagramos, vaizduojančios veiksmų eigą tarp objektų:



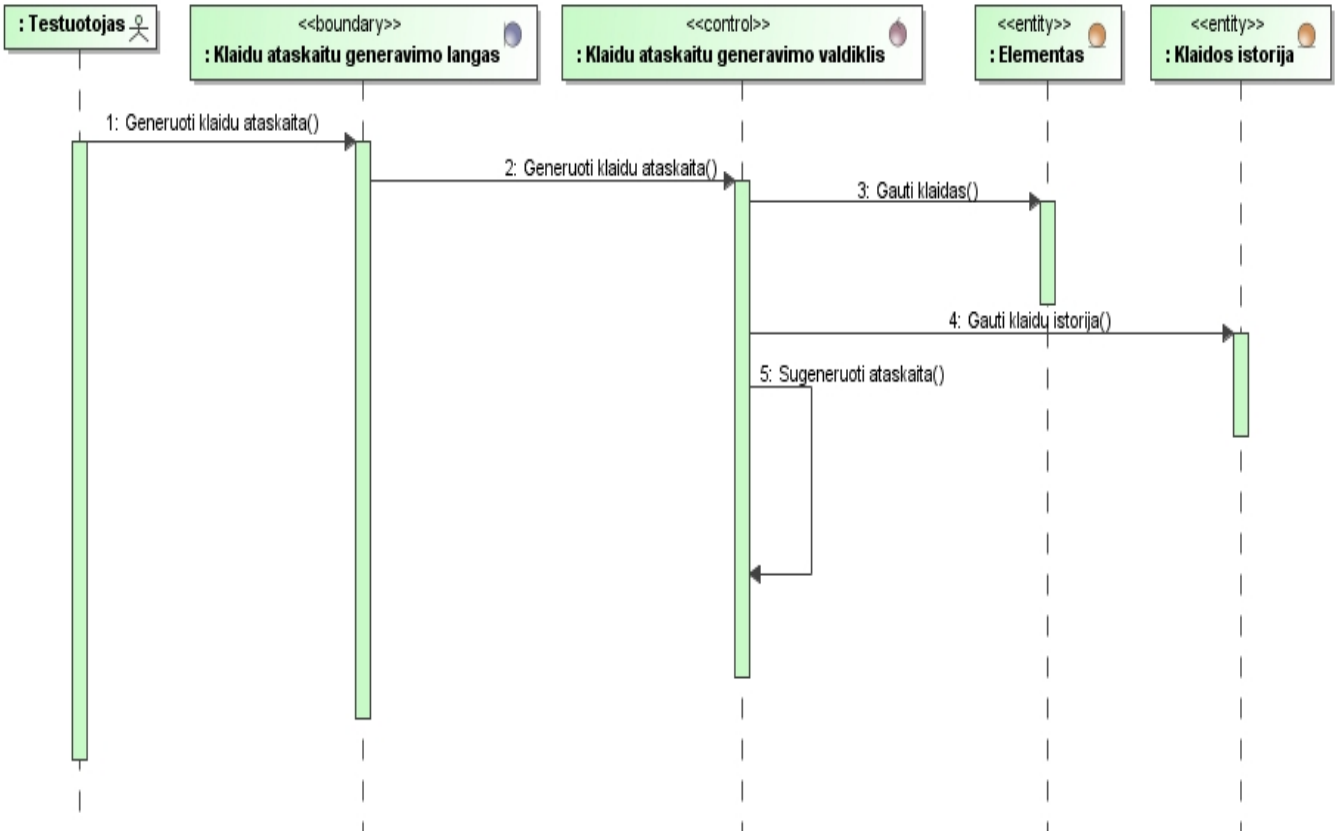
Pav. 20 Testuotojo sekų diagrama – Registruoti testavimo atvejus

Šioje diagramoje (Pav. 20) vaizduojama kaip testuotojas sudaro testavimo planą, testavimo atvejus bei testavimo žingsnius. Pildydamas testavimo planą, testuotojas turi užpildyti testavimo atvejus bei suvesti jų testavimo žingsnius.



Pav. 21 Testuotojo sekų diagrama – Registruoti klaidas

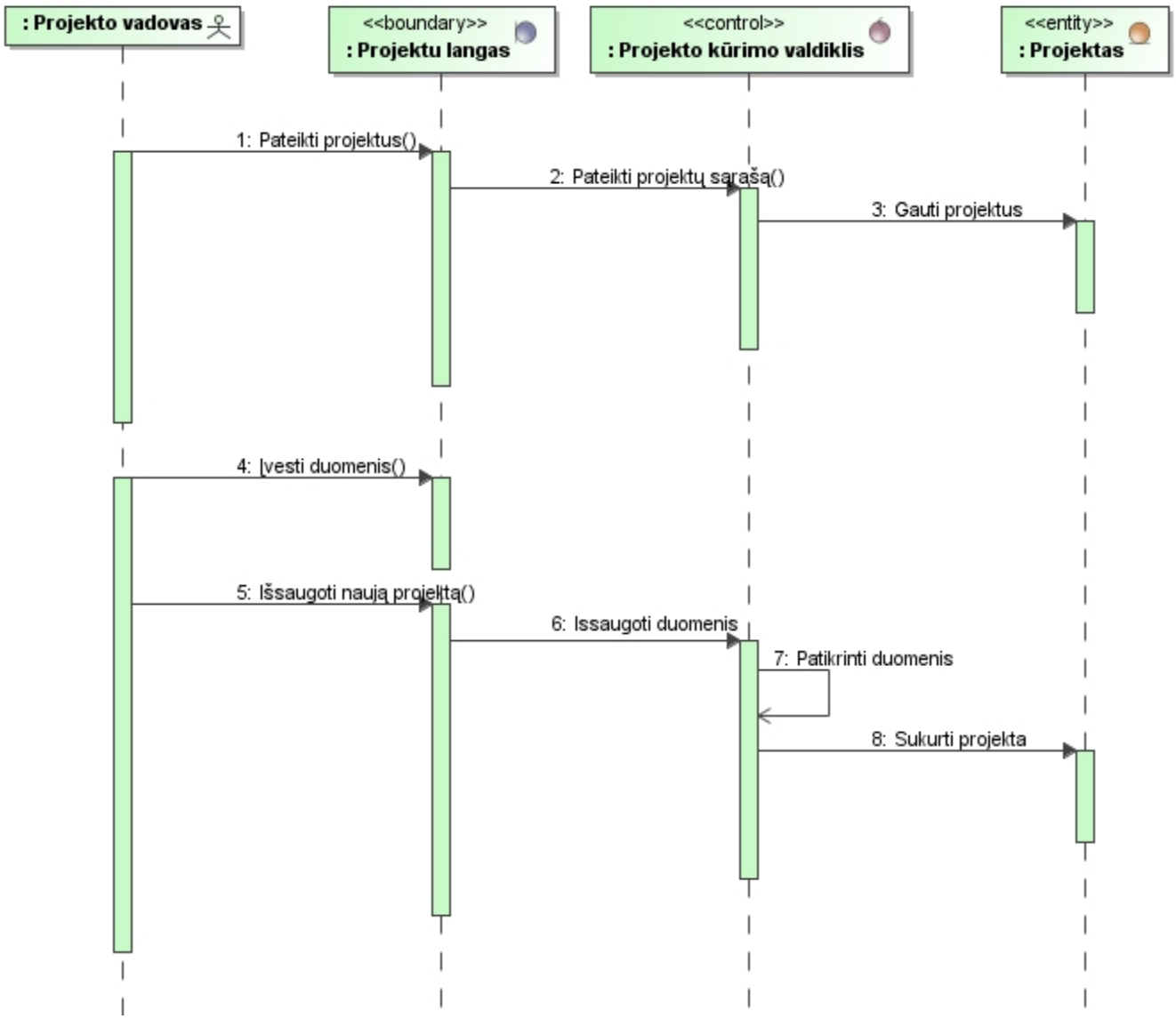
Ši diagrama (Pav. 21) vaizduoja testuotojo klaidų registravimą.



Pav. 22 Testuotojo sekų diagrama– Generuoti klaidų ataskaitas

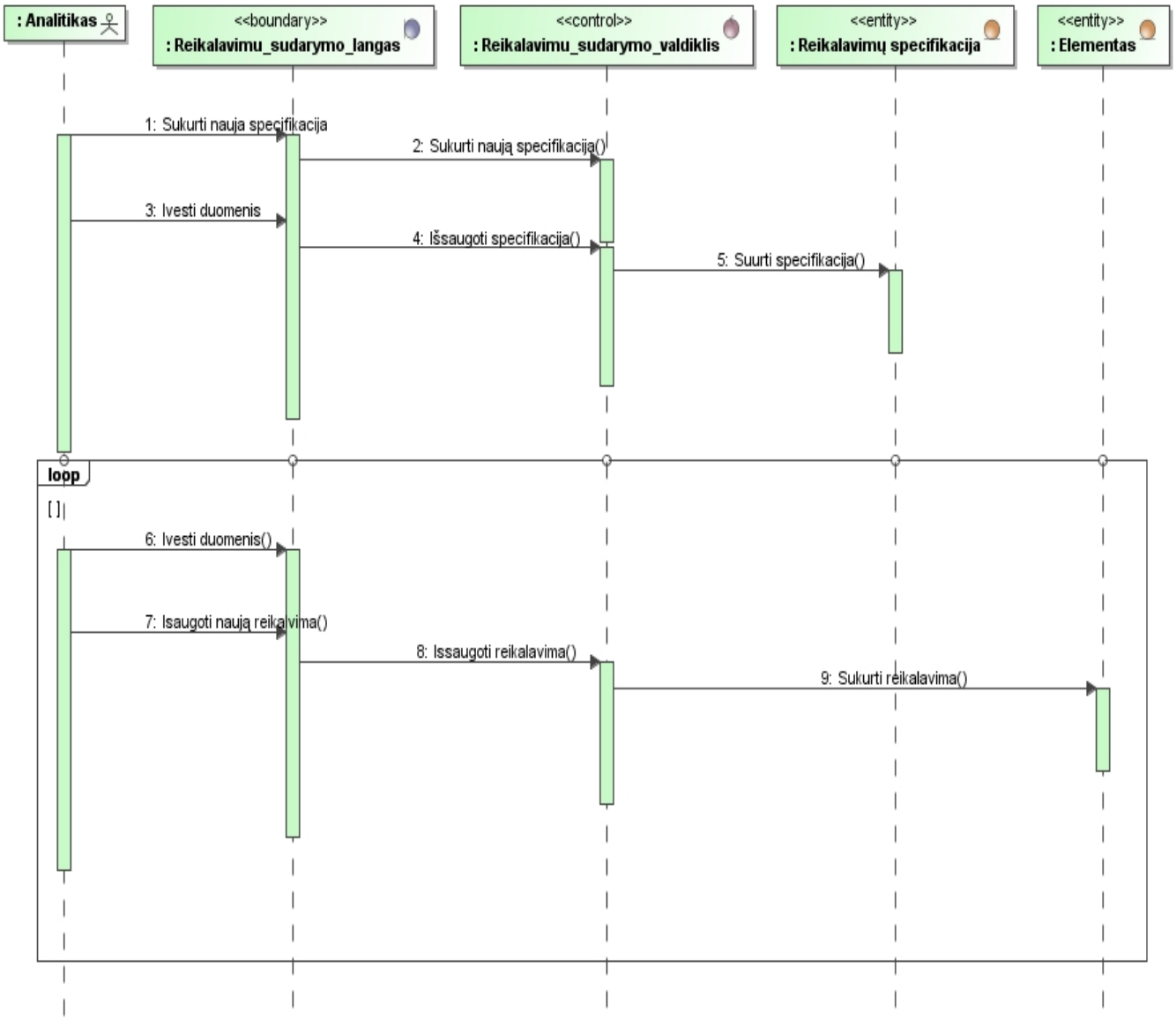
Ši diagrama (Pav. 22) parodo kaip testuotojas gali generuoti klaidų ataskaitas pagal klaidų istoriją.

Toliau pateikiamos sekų diagramos yra kitų sistemos aktorių: projekto vadovo, analitiko bei programuotojo bendravimo su sistema sekų diagramos:



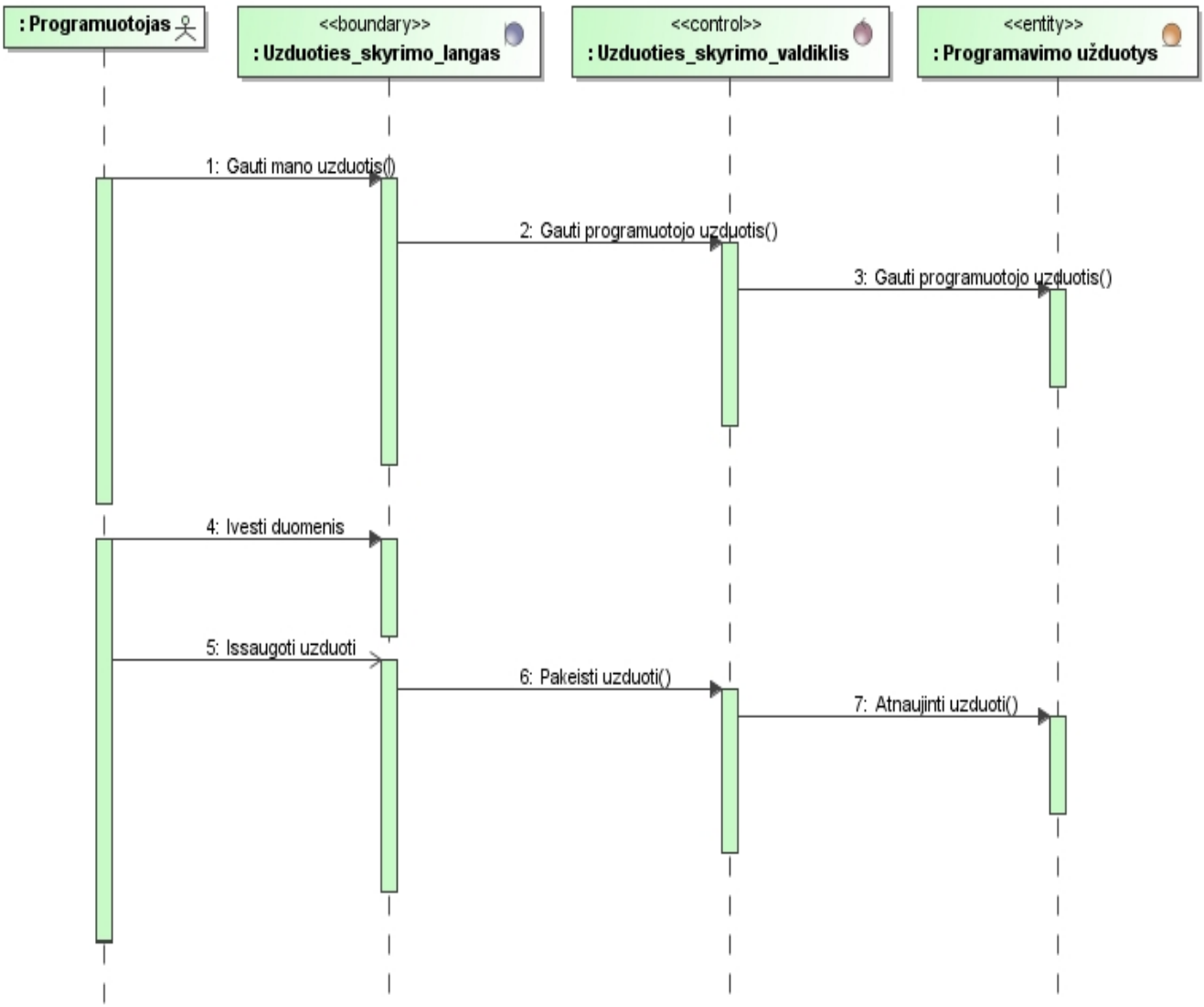
Pav. 23 Projekto vadovo posistemio sekų diagrama

Šioje diagramoje (Pav. 23) vaizduojama, kaip projektų vadovas gali teikti bei išsaugoti projektus, įvesti duomenis.



Pav. 24 Analitiko posistemio sekų diagrama

Analitiko posistemio sekų diagramoje (Pav. 24) vaizduojama, kaip analitikas gali sukurti naują specifikaciją, įvesti duomenis bei išsaugoti reikalavimus. Specifikacijoje gali būti saugoma daug reikalavimų įrašų.

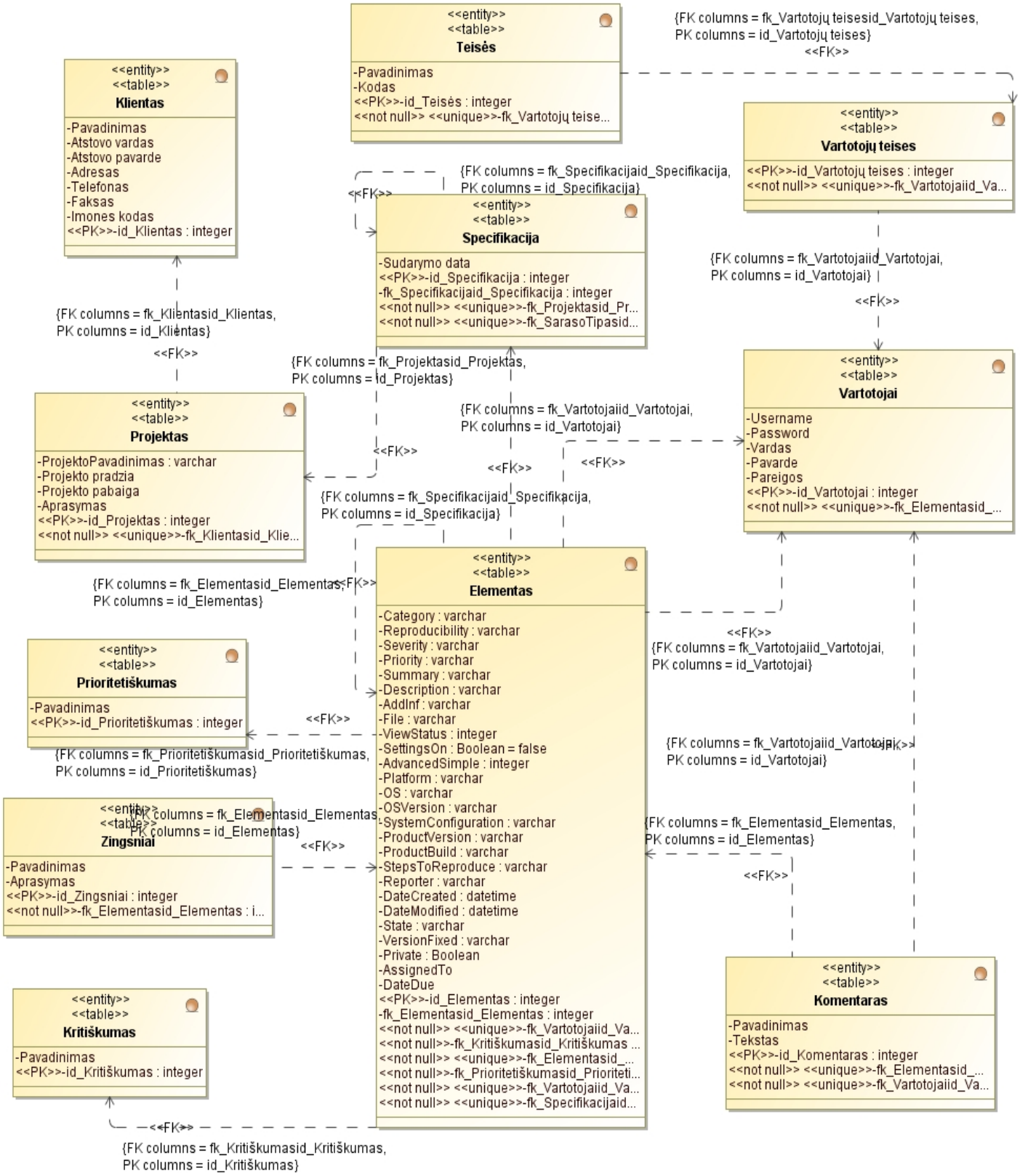


Pav. 25 Programuotojo posistemio sekų diagrama

Programuotojo posistemio sekų diagramoje (Pav. 25) vaizduojami programuotojo veiksmai: gauti užduotį, įvesti duomenis, išsaugoti užduotį.

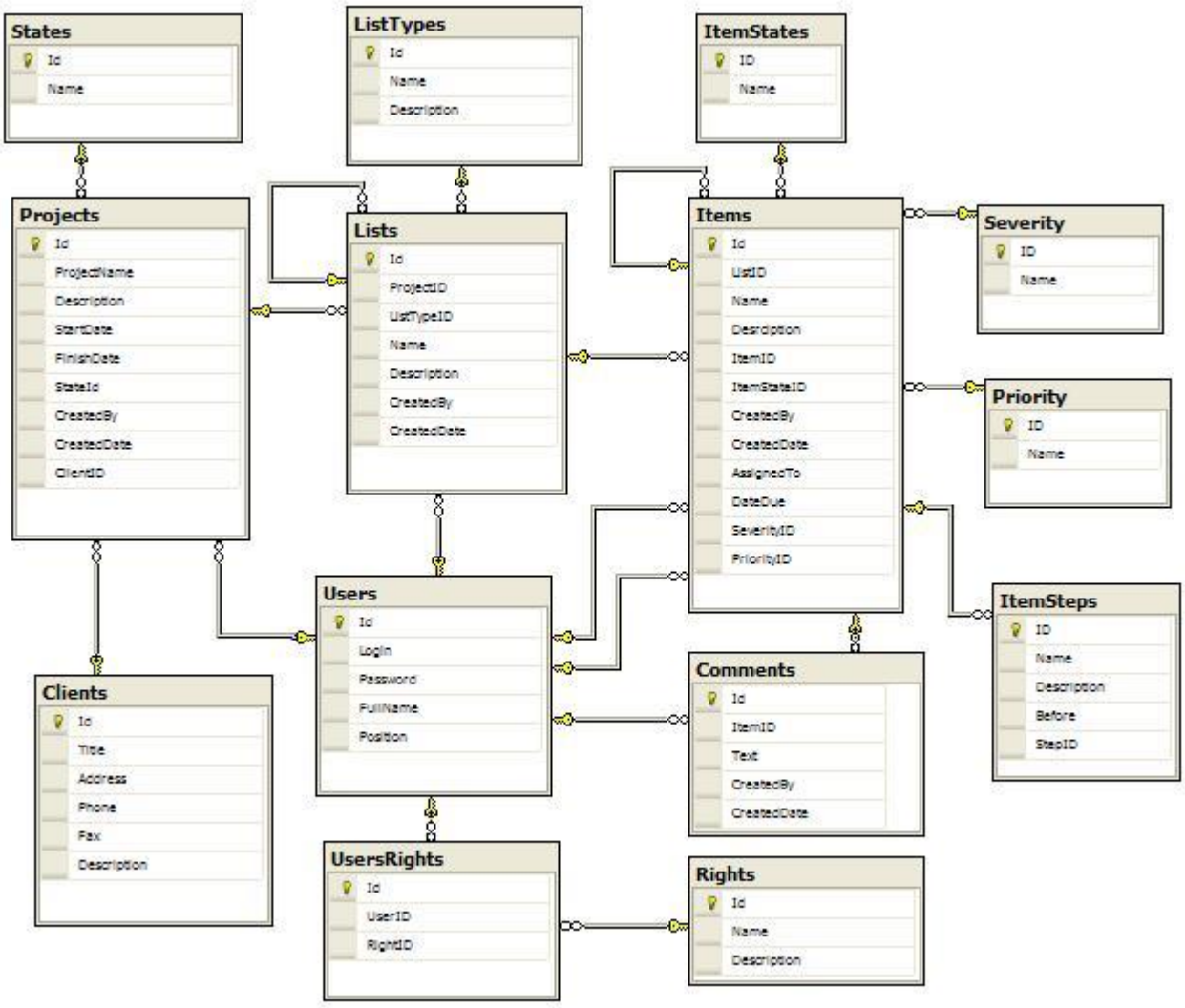
4.4. Duomenų bazės schema

Sekančiame paveiksle (Pav. 26) pateikiama sugeneruota duomenų bazės schema.



Pav. 26 DB schema

Duomenų bazės schemos diagrama, gauta iš MS SQL Server 2005 duomenų bazės:



Pav. 27 DB schema iš MS SQL Server 2005 duomenų bazės

Kadangi realizuotoje duomenų bazėje naudoti angliški terminai, žemiau pateikiami lentelių atitikmenys:

Lietuviškas atitikmuo	Angliškas atitikmuo
Elementas	Items
Komentaras	Comments
Vartotojai	Users
Vartotojų teisės	UserRights
Teisės	Rights

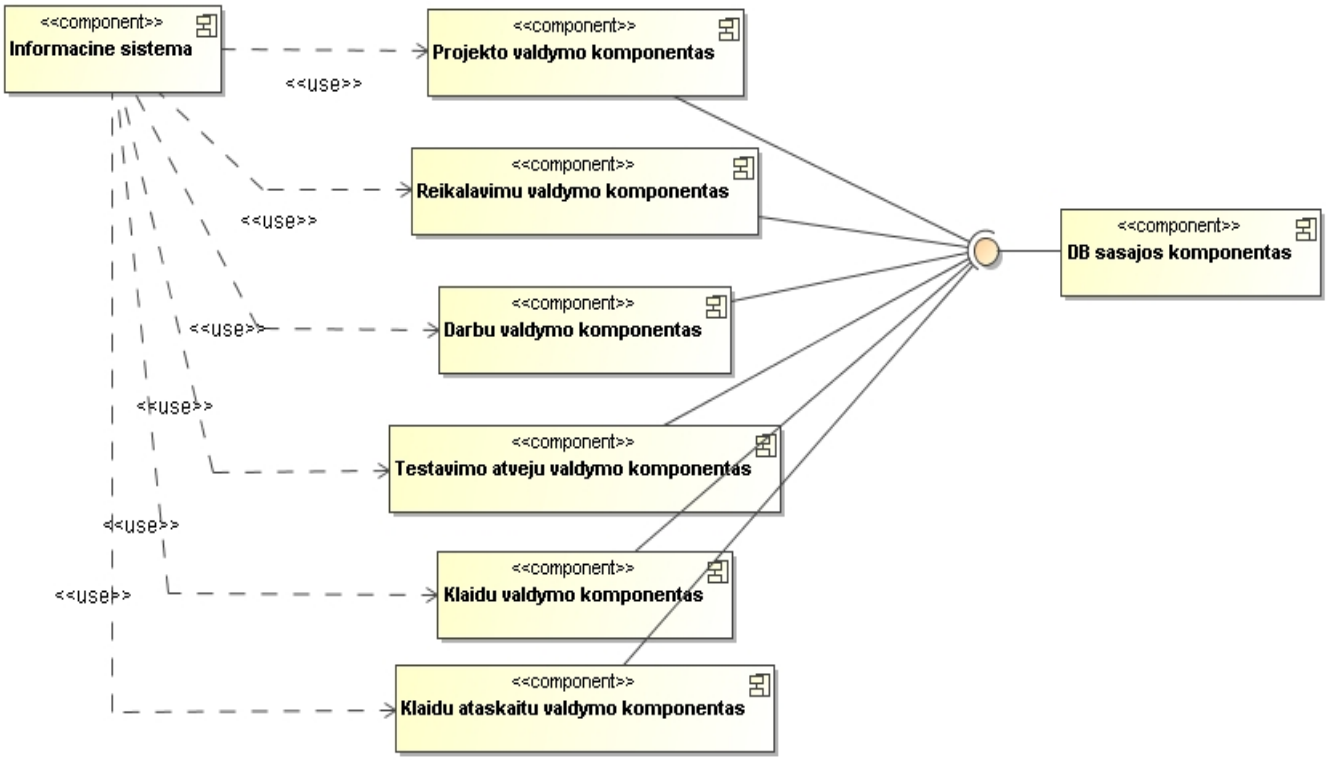
Specifikacija	Lists
Projektas	Projects
Klientas	Clients
Prioritetiškumas	Priority
Kritiškumas	Severity
Žingsniai	ItemSteps
Specifikacijos tipas	ListTypes
Elementų būsenos	ItemStates
Projekto būsenos	States

4.5. Realizacijos modelis (programinių komponentų architektūra, įdiegimo modelis)

Kuriamoje informacinėje sistemoje buvo sukurti atskiri programų komponentai, kurie realizuoja tam tikrą sistemos funkcionalumą. Pagrindiniai sistemos komponentai:

- **Projekto valdymo komponentas.** Šiame komponente yra atliekamas projekto informacijos valdymas.
- **Reikalavimų valdymo komponentas.** Šiame komponente atliekamas reikalavimų valdymas.
- **Darbų valdymo komponentas.** Šiame komponente yra atliekamas programuotojų darbų valdymas, skyrimas ir vykdymas.
- **Testavimo atvejų valdymo komponentas.** Šiame komponente atliekamas testavimo atvejų valdymas.
- **Klaidų valdymo komponentas.** Šia komponente atlikimas projektų klaidų valdymas.
- **DB sąsajos komponentas.** Šiame komponente yra visi duomenų lygmens objektai ir jų operacijos su duomenų baze.
- **Informacinė sistema.** Šiame komponente yra saugoma sistemos saugumo, meniu sudarymo ir kita pagalbinė logika.

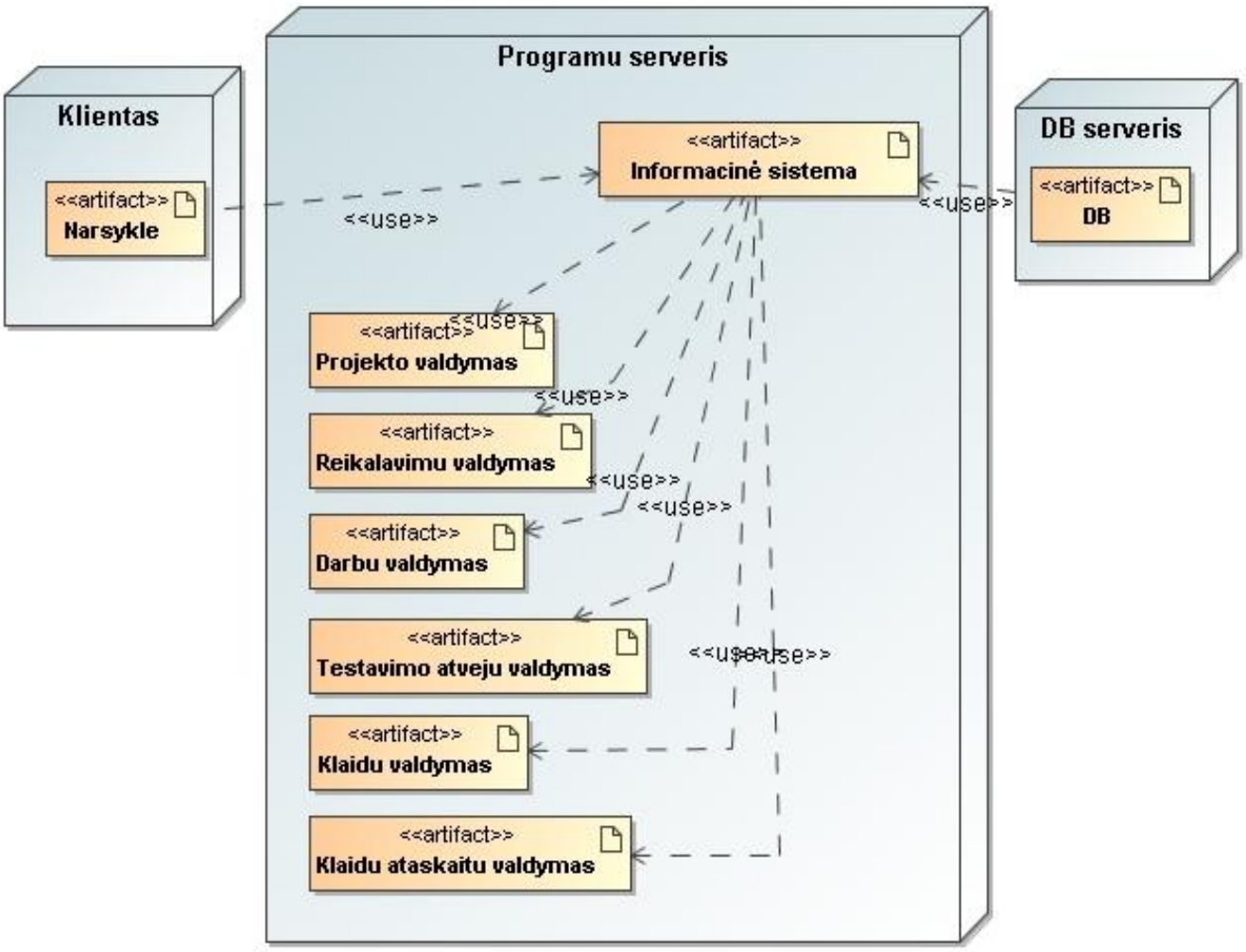
Komponentų diagrama vaizduojama sekančiame paveiksle (pav. 28).



Pav. 28 Komponentų diagrama

Sistemos komponentai buvo išreikšti atitinkamais sistemos artefaktais ir sudiegti į atitinkamus serverius. Diegimo diagrama pateikta Pav. 29.

Sistema numatyta realizuoti ASP.NET 2.0 programavimo technologija, o naudojama duomenų bazių valdymo sistema Ms SQL Server 2005. Vartotojas su programa galės dirbti Interneto naršyklės pagalba.



Pav. 29 Diegimo diagrama

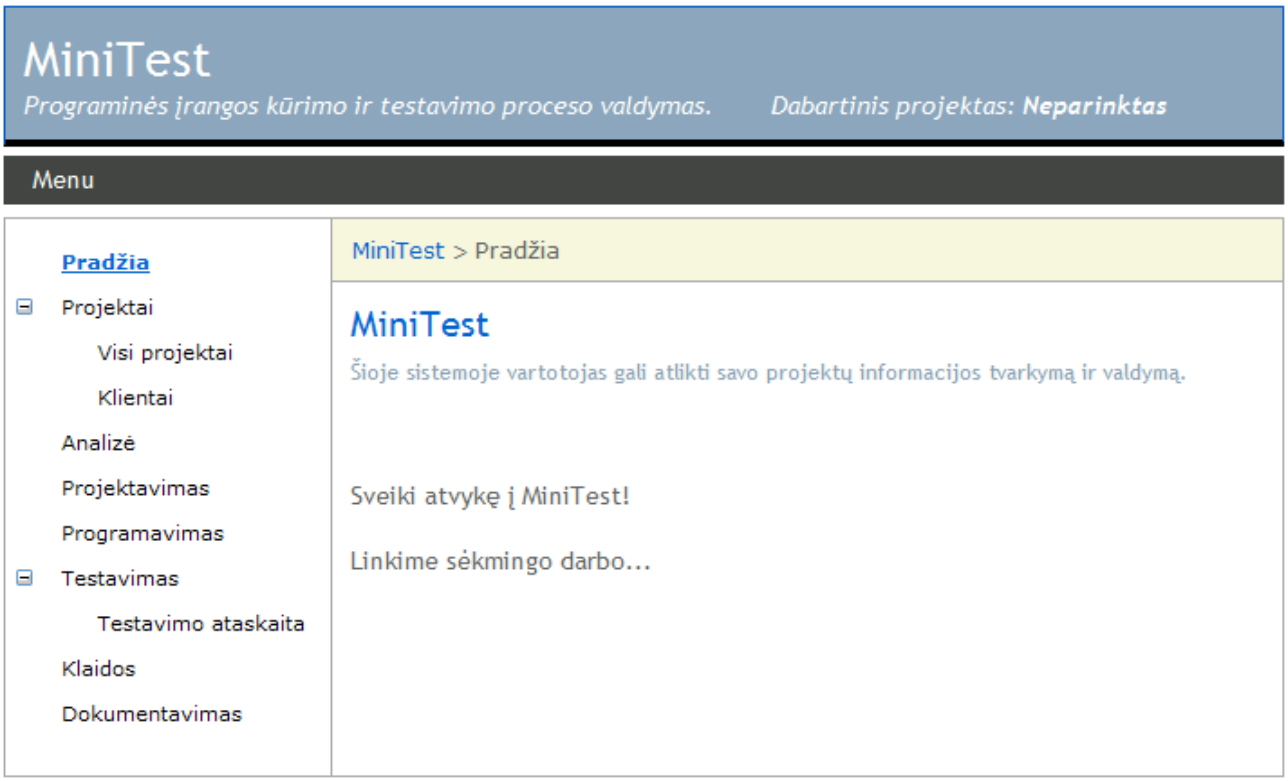
5. Realizacija

Išsiaiškinus programinės įrangos testavimo proceso valdymo automatizavimui skirto specializuoto įrankio poreikį, buvo realizuota sistema „MiniTest“. Šiame skyriuje yra aprašytas sistemos veikimas, pateiktas sistemos vaizdas, testavimo modelis ir pavyzdys bei apibendrintas sukurtas sprendimas.

5.1. Sistemos veikimo aprašymas

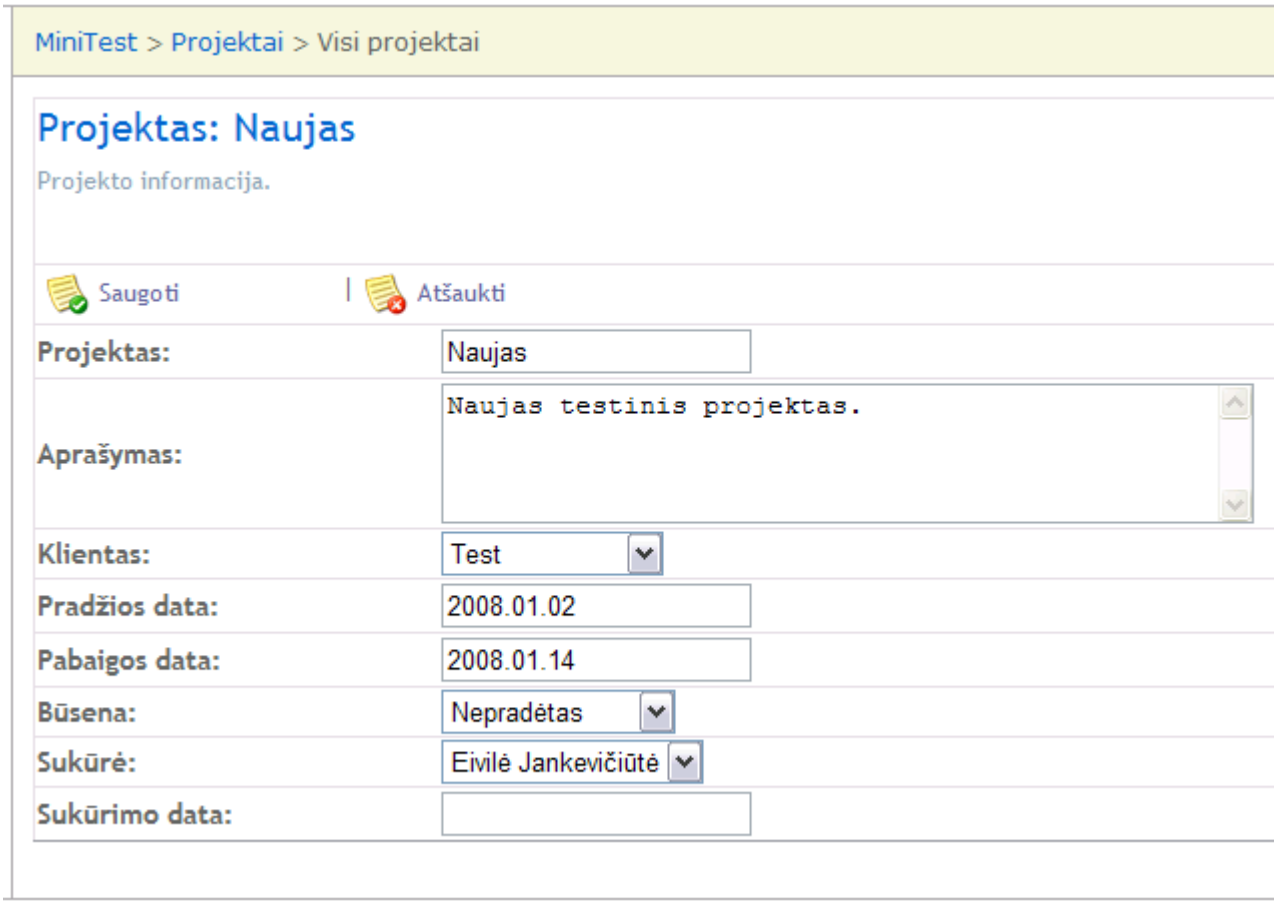
MiniTest – tai programinės įrangos kūrimo ir testavimo proceso valdymo automatizavimui skirtas įrankis. Jo pagalba įmonės gali automatizuoti programinės įrangos kūrimo gyvavimo ciklo metu vykstančių procesų valdymą, apjungti šiems procesams reikalingą informaciją bei suintegruoti

ją į bendrą visumą. Kaip ir buvo suprojektuota, sistemoje realizuoti šeši moduliai: projektų vadovo, analitiko, projektuotojo, programuotojo, testuotojo bei dokumentuotojo (Pav. 30).

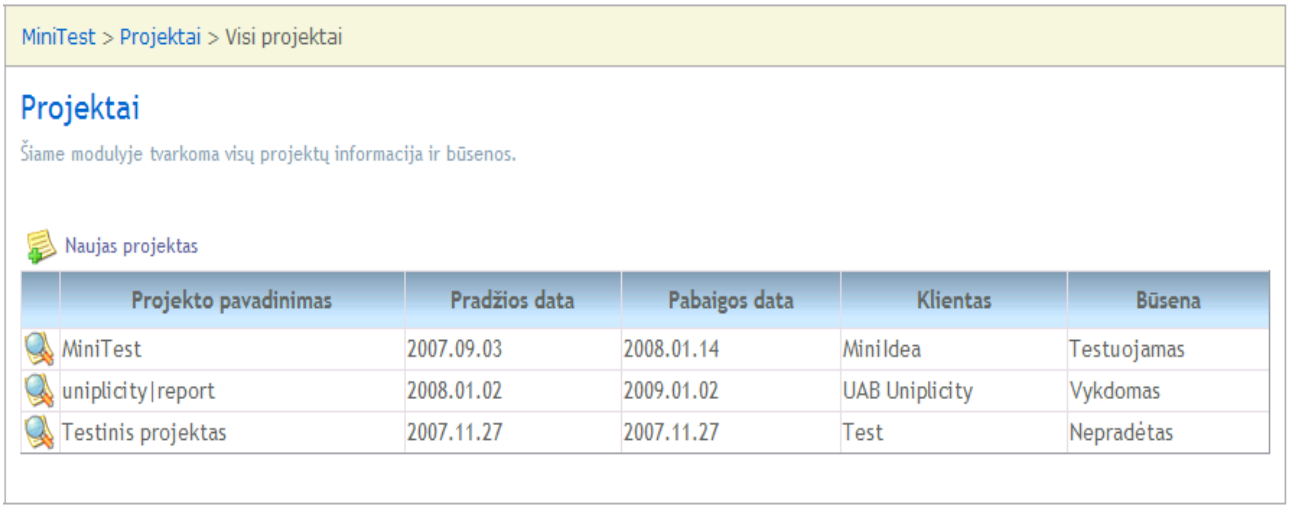


Pav. 30 Pagrindinis Meniu – sistemos moduliai

Vartotojas, prisijungęs prie sistemos (projektų vadovas), gali matyti jau sukurtus projektus (Pav. 31). Taip pat gali suvesti naujus projektus, redaguoti ar šalinti jau esančius (Pav. 32).

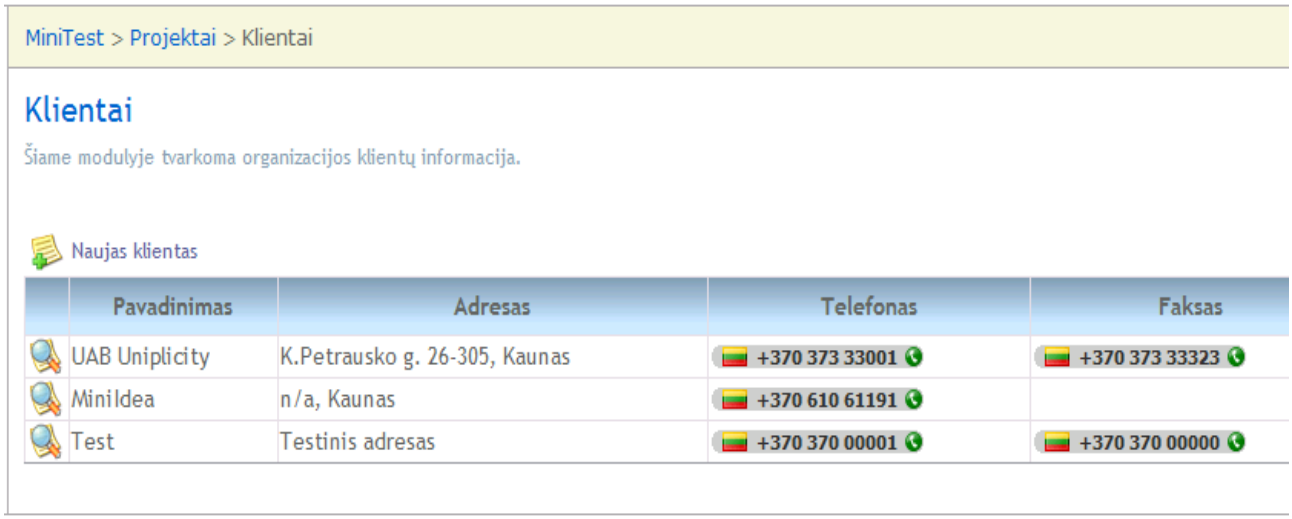


Pav. 31 Projektų vadovo modulis – naujo projekto kūrimo langas

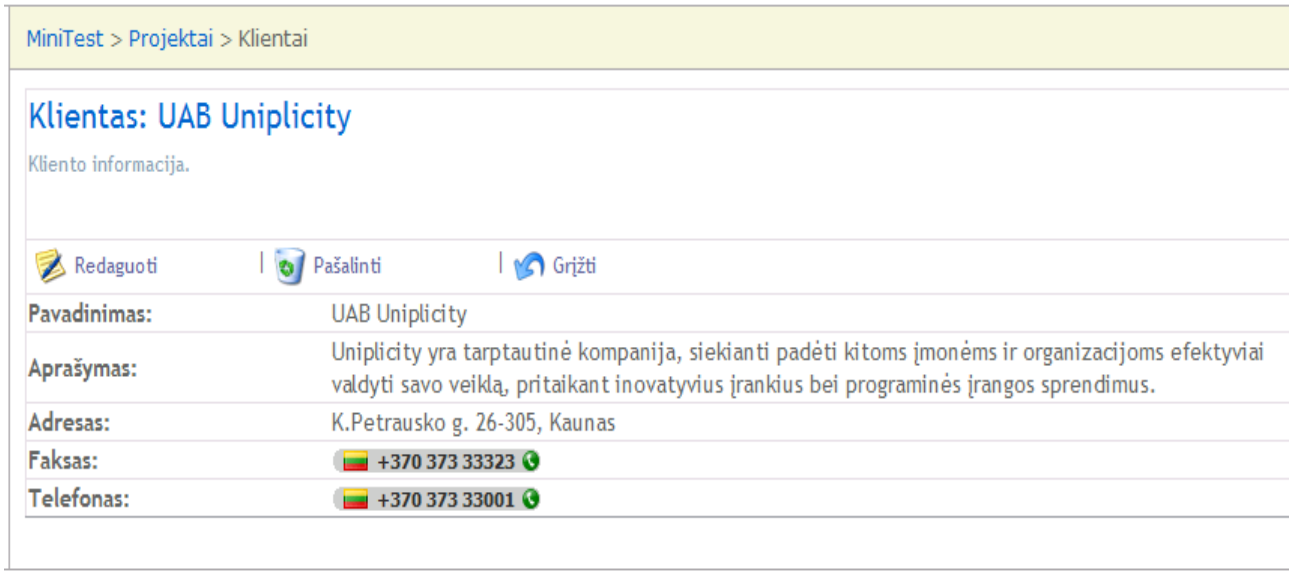


Pav. 32 Projektų vadovo modulis – projektų kūrimo/redagavimo langas

Lygiai taip pat galima žiūrėti ar redaguoti klientus (Pav. 33, 34).

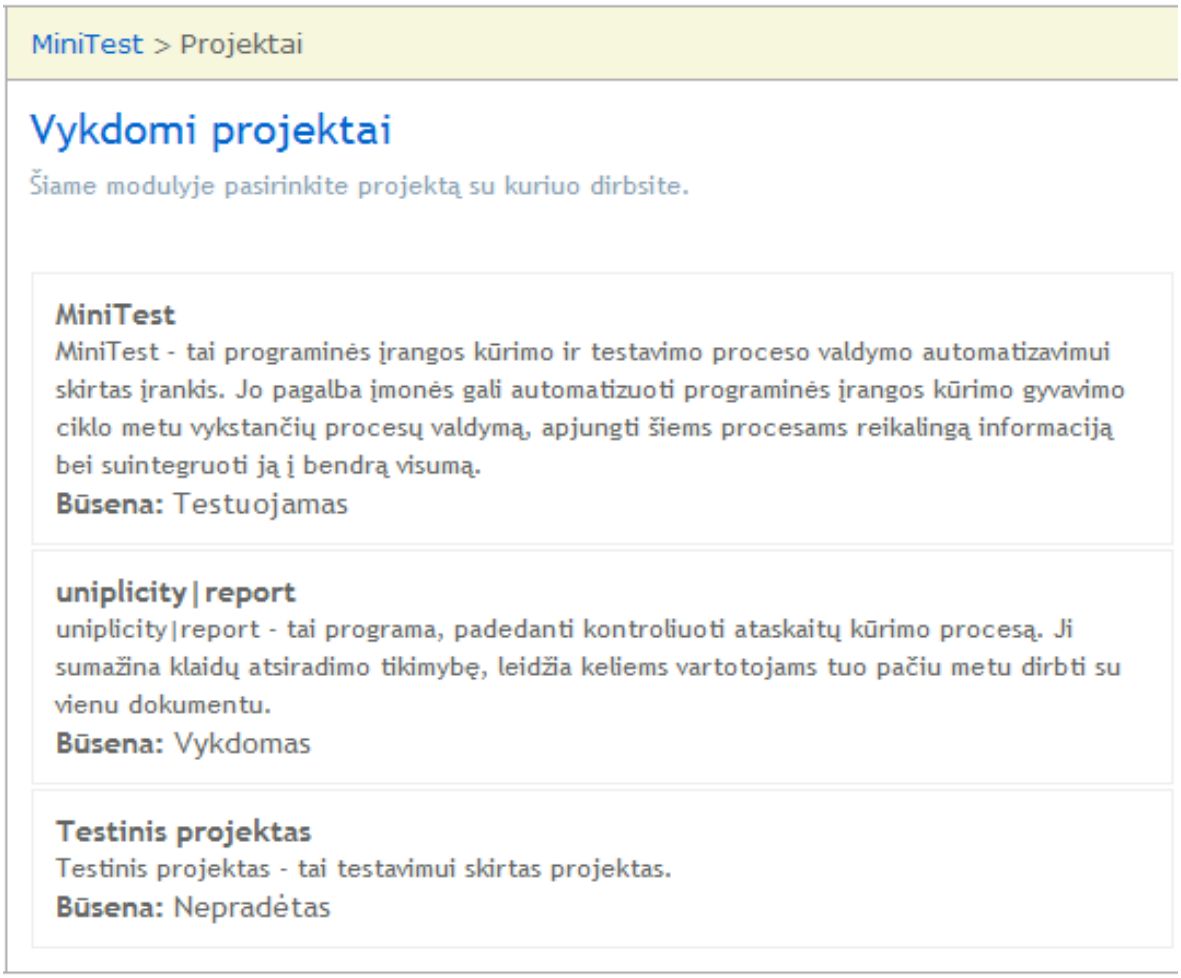


Pav. 33 Projektų vadovo modulis – klientų sąrašas



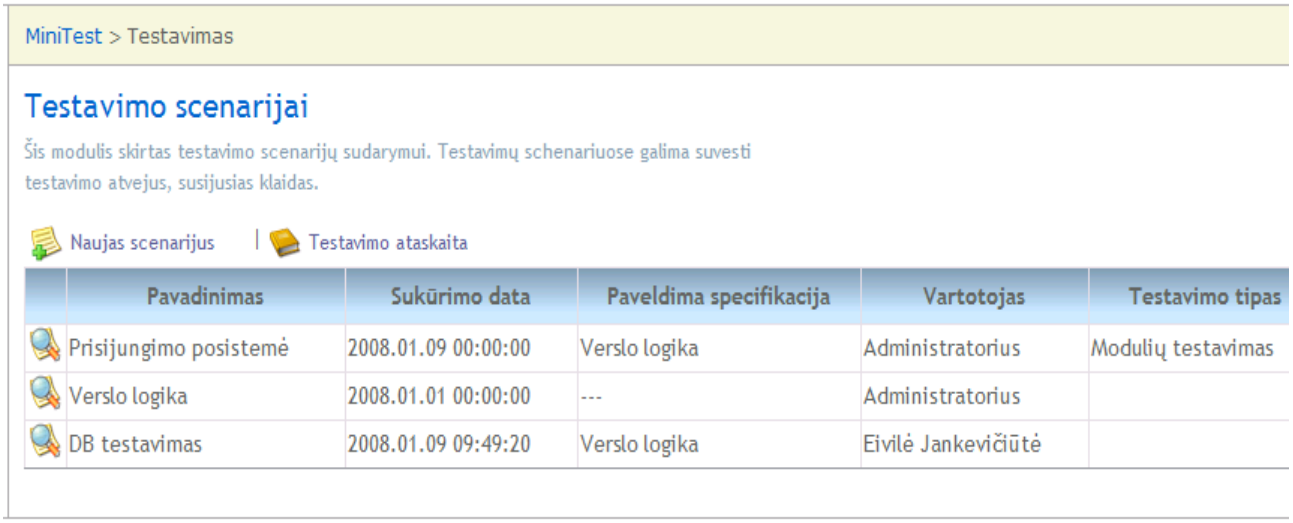
Pav. 34 Projektų vadovo modulis – klientų redagavimo/šalinimo pasirinkimo langas

Suvedus ar paredagavus projektus, vartotojas gali peržiūrėti jau pradėtus vykdyti projektus, juos pasirinkti bei įeiti. Šią funkciją vaizduoja sekantis paveikslas (Pav. 35).



Pav. 35 Projektų vadovo modulis – vykdomų projektų sąrašas

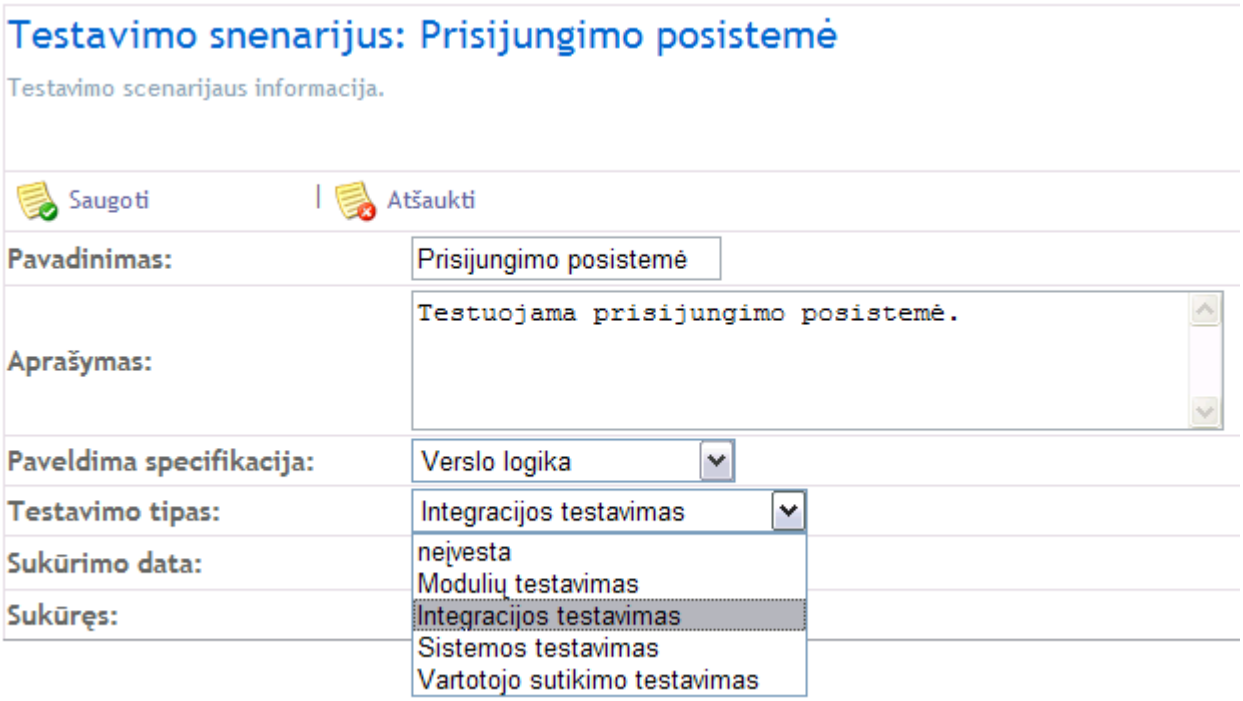
Pasirinkus norimą projektą ir į jį įėjus, Testavimo modulyje matome jam priskirtų testavimo scenarijų sąrašą (Pav. 36).



Pavadinimas	Sukūrimo data	Paveldima specifikacija	Vartotojas	Testavimo tipas
Prisijungimo posistemė	2008.01.09 00:00:00	Verslo logika	Administratorius	Modulių testavimas
Verslo logika	2008.01.01 00:00:00	---	Administratorius	
DB testavimas	2008.01.09 09:49:20	Verslo logika	Eivilė Jankevičiūtė	

Pav. 36 Testuotojo modulis – testavimo scenarijų sąrašas

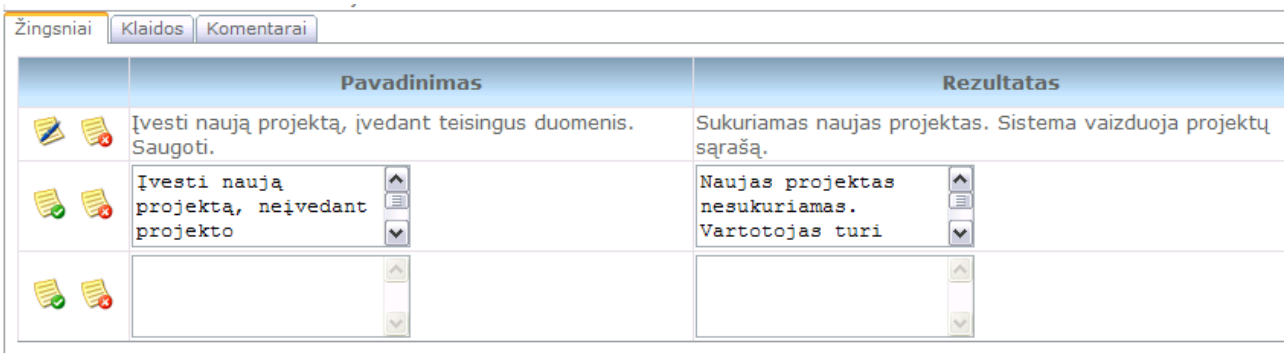
Testavimo scenarijus testuotojas gali redaguoti bei šalinti. Tai matome Pav. 37.



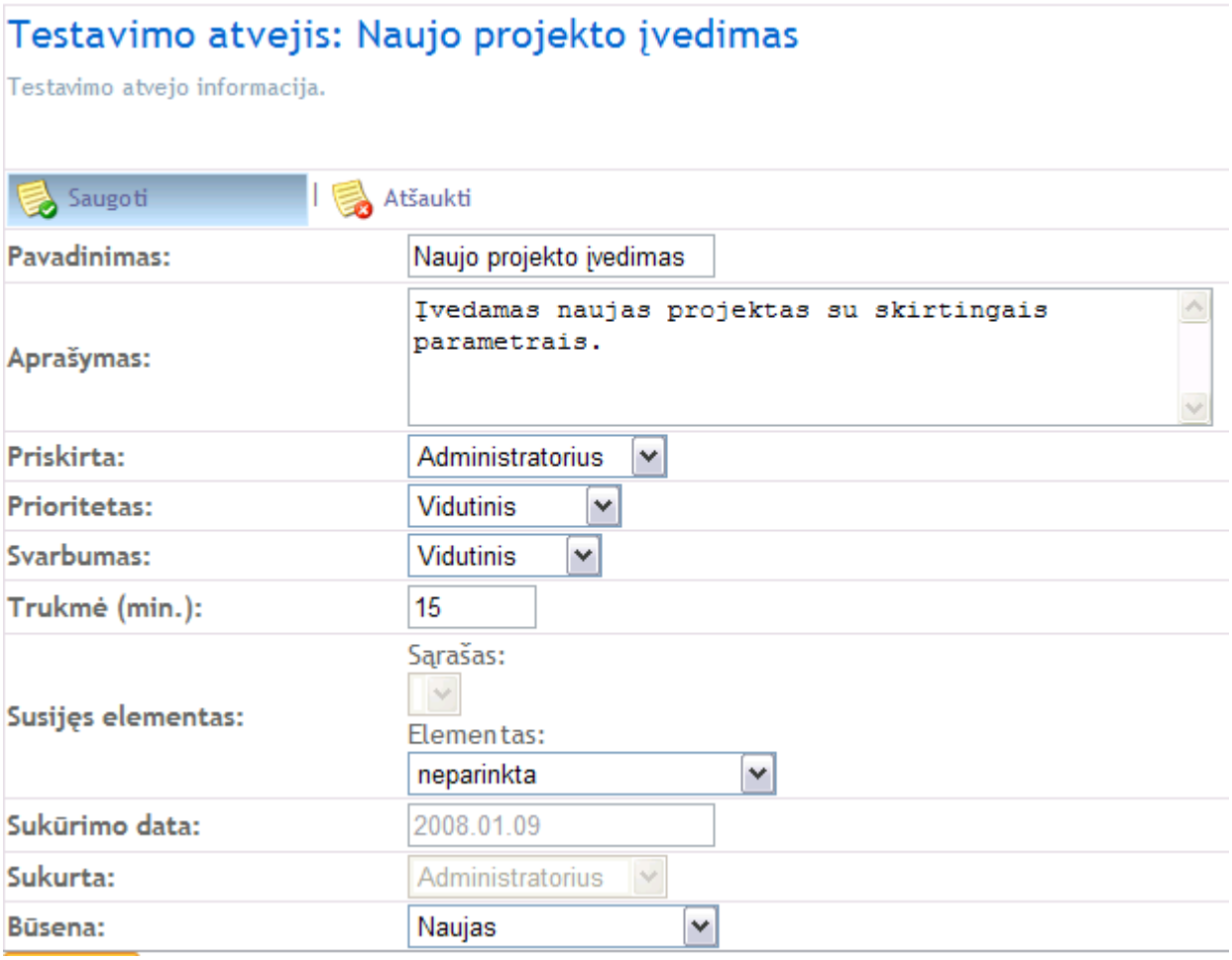
Pav. 37 Testuotojo modulis – Testavimo scenarijaus redagavimo/šalinimo pasirinkimo langas

Šiame paveiksle atsispindi dar viena naudinga sistemos funkcija – paveldimos specifikacijos parinkimas. Vartotojas, vesdamas testavimo scenarijų, turi galimybę pasirinkti elementą, su kuriuo jis bus susietas. Pavyzdžiui, testavimo scenarijus gali būti susietas su reikalavimo specifikacija ar programavimo užduočių sąrašu, ar panašiai.

Toliau vaizduojami testavimo žingsnio (Pav. 38) bei testavimo atvejo (Pav. 39) įvedimo langai.

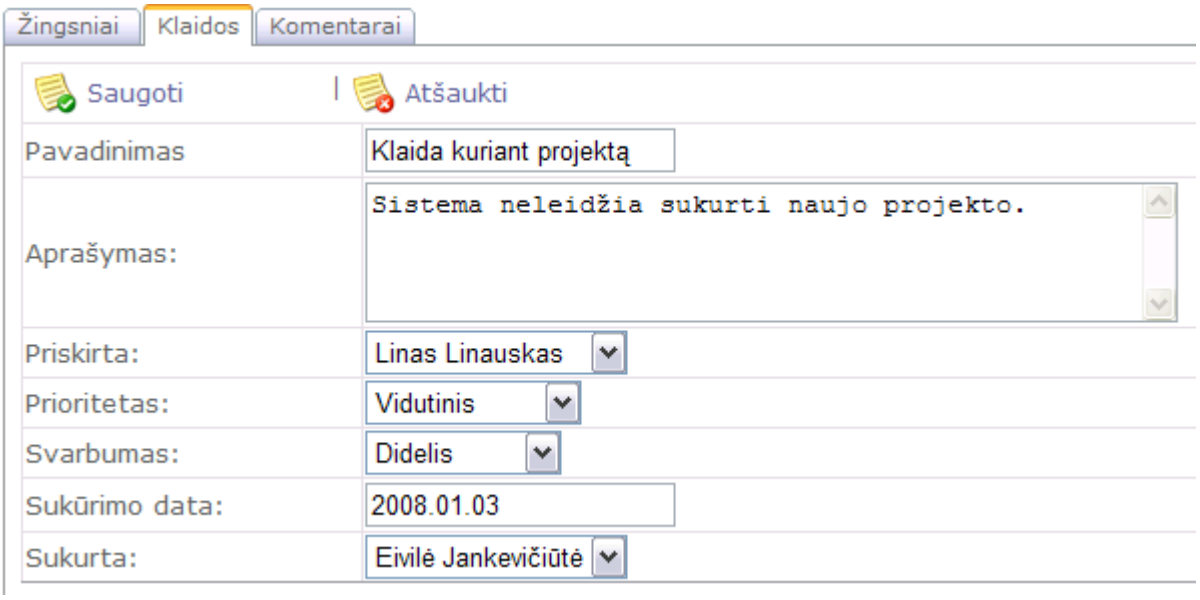


Pav. 38 Testuotojo modulis – Testavimo žingsnio įvedimo langas



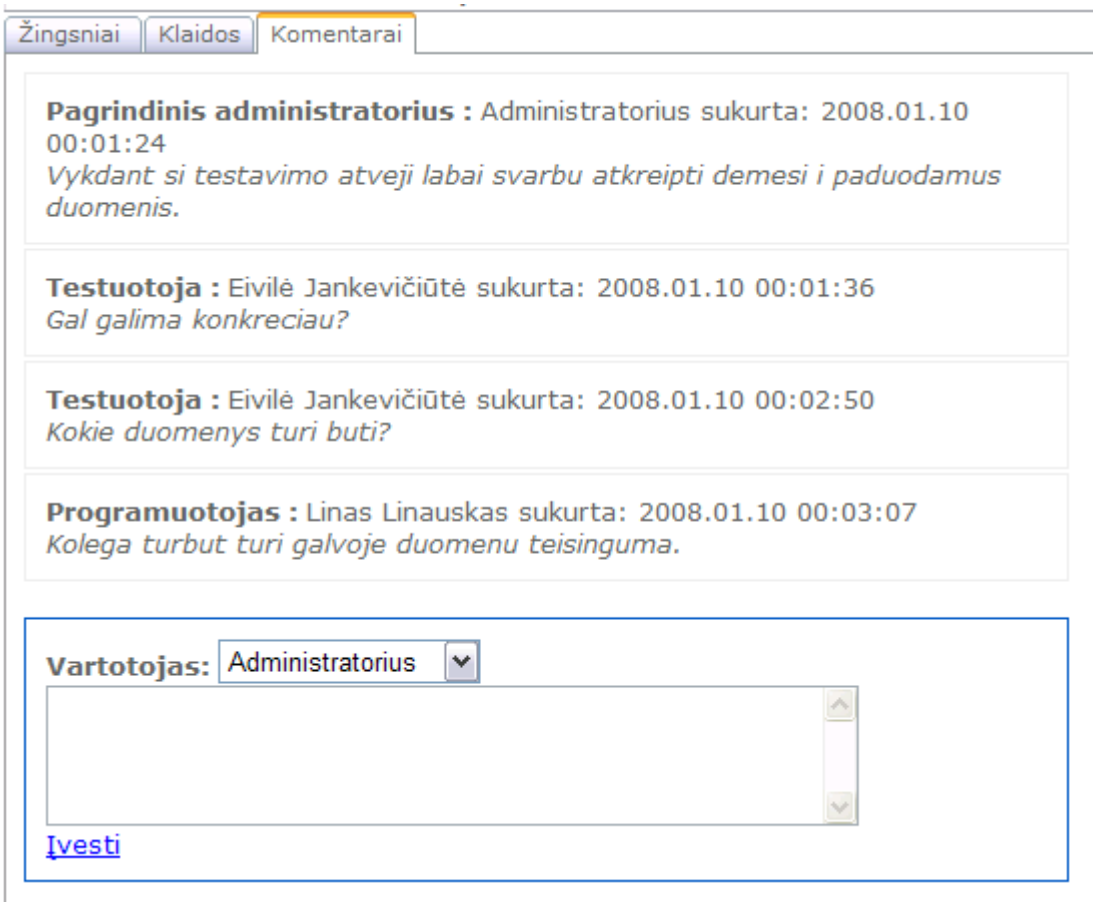
Pav. 39 Testuotojo modulis – Testavimo atvejo įvedimo langas

Toliau pateikiama klaidos įvedimo langas (Pav. 40).



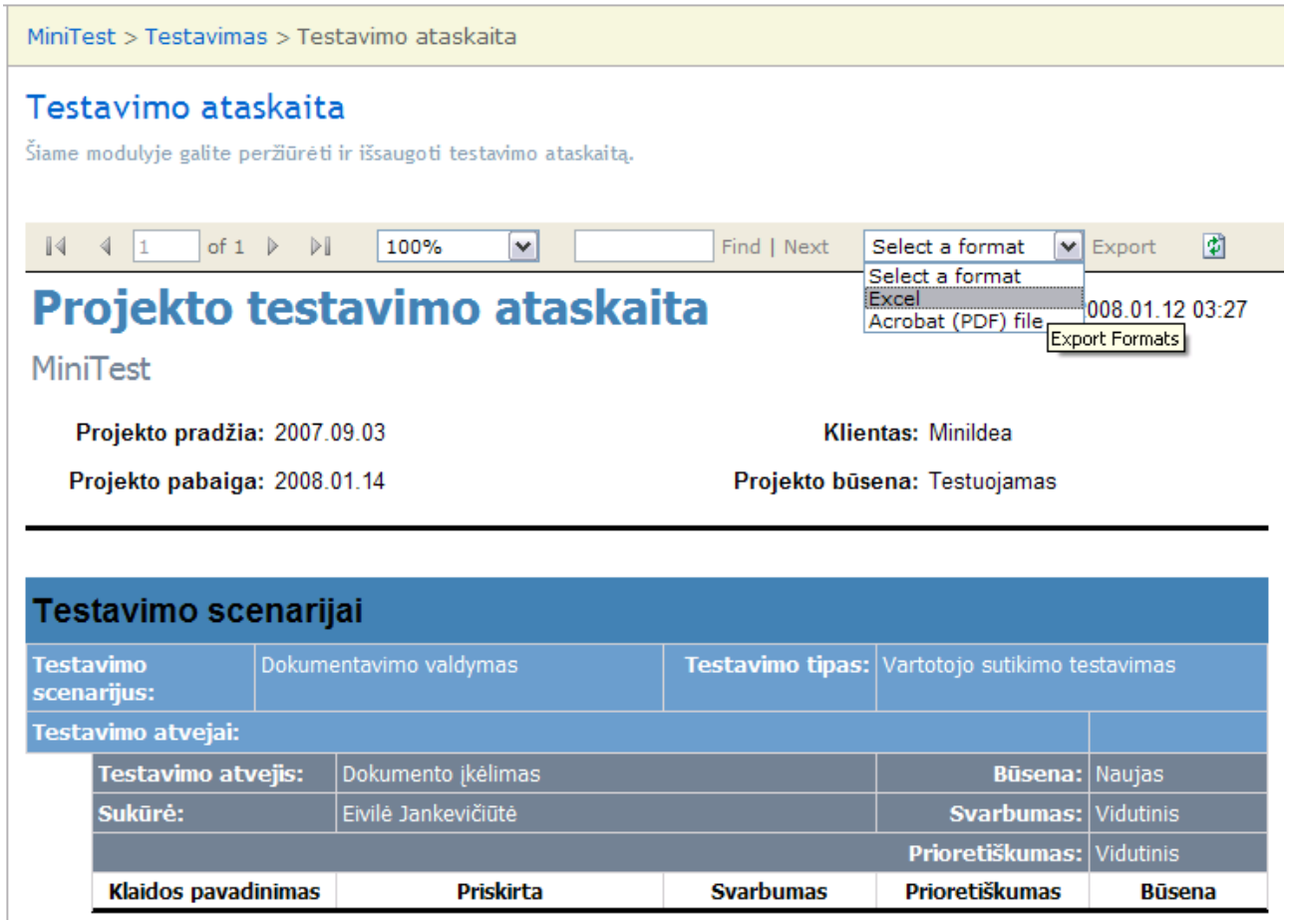
Pav. 40 Testuotojo modulis – Klaidos įvedimo langas

Taip pat testavimo atvejui galima rašyti komentarus. Ši funkcija vaizduojama Pav. 41.



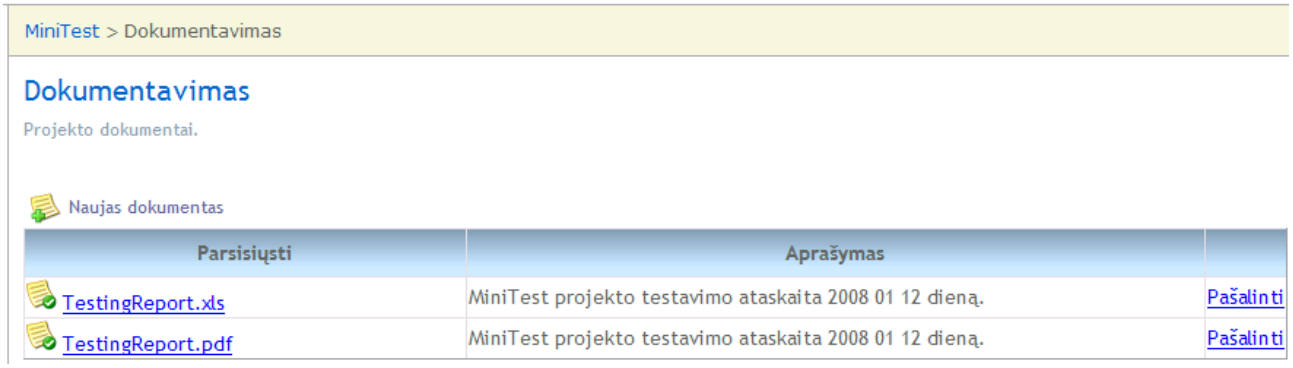
Pav. 41 Testuotojo modulis – Komentarų įvedimas

Dirbant su sistema, kuriant testavimo scenarijus, scenarijams priskiriant testavimo atvejus, o šiems įvedant žingsnius bei registruojant klaidas, dažnai yra poreikis statistiškai pavaizduoti susikaupusių duomenų gausą. „MiniTest“ suteikia tokią galimybę – sugeneruoja ataskaitą projektui, kurioje matomi projekto testavimo scenarijai, testavimo atvejai, žingsniai bei užregistruotos klaidos. Sekančiame paveiksle vaizduojamas šios ataskaitos fragmentas (Pav. 42). Taip pat čia matoma ir dar viena sistemos funkcija – yra galimybė sugeneruotą ataskaitą išeksportuoti į savo kompiuterį *.xls arba *.pdf formatais. Tai ypač patogu norint persiųsti informaciją kitam asmeniui ar norint vienaip ar kitaip apdoroti turimus duomenis.



Pav. 42 Testuotojo modulis – Testavimo ataskaitos fragmentas

Paskutiniame sistemos modulyje, dokumentavime, realizuota dar viena sistemos funkcija – dokumentų įkėlimas, parsisiuntimas bei šalinimas (Pav. 43). Labai patogu tokiu būdu saugoti sistemoje sugeneruotą ataskaitą, nes ataskaita yra generuojama esamiems duomenims ir jiems pasikeitus – ataskaita irgi pasikeičia.



Pav. 43 Dokumentuotojo modulis – Dokumentų saugojimas

5.2. Testavimo modelis bei duomenys, kontrolinis pavyzdys

Norint ištestuoti sukurtą programinės įrangos kūrimo ir testavimo automatizavimo sistemą, bus atliekamas sukurtos informacinės sistemos testavimas pagal darbo analizės metu iškeltus sistemų vertinimo kriterijus.

Naudojami testavimo metodai:

- Modulių testavimas (angl. *Unit testing*)
- Integracijos testavimas (angl. *Integration testing*)
- Sistemos testavimas (angl. *System testing*)
- Vartotojo sutikimas (angl. *User acceptance*)

Modulių testavimas – tai testavimas, skirtas testuoti atskirus programinės įrangos komponentus [18]. Tai testavimas pradinėje stadijoje, kada turi būti užtikrintas atskirų programinės įrangos komponentų teisingas veikimas. Dažniausiai modulių testavimas yra atliekamas pačių programuotojų arba testuotojų, bet ne galinių vartotojų.

Šiame projekte modulių testavimas buvo atliekamas modulių kūrimo metu.

Integracijos testavimas – tai tokia programinės įrangos testavimo fazė, kurioje atskiri programinės įrangos komponentai yra sujungiami ir testuojami kaip grupė, kaip vientisai veikianti sistema [19]. Ši fazė seka po modulių testavimo ir atliekama prieš sistemos testavimą. Jos metu moduliai, praeitoje fazėje testuoti atskirai, yra apjungiami ir testuojama jų integracija, kuri vėliau bus paduodama sekančiam etapui – sistemos testavimui.

Kontrolinis integracijos testavimo pavyzdys pateikiamas sekančioje lentelėje.

Lentelė 3 Kontrolinis integracijos testavimo pavyzdys

Testavimo scenarijus:	Projekto valdymas	Testavimo tipas:	Integracijos testavimas	
Testavimo atvejai:				
Testavimo atvejis:	Įvesto projekto redagavimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Testavimo atvejui klaidų neužregistruota!				

Testavimo atvejis:	Naujo projekto įvedimas		**Būsena:**	Naujas
Sukūrė:	Administratorius		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Neivedus pavadinimo - klaida.	Eivilė Jankevičiūtė	Didelis	Aukštas	Naujas
Aprašymas:	*Server Error in '/minitest' Application.*			

Testavimo atvejis:	Projekto šalinimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Šalinant projektą, sistema palūžta.	Linas Linauskas	Blokuojantis	Vidutinis	Naujas
Aprašymas:	*Server Error in '/DevelopmentWebsite' Application.*			

Sistemos testavimas – tai programinės įrangos testavimo fazė, kurios metu testavimas yra atliekamas išbaigtoje, integruotoje sistemoje ir yra vertinamas sistemos atitikimas nustatytiems reikalavimams [20]. Tai testavimas, paremtas „juodos dėžės“ principu, todėl testuotojas neprivalo išmanyti nei vidinės kodo struktūros, nei programavimo logikos. Kaip taisyklė, sistemos testavime kaip pradiniai duomenys yra paduodami visi suintegruoti programos komponentai, kurie buvo sėkmingai ištestuoti integracijos testavimo metu. Taip pat testuojama pati programinės įrangos sistema, suintegruota su taikomosiomis kompiuterių sistemomis.

Integracijos testavimo tikslas yra aptikti bet kokius nesuderinamumus tarp suintegruotų tarpusavyje programinės įrangos komponentų arba tarp šių grupių bei techninės įrangos. Sistemos testavimas yra labiau ribojantis testavimo tipas. Jo metu siekiama aptikti defektus tarp programinės ir techninės įrangos bei sistemoje kaip visumoje.

Kontrolinis sistemos testavimo pavyzdys pateikiamas sekančioje lentelėje.

Lentelė 4 Kontrolinis sistemos testavimo pavyzdys

Testavimo scenarijus:	Testavimo valdymas	**Testavimo tipas:**	Sistemos testavimas	
Testavimo atvejai:				
Testavimo atvejis:	Klaidos įvedimas		**Būsena:**	Priskirtas
Sukūrė:	Administratorius		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Testavimo atvejo įvedimas		**Būsena:**	Naujas
Sukūrė:	Administratorius		**Svarbumas:**	Išplėtimas
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Testavimo scenarijaus įvedimas		**Būsena:**	Naujas
Sukūrė:	Administratorius		**Svarbumas:**	Išplėtimas
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Klaida įvedant scenarijų	Petras Petraitis	Didelis	Vidutinis	Naujas
Aprašymas:	*Norint įvesti scenarijų, neleidzia paspausti naujo scenarijaus sukurimo mygtuko.*			

Testavimo atvejis:	Testavimo scenarijaus redagavimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Neleidžia redaguojti scenarijaus	Linas Linauskas	Didelis	Vidutinis	Naujas
Aprašymas:	*Norint redaguoti scenarijų, sistema nukeliauja į scenarijų sąrašą.*			
Scenarijų sąraše nerodo kam priskirta.	Eivilė Jankevičiūtė	Vidutinis	Vidutinis	Naujas
Aprašymas:	*Paredagavus lauką kam priskirta, jo neberodo scenarijų sąraše.*			

Testavimo atvejis:	Testavimo scenarijaus šalinimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Testavimo žingsnio įvedimas		**Būsena:**	Naujas
Sukūrė:	Administratorius		**Svarbumas:**	Išplėtimas
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Sekančioje lentelėje (Lentelėje 5) pateikiamas sistemos testavimo pavyzdys.

Lentelė 5 Sistemos testavimas

Testavimo atvejis:	Sistemos testavimas (angl. System testing)
Laukiamas rezultatas:	Laukiamas rezultatas - programinės įrangos kūrimo ir testavimo automatizavimo sistema, padedanti automatizuoti programinės įrangos testavimo valdymą bei integruoti testavimo procesą į bendrą programinės įrangos kūrimo proceso visumą. Sistema turi atitikti analizės bei projektavimo metu iškeltus reikalavimus bei atlikti numatytas funkcijas.
Testavimo rezultatas:	Sukurtos programinės įrangos kūrimo ir testavimo automatizavimo sistemos pagalba testavimo procesas bus integruotas į programinės įrangos kūrimo procesų visumą. Testavimo proceso valdymas sistemos pagalba taps labiau automatizuotas.
Testavimo atvejo įvertinimas:	**Teigiamas**

Vartotojų sutikimo testavimas – tai „juodos dėžės“ principo testavimas, vykdomas sistemoje prieš jos diegimą [21]. Šis testavimo metodas dar vadinamas funkciniu, „juodos dėžės“, paleidimo, kokybės užtikrinimo, galutiniu, vartotojo patogumo testavimu ir pan. Dažnai šis testavimas yra atliekamas galutinių vartotojų. Tokiais atvejais, jis vadinamas „beta“ testavimu, vartotojo sutikimo testavimu ar galutinio vartotojo testavimu. Šis testavimas yra atliekamas bandant priversti sistemą palūžti, naudojant realius duomenis, su kuriais ta sistema yra sukurta realiai dirbti. Operuojama tokioje aplinkoje ir tokiame kontekste, kokiame sistema yra numatyta dirbti.

Kontrolinis vartotojo sutikimo testavimo pavyzdys pateikiamas sekančioje lentelėje.

Lentelė 6 Kontrolinis vartotojo sutikimo testavimo pavyzdys

Testavimo scenarijus:	Dokumentavimo valdymas	**Testavimo tipas:**	Vartotojo sutikimo testavimas	
Testavimo atvejai:				
Testavimo atvejis:	Dokumento įkėlimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
			Testavimo atvejui klaidų neužregistruota!	

Testavimo atvejis:	Dokumento parsisiuntimas į kompiuterį		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Saugoti dokumentą toje pačioje vietoje	Linas Linauskas	Išplėtimas	Vidutinis	Naujas
Aprašymas:	*Parsisiunčiant dokumentą į kompiuterį, sistema turėtų atsiminti paskutinį saugojimo kelią ir toje vietoje siūlyti saugoti vėl.*			

Testavimo atvejis:	Dokumento šalinimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Nedidelis
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Šalinant dokumentą, paklausti ar tikrai	Linas Linauskas	Išplėtimas	Vidutinis	Naujas
Aprašymas:	*Šalinant dokumentą, sistema galėtų ne iš karto trinti, o paklausti ar tikrai reikia pašalinti.*			

5.3. Sukurto sprendimo apibendrinimas

Sukurta testavimo valdymo sistema „MiniTest“ buvo ištestuota ir rastos klaidos ištaisytos. Kadangi ji atitinka visus anksčiau iškeltus reikalavimus bei pateisina vartotojų lūkesčius, galima teigti, jog sprendimas yra geras ir rezultatas yra teigiamas.

6. Eksperimentinis sistemos tyrimas

Darbo metu buvo sukurta programinės įrangos testavimo valdymo sistema, automatizuojanti ir integruojanti testavimo procesą į bendrą programinės įrangos kūrimo procesų visumą. Norint išsiaiškinti sukurto produkto svarbą bei patrauklumą vartotojui, buvo atliktas eksperimentinis sistemos tyrimas.

6.1. Eksperimentas. „MiniTest“ testavimo ataskaita

Šis tyrimas buvo atliekamas testuojant pačią sistemą ir vėliau suintegruojant informaciją. Tyrimo metu buvo testuojama sistema „MiniTest“, naudojantis „V“ modelyje rekomenduojamais testavimo metodais: modulių testavimu, integracijos testavimu, sistemos testavimu bei vartotojo sutikimo testavimu. Darbo su sistema metu buvo sukaupta informacija, iš kurios vėliau buvo sugeneruojama ataskaita, kuri patiekiama sekančiame paveiksle (Pav. 44).

Projekto testavimo ataskaita

Sudaryta:
2008.01.12 04:43

MiniTest

Projekto pradžia:	2007.09.03	**Klientas:**	MiniIdea
Projekto pabaiga:	2008.01.14	**Projekto būsena:**	Testuojamas

Testavimo scenarijai

Testavimo scenarijus:	Dokumentavimo valdymas	**Testavimo tipas:**	Vartotojo sutikimo testavimas
Testavimo atvejai:			

Testavimo atvejis:	Dokumento įkėlimas	**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė	**Svarbumas:**	Vidutinis
		Prioretiškumas:	Vidutinis

Klaidos pavadinimas	Priskirta	Svarbumas	Prioretiškumas	Būsena

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Dokumento parsisiuntimas į kompiuterį	**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė	**Svarbumas:**	Vidutinis
		Prioretiškumas:	Vidutinis

Klaidos pavadinimas	Priskirta	Svarbumas	Prioretiškumas	Būsena
Saugoti dokumentą toje pačioje vietoje	Linas Linauskas	Išplėtimas	Vidutinis	Naujas

Aprašymas:	*Parsisiunčiant dokumentą į kompiuterį, sistema turėtų atsiminti paskutinį saugojimo kelią ir toje vietoje siūlyti saugoti vėl.*

Testavimo atvejis:	Dokumento šalinimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Nedidelis
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Šalinant dokumentą, paklausti ar tikrai	Linas Linauskas	Išplėtimas	Vidutinis	Naujas
Aprašymas:	*Šalinant dokumentą, sistema galėtų ne iš karto trinti, o paklausti ar tikrai reikia pašalinti.*			

Testavimo scenarijus:	Meniu testavimas	**Testavimo tipas:**	Modulių testavimas	
Testavimo atvejai:				
Testavimo atvejis:	Dokumentavimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Neveikia dokumentavimo modulis.	Linas Linauskas	Vidutinis	Vidutinis	Naujas
Aprašymas:	*Norint patekti į dokumentavimo modulį, paspaudus meniu punktą "Dokumentavimas", jis neveikia.*			

Testavimo atvejis:	Pradžia		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Projektai	**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė	**Svarbumas:**	Vidutinis

			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Testavimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Aukštas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo scenarijus:	Projekto valdymas	**Testavimo tipas:**	Integracijos testavimas	
Testavimo atvejai:				

Testavimo atvejis:	Įvesto projekto redagavimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Naujo projekto įvedimas		**Būsena:**	Naujas
Sukūrė:	Administratorius		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Neivedus pavadinimo - klaida.	Eivilė Jankevičiūtė	Didelis	Aukštas	Naujas
Aprašymas:	*Server Error in '/minitest' Application.* *Error 515, Level 16, State 2, Procedure InsertProjects, Line 19, Message: Cannot insert the value NULL into column 'ProjectName', table 'MINI.dbo.Projects'; column does not allow nulls. INSERT fails.*			

Testavimo atvejis:	Projekto šalinimas		Būsena:	Naujas
Sukūrė:	Eivilė Jankevičiūtė		Svarbumas:	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Šalinant projektą, sistema palūžta.	Linas Linauskas	Blokuojantis	Vidutinis	Naujas
Aprašymas:	*Server Error in '/DevelopmentWebsite' Application.* *---* *The DELETE statement conflicted with the REFERENCE constraint "FK_tbl_items_tbl_lists". The conflict occurred in database "MINI", table "dbo.Items", column 'ListID'.* *The DELETE statement conflicted with the REFERENCE constraint "FK_tbl_lists_tbl_projects". The conflict occurred in database "MINI", table "dbo.Lists", column 'ProjectID'.* *The statement has been terminated.* *The statement has been terminated.*			

Testavimo scenarijus:	Testavimo valdymas	Testavimo tipas:	Sistemos testavimas	
Testavimo atvejai:				
Testavimo atvejis:	Klaidos įvedimas		**Būsena:**	Priskirtas
Sukūrė:	Administratorius		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Testavimo atvejo įvedimas		Būsena:	Naujas
Sukūrė:	Administratorius		Svarbumas:	Išplėtimas
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Testavimo scenarijaus įvedimas		Būsena:	Naujas
Sukūrė:	Administratorius		Svarbumas:	Išplėtimas
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Klaida įvedant scenarijų	Petras Petraitis	Didelis	Vidutinis	Naujas
Aprašymas:	*Norint įvesti scenarijų, neleidzia paspausti naujo scenarijaus sukurimo mygtuko.*			

Testavimo atvejis:	Testavimo scenarijaus redagavimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Vidutinis
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**
Neleidžia redaguojti scenarijaus	Linas Linauskas	Didelis	Vidutinis	Naujas
Aprašymas:	*Norint redaguoti scenarijų, sistema nukeliauja į scenarijų sąrašą.*			
Scenarijų sąraše nerodo kam priskirta.	Eivilė Jankevičiūtė	Vidutinis	Vidutinis	Naujas
Aprašymas:	*Paredagavus lauką kam priskirta, jo neberodo scenarijų sąraše.*			

Testavimo atvejis:	Testavimo scenarijaus šalinimas		**Būsena:**	Naujas
Sukūrė:	Eivilė Jankevičiūtė		**Svarbumas:**	Vidutinis
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Testavimo atvejis:	Testavimo žingsnio įvedimas		**Būsena:**	Naujas
Sukūrė:	Administratorius		**Svarbumas:**	Išplėtimas
			Prioretiškumas:	Žemas
Klaidos pavadinimas	**Priskirta**	**Svarbumas**	**Prioretiškumas**	**Būsena**

Testavimo atvejui klaidų neužregistruota!

Pav. 44 „MiniTest“ testavimo ataskaita

Šis testavimas įvertinamas pagal testavimo brandumo modelį TMM. Įverčiai pateikiami sekančioje lentelėje (Lentelė 7).

Lentelė 7 Sistemos įverčiai pagal TMM

TMM lygmuo	Lygmens savybės	„MiniTest“ savybių atitikimas	„MiniTest“ įvertinimas pagal TMM
1. Pradinis	• Testavimas yra chaotiškas procesas.	-	Neatitinka
	• Ryškus resursų, įrankių ir gerai apmokytų darbuotojų trūkumas.	-	
2. Apibrėžtas	• Apibrėžti tikslai.	+	Atitinka
	• Atliekamas planavimas.	+	
	• Įtvirtintos pagrindinės testavimo technologijos ir metodai. Testavimas išskirtas kaip atskira fazė.	+	
	• Tikslas – parodyti, kad programa atitinka specifikaciją.	+	
3. Integravimo	• Organizacijoje yra profesionalus personalas – testuotojai.	+	Atitinka
	• Testavimas integruojamas į gyvavimo ciklą.	+	
	• Testavimas suprantamas kaip profesionali veikla.	+	
	• Testavimo procesas nėra matuojamas, nedaromos peržiūros.	+	
4. Valdymo	• Peržiūros, matavimai, produktų kokybės vertinimas.	+/-	Dalinai atitinka
	• Testavimo scenarijai saugomi duomenų bazėje ir juos galima panaudoti iš naujo.	+	
	• Registruojami defektai turi svarbumo prioritetus.	+	

TMM lygmuo	Lygmens savybės	„MiniTest“ savybių atitikimas	„MiniTest“ įvertinimas pagal TMM
	• Trūksta defektų prevencijos, automatizuoto proceso duomenų rinkimo, testavimo proceso analizės.	+	
5. Optimizuojantis	• Defektų prevencija.	-	Neatitinka
	• Kokybės kontrolė.	-	
	• Proceso optimizavimas.	-	

Atlikus įvertinimą pagal testavimo brandumo modelį TMM, galima pastebėti, jog „MiniTest“ sistemos brandumo lygmuo yra tarp trečiojo ir ketvirtojo. Tai yra žymiai aukščiau už analizės metu nagrinėtų įrankių įvertinimą pagal šį brandumo modelį. Apibendrinta visų įrankių savybių analizė pateikiama sekančiame skyrelyje.

6.2. Savybių analizė

Kokybei įvertinti, buvo pasitelkta atlikta egzistuojančių produktų lyginamoji analizė bei joje naudojami vertinimo kriterijai. Rezultatas pateikiamas sekančioje lentelėje (Lentelė 8).

Lentelė 8 Savybių analizė

Lyginimo kriterijai	„Mantis“	„Bugzilla“	„Test Link“	„Microsoft Team System“	Testavimo valdymo sistema „MiniTest“
Galimybė sudaryti testavimo atvejus	Nerealizuota	Nerealizuota	Realizuota	Realizuota	Realizuota
Galimybė sekti bei valdyti klaidas	Realizuota	Realizuota	Nerealizuota	Realizuota	Realizuota

Lyginimo kriterijai	„Mantis“	„Bugzilla“	„Test Link“	„Microsoft Team System“	Testavimo valdymo sistema „MiniTest“
Galimybė generuoti diagramas apie produkto kokybę (klaidų skaičių bei svarbumą)	Nerealizuota	Nerealizuota	Nerealizuota	Realizuota	Numatyta realizuoti
Galimybė kaupti užregistruotų klaidų statistiką	Realizuota	Realizuota	Nerealizuota	Realizuota	Realizuota
Vartotojo sąsajos patogumas, lankstumas	Patogi	Patogi	Turi trūkumų	Reikalauja daug žinių ir yra sunkiai suprantamas	Patogi
Produkto kainos dydis	Nemokamas	Nemokamas	Nemokamas	Mokamas, netgi labai brangus	Nemokamas
TMM brandumo lygmuo	2	2	2	3	3 / 4

Apibendrintus panašių produktų savybių analizės rezultatus galima pastebėti, jog pasiūlytas sprendimas yra pranašesnis už nagrinėtus analogus ir jis apeliuoja į mažiausiai trečiąjį, o dalinai ir į ketvirtąjį TMM lygmenį. Tai suteikia įrankiui „MiniTest“ pranašumo.

6.3. Taikymo rekomendacijos

Testavimo proceso valdymo sistemą „MiniTest“ rekomenduojama naudoti vartotojams, programinės įrangos kūrimo metu naudojantiems testavimo procesą ir norintiems jį susieti su kitais programinės įrangos gyvavimo ciklo etapais. „MiniTest“ ypač tinka klaidų sekimui bei valdymui, informacijos apie sistemos klaidas ir sistemos kokybę generavimui. Taip pat „MiniTest“ gali būti išplečiama ir naudojama viso programinės įrangos kūrimo proceso valdymui ir visos informacijos apjungimui.

7. Išvados

Darbo metu buvo išanalizuota programinės įrangos testavimo proceso specifika ir prieita tokių išvadų:

1. Išanalizavus programinės testavimo procesą nustatyta, kad programinės įrangos testavimo procesas turi būti integruotas į visą programinės įrangos kūrimo gyvavimo ciklą.

2. Programinės įrangos testavimo proceso analizė parodė, kad testavimas turi būti atliekamas visuose programinės įrangos kūrimo etapuose ir turi būti galimybė informaciją, reikalingą testavimo etapui, susieti su kituose programinės įrangos kūrimo etapuose esančia informacija.

3. Panašių testavimo įrankių analizė parodė, kad esami testavimo ir klaidų registravimo įrankiai negali integruoti viso programinės įrangos proceso arba jie yra per brangūs ir per galingi nagrinėjamai sričiai.

4. Išanalizavus testavimo procesą ir esamus įrankius jam valdyti, buvo nuspręsta sukurti sistemą, kuri integruotų duomenis iš visų PĮ kūrimo etapų ir susietų juos su testavimo etapu.

5. Sukurtas specializuotas įrankis automatizuoja testavimo proceso valdymą bei integruoja testavimo procesą į bendrą programinės įrangos kūrimo procesų visumą.

6. Įrankis ištestuotas ir jo pagalba atliktas eksperimentas, leidžiantis spręsti, jog „MiniTest“ padeda gerai valdyti programinės įrangos testavimo procesą ir automatizuoti šio proceso valdymą.

7. Pagal atliktą panašių sistemų ir sukurtos sistemos savybių analizę matoma, jog pastaroji pilnai patenkina mažiausiai trečiąjį testavimo brandumo modelio (TMM) lygmenį ir dalinai patenkina ketvirtąjį.

8. Pagal atlikto darbo rezultatus galima spręsti, kad sukurtu įrankiu galima valdyti programinės įrangos testavimo procesą bei automatizuoti jo valdymą ir kad sukurtas įrankis savo savybėmis yra pranašesnis už egzistuojančius panašius analogus.

8. Literatūra

[1] C.Kaner, J.Bach, B.Pettichord. Lessons Learned in Software Testing: a context-driven approach, Wiley Computer Publishing. 2001 m., p. 352.

[2] Rex Black, Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, Wiley Publishing, Inc., 2002 m., p. 528.

[3] IEEE Standard for Software Test. Documentation. IEEE Std 829-1998

[4] Metrics and Models in Software Quality Engineering, Second Edition, Stephen H. Kan, 2002

[5] Inefficiency and Ineffectiveness of Software Testing: A Key Problem in Software Engineering, Cem Kaner, Ph.D. (http://www.kaner.com/pdfs/Top5SEissues.pdf) (žiūrėta 2006 10)

[6] Illinois Institute of Technology - Testing Maturity Model Project (http://www.cs.iit.edu/research/tmm.html) (žiūrėta 2006 11)

[7] Ilene Burnstein, Taratip Suwannasart, and C.R. Carlson, llinois Institute of Technology - Developing a Testing Maturity Model: Part I (http://www.stsc.hill.af.mil/crosstalk/1996/08/developi.asp) (žiūrėta 2006 11)

[8] Ilene Burnstein, Taratip Suwannasart, and C.R. Carlson, llinois Institute of Technology - Developing a Testing Maturity Model: Part II (http://www.stsc.hill.af.mil/crosstalk/1996/09/developi.asp) (žiūrėta 2006 11)

[9] Master's thesis proposal by Brian Nielsen & Kim G. Larsen - Automatic Testing (http://www.cs.auc.dk/~bnielsen/testudbud/) (žiūrėta 2006 10)

[10] Software Process Resources http://www2.umassd.edu/SWPI/1docs/SPResource.html#scope (žiūrėta 2007 04)

[11] R.Hundhausen. Working With Microsoft Visual Studio 2005 Team System, Microsoft Press. 2006 m., p. 312.

[12] http://www.mantisbt.org/ (žiūrėta 2006 12)

[13] http://www.bugzilla.org/ (žiūrėta 2007 03)

[14] www.ets.org/testcoll/ (žiūrėta 2007 04)

[15] http://msdn2.microsoft.com/en-us/teamsystem/default.aspx (žiūrėta 2007 05)

[16] C.Kaner, J.Falk, H.Q.Nguyen. Testing Computer Software, 2nd Edition, 1999 m., p. 496.

[17] http://en.wikipedia.org/wiki/V-Model_(software_development) (žiūrėta 2007 09)

[18] http://en.wikipedia.org/wiki/Unit_test (žiūrėta 2007 12)

[19] http://en.wikipedia.org/wiki/Integration_testing (žiūrėta 2007 12)

[20] http://en.wikipedia.org/wiki/System_testing (žiūrėta 2007 12)

[21] http://en.wikipedia.org/wiki/Acceptance_testing (žiūrėta 2007 12)

9. Priedai

Straipsnis, publikuotas konferencijoje "Informacinė visuomenė ir universitetinės studijos 12" 2007 m. gegužės mėnesį

Programinės įrangos kūrimo ir testavimo valdymo tyrimas

Eivilė Jankevičiūtė

Kauno technologijos universitetas

Informacijos sistemų katedra, Studentų g. 50-309

Straipsnyje analizuojamos programinės įrangos kūrimo proceso problemos, koncentruojantis į testavimo etapą. Bus apžvelgiamas programinės įrangos testavimo proceso valdymo sistemos modelis, leidžiantis formalizuoti bei apjungti programinės įrangos testavimo procesą į vieną visumą, padedantis išsamiai aprašyti ir efektyviai valdyti testavimo procesą, į jį įtraukiant duomenis iš kitų etapų. Nagrinėjamos programinės įrangos testavimo proceso valdymo problemos.

1. Įvadas

Kiekvieną dieną pasaulyje sukuriama vis naujų programinės įrangos produktų. Programinės įrangos kūrimas yra iteratyvus procesas, kurio metu visi procese dalyvaujantys asmenys privalo harmoningai ir betarpiškai bendradarbiauti. [1] Šis bendradarbiavimas yra esminis visame PĮ kūrimo procese, o jo efektyvumas yra didžiausias produkto sėkmingumo garantas. Tačiau čia susiduriame su problema – nestruktūrizuotas bendradarbiavimas, bendravimas kūno kalba neatneš tokios sėkmės. Tai, ką prieš kelias minutes pasakė kolega, gali būti jau pasikeitę arba tai tiesiog pasimiršta. Kiekvienas proceso žingsnis, kiekviena iteracija, bet kokia smulkmena turi būti struktūrizuota ir formalizuota – taip bus išvengiama bereikalingų klaidų, bus galima efektyviai valdyti visą programinės įrangos kūrimo procesą, lengvai įtraukti naujus narius į komandos darbą bei sklandžiai vystyti visą projektą.

Kokybės užtikrinimas yra probleminė sritis, su kuria susiduria programinės įrangos gamintojai. Norint pasiekti programinių produktų kokybę, juos reikia nuodugniai ištestuoti. Tokiu būdu bus išgaunama aukšta produkto kokybė ir potencialų vartotoją pasieks patikimas produktas, atitinkantis jo poreikius.

Šiame straipsnyje nagrinėjamas programinės įrangos kokybės užtikrinimas bei testavimo proceso įjungimas į programinės įrangos kūrimo procesų visumą. Straipsnyje siūlomas sprendimas, kaip spręsti programinės įrangos testavimo proceso integravimo problemą.

Visų atskirų PĮ kūrimo procesų integravimas į vieną visumą ir, atvirkščiai, viso proceso suskaidymas į atskirus žingsnius, leidžia objektyviau įvertinti projekto situaciją, efektyviau valdyti kiekvieno komandos nario darbą, paskirstyti užduotis, kaupti informaciją, duomenis, kurių bet kada gali prireikti. Kaip jau buvo minėta, šiame straipsnyje bus koncentruojamasi ties programinės įrangos testavimo procesu. [2] Deja, remiantis daugelio programinės įrangos kūrimo įmonių patirtimi, šis etapas dažniausiai būna labiausiai pamirštas procesas visoje programinės įrangos kūrimo procesų aibėje.

Testavimo procesui valdyti yra sukurta įvairių programinės įrangos sistemų. Kai kurios iš esmės yra panašios savo savybėmis ir funkcionalumu. Tačiau tarpusavyje jos skiriasi tuo, kad apima tik tam tikrą testavimo proceso dalį ir neturi galimybių pilnai valdyti visą procesą.

2. Programinės įrangos testavimui naudojamų priemonių lyginamoji analizė

Šiame skyriuje pateikiama atlikta lyginamoji analizė ~~kitų~~ egzistuojančių produktų, turinčių vienas ar kitas nagrinėjamas savybes.

1.1. Egzistuojantys specializuoti produktai

Įvairūs testavimui specializuoti produktai dažniausiai apima tik tam tikrą siaurą sritį ir įmonėse dažniausiai naudojami tam tikrai informacijai bei duomenis apdoroti kaip pagalbinės priemonės. Šiame skyriuje palygintos kelios programos, dažniausiai naudojamos nedidelėse įmonėse: *Mantis* (klaidų sekimo programa), *Bugzilla* (internetinė klaidų registravimo sistema), *Test Link* (testų rašymo programa), *Win CVS* (versijavimo valdymo sistema).

***Mantis* – klaidų sekimo internetinė programa [4].** Ji leidžia užregistruoti surastas klaidas, joms priskirti statusą, svarbumo lygį, kategoriją, atkartojamumą, nukreipti konkrečiam darbuotojui, pridėti žinutę (angl. bugnote), prisegti failą, susieti klaidas, filtruoti pagal tam tikrus kriterijus. Vartotojo sąsaja yra patogi, maloni ir aiški. Tačiau šioje sistemoje negalima rašyti testavimo atvejų, kurti testavimo planų, rašyti ataskaitų, generuoti diagramų apie produkto kokybę. Joje galima matyti tik statistiką apie visas užregistruotas klaidas įvairiais pjūviais. Tačiau šis produktas yra nemokamas.

***Bugzilla* – internetinė klaidų registravimo sistema [5].** Kaip ir *Mantis*, ši sistema leidžia registruoti klaidas, joms priskiriant statusą, svarbumo lygį, kategoriją, atkartojamumą, nukreipti konkrečiam darbuotojui, pridėti žinutę, prisegti failą, susieti klaidas, matyti statistiką. Ji taip pat neturi testavimo planų valdymo funkcionalumo. Bet šis produktas taip pat yra nemokamas.

***Test Link* – testų rašymo programa [6].** Ji leidžia rašyti arba įkelti programos testus, kurti testavimo atvejus, rašyti testavimo žingsnius. Tačiau šios programos vartotojo sąsaja nėra labai patogi, turi nemažai trūkumų. Taip pat joje negalima atlikti klaidų sekimo kontrolės. Bet produktas - nemokamas.

Apibendrinus galima pasakyti, kad visos trys nagrinėtos programos yra specializuotos, tam tikrai siaurai sričiai skirtos, puikiai išpildančios įmonės reikalavimus programos, kurių vienintelis trūkumas – integracijos su kitais procesais ar jų dalimis stygius. Tačiau, jei nėra vieno integruoto produkto, kuris leistų valdyti visą testavimo procesą ir bendradarbiauti šio proceso aktoriams su kitų procesų aktoriais, visas procesas tampa nenuoseklus, nekontroliuojamas, trumpiau tariant – neefektyvus.

Apžvelgus keletą specializuotų programų buvo prieita išvados, kad įmonėje vieno produkto kūrimui naudoti daug atskirų programų yra nepatogu. Taip neišvengiama klaidų, atsirandančių dėl produktų nesuderinamumo bei nepatogumo darbuotojams, dirbantiems prie projekto, atsirandančio dėl nestruktūrizuotos informacijos nesuderinamumo. Dėl to atsiranda poreikis turėti integruotą informaciją, apimančią viso programinės įrangos kūrimo proceso eigą. Vienas iš tokių realizuotų produktų yra *Microsoft Team System*.

***Microsoft Team System* – tai visos gyvavimo grandinės (*lifecycle*) valdymui, užduočių (*tasks*), klaidų (*bugs*), versijavimo kontroliavimui skirtas produktas [7].** Jame yra galimybė aprašyti savo programinės įrangos kūrimo procesus, vesti projekto valdymą, valdyti programos kodo versijavimą bei atlikti testavimą. *Visual Studio Team System* galima sugeneruoti modulinius testus, atlikti internetinių aplikacijų testavimą ir vykdyti apkrovimo testus. [3]

Taciau *Team System* yra visiškai integruotas su *Visual Studio* projektais, todėl yra sunkiai pritaikomas kitiems projektams. Jis reikalauja daug žinių ir yra sunkiai suprantamas. Be to, produktas yra mokamas, netgi labai brangus.

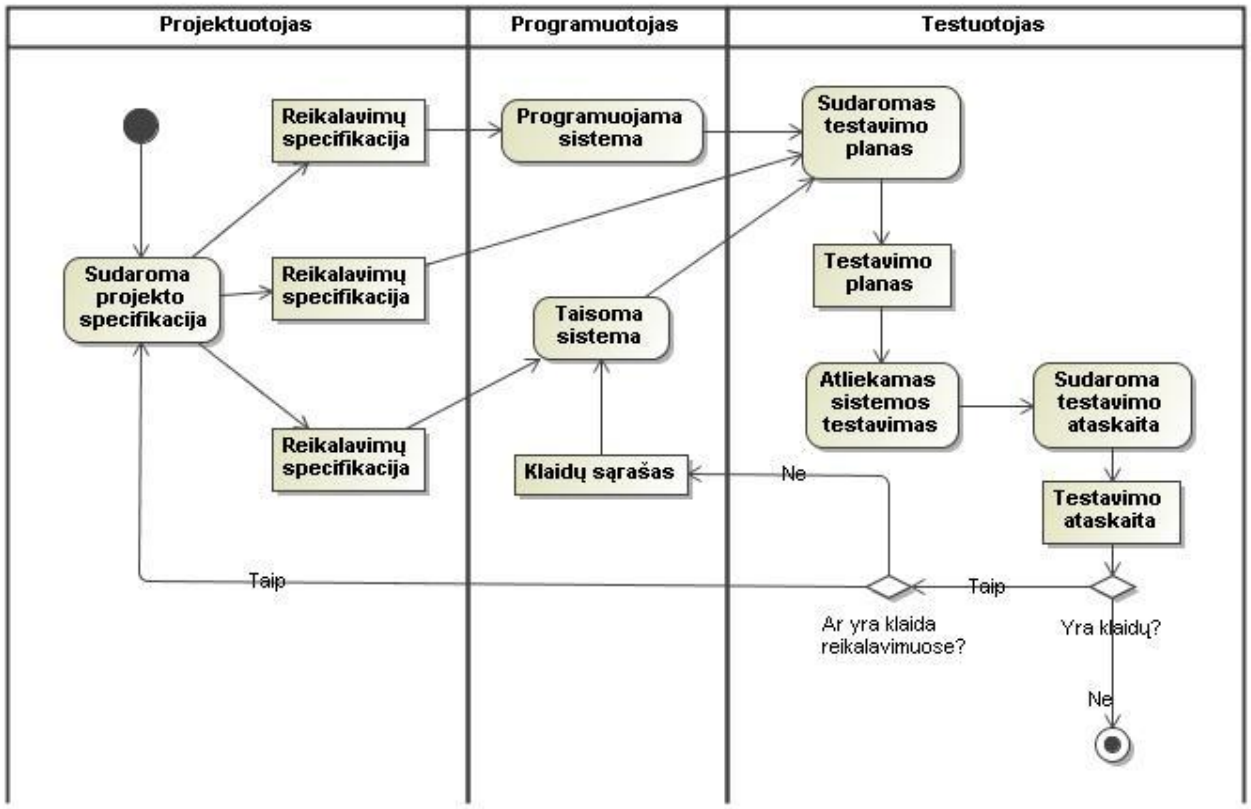
3. Siūlomas sprendimas

Išanalizavus alternatyvius produktus, paaiškėjo, kad pirmieji neapima viso testavimo proceso, o paskutinis Microsoft'o produktas yra be galo didelis, plačiai apimantis visą programinės įrangos kūrimo procesą ir yra labai brangus. Taigi yra poreikis produktui, kuris apimtų daugelį naudojamų programinės įrangos testavimo proceso etapų ir leistų integraliai testuoti programinę įrangą.

PĮ kūrimo procesą pagerintų sistema, apjungianti projekto valdymo, analizės, projektavimo, realizacijos, testavimo bei dokumentavimo etapus (1 pav.).

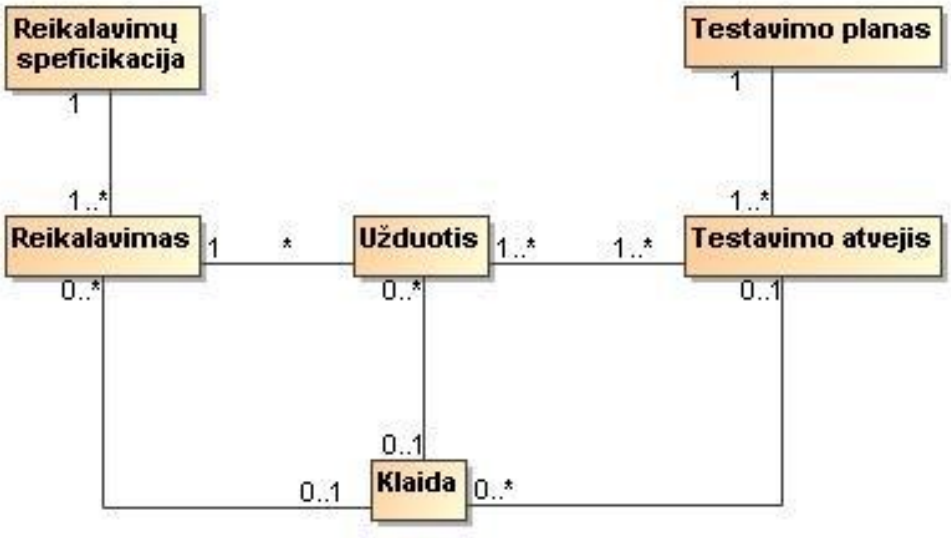
1 pav. PĮ kūrimo proceso valdymo sistemos modelis

Šiame modelyje kiekvieno etapo dalyvis turi savo individualias užduotis, kurios turi sąryšį su kitais etapais. Projektų vadovas gali pradėti naujus projektus, suskirstyti užduotis visiems darbuotojams, valdyti PĮ kūrimo procesą. Analitikai gali specifikuoti reikalavimus, kuriuos projektuoja projektuotojai. Pagal suprojektuotus modelius programuotojai gali projektą realizuoti. Visa tai testuoja testuotojas, kuris iš reikalavimų gali sudaryti testavimo atvejus, o ištestavęs gali pateikti ataskaitas apie projekto kūrimo eigą ir kokybę. Iš reikalavimų, ataskaitų ir rezultatų dokumentuotojai gali dokumentuoti produktą. Tačiau ši sistemos integracija yra paliekama kaip išplėtimo galimybė, o realizuojamas testavimo etapas. Kaip yra vykdomas veiklos procesas, matome 2 pav.:



pav. 2 Programinės įrangos testavimo proceso veiklos diagrama

Projektavimo ir realizacijos dalyje turėtų būti registruojamos užduotys, iš kurių būtų galima sudaryti testavimo atvejus. Kiekviena užduotis gali būti susieta su vienu ar daugiau testavimo atvejų, arba gali būti parašomi tokie testavimo atvejai, kurie apimtų kelias užduotis. Pagal testavimo atvejus gali būti registruojamos klaidos. Norint ištaisyti tas klaidas, sukuriamos naujos užduotys, pagal kurias kuriami nauji testavimo atvejai. Klaidos gali iš naujo įtakoti reikalavimus tokiu atveju, jeigu jie yra nekorektiški. Toks modelis gali būti naudojamas ir projekto palaikymui. Sąsajos tarp šių esybių pavaizduotos klasių diagramoje 3 pav.:



pav. 3 Supaprastinta sistemos klasių diagrama, sauganti informaciją apie procesus

Lyginant siūlomą sprendimą su analizuotais, galima pastebėti, kad *Mantis* ir *Bugzilla* apima tik klasę „Klaida“, o *Test link* – „Testavimo atvejis“.

Siekiamas sistemos privalumas turi būti integralumas, leidžiantis tiksliai, betarpiškai ir vieningai dirbti siekiant rezultato. Taip suprojektuota sistema leistų visiškai stebėti sąsajas tarp visų etapų: t.y., galima būtų matyti klaidų įtaką darbams, planuoti kaštus. Galima tokios sistemos rinka yra nedidelės programinės įrangos kūrimo ir plėtojimo įmonės, savo veiklos eigoje naudojančios testavimo procesą. Naudojantis šia sistema, būtų galima dirbti bet kokioje aplinkoje, priešingai negu *Microsoft Team System*, kuriai yra reikalinga *Visual Studio* aplinka.

4. Apibendrinimas ir tolimesni darbai

Išanalizavus programinės testavimo procesą ir atlikus alternatyvių produktų lyginamąją analizę nustatyta, kad reikia integruoto įrankio, kuris leistų valdyti informaciją viso testavimo proceso metu. Dabartiniai sukurti įrankiai neatitinka visų reikalavimų arba yra labai brangūs. Todėl pasiūlytas modelis sistemos, išpildančios straipsnio metu nagrinėtus reikalavimus, tokius kaip integralumas bei integruotas testavimo procesas.

Vykdant magistrinį tiriamąjį darbą, bus realizuotas programinės įrangos testavimo etapas ir su juo susijęs funkcionalumas. Integracija su kitais etapais bus palikta kaip išplėtimo galimybė tolimesniems darbams.

Literatūros sąrašas

[1] **C.Kaner, J.Bach, B.Pettichord.** Lessons Learned in Software Testing: a context-driven approach, *Wiley Computer Publishing*. 2001 m., p. 352.

[2] **R.Black.** Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, *Wiley Computer Publishing.* 2002 m., p. 528.

[3] **R.Hundhausen.** Working With Microsoft Visual Studio 2005 Team System, *Microsoft Press.* 2006 m., p. 312.

[4] http://www.mantisbt.org/

[5] http://www.bugzilla.org/

[6] www.ets.org/testcoll/

[7] http://msdn2.microsoft.com/en-us/teamsystem/default.aspx

Analysis of Software Developing and Testing Management

In this paper software development process problems are analyzed, highlighting the importance of software testing process. Software testing process management system model is proposed which enables integrate whole software testing process and efficiently manage testing process collecting information from other stages. Software testing process management problems are analysed.