KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Andrius Borkauskas

Testinių atvejų generavimo sistema. Kūrimas ir tyrimas

Magistro darbas

Darbo vadovas:
prof. E. Bareiša

KAUNAS, 2009

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Andrius Borkauskas

Testinių atvejų generavimo sistema. Kūrimas ir tyrimas

Magistro darbas

Recenzentas:
E. Karčiauskas
2010-05-31

Vadovas:
prof. E. Bareiša
2010-05-31

Atliko:
IFM-3/4 gr. stud.
Andrius Borkauskas
2010-05-31

Kaunas, 2010

Turinys

Summary
Test case generation system. Development and research

This master‘s degree work consists of three main parts: software development methods review, DisksDB system requirement specification and summary and results analysis of performed unit testing.

In the first part of this work software development techniques are compared. Automated testing tool .Test is introduced.

In the second part of this work main parts of program to be tested are introduced, main requirements and their descriptions are presented. Main data of the system that was used in testing is being introduced.

In the third part of this work unit testing results got by using automated testing Parasoft .Test are presented.

All the tests results are summarized in the conclusion.

Santrauka
Testinių atvejų generavimo sistema. Kūrimas ir tyrimas

Šis magistrinis darbas susideda iš trijų pagrindinių dalių: programinės įrangos (PĮ) kūrimo ir jos automatinio testavimo metodikų apžvalgos, DisksDB sistemos realizavimo ir reikalavimų specifikacijos, bei gautų kodo modulių testavimo rezultatų analizės apibendrinimo.

Pirmojoje dalyje pateikiamas programinės įrangos (*PĮ*) kūrimo metodikų palyginimas. Supažindinama su testavimo įrankiu .Test.

Antroje dalyje pateikiama analizuojamos sistemos pagrindinai reikalavimai ir jų aprašymai. Pateikiami pagrindinai realizuotos sistemos duomenys, kurie naudojami testavimo metu.

Trečioje dalyje analizuojami kodo modulių testavimo atlikto automatinio testavimo įrankiu Parasoft .Test rezultatai.

Visų gautų rezultatų apibendrinimas pateikiamas išvadose.

1 Įvadas

PĮ kūrimas yra sudėtingas procesas, kuris privalo būti itin efektyviai valdomas, norint pasiekti rezultatą pagal aukščiausius reikalavimus. Kuriant PĮ dirbama komandoje. Programuotojams kas dieną keliant vis nauja programinio kodo versiją tenka kas kart tikrinti ne tik pateikiamo kodo korektiškumą, bet ir keliamo kodo teisingą sąveiką su iš anksčiau veikiančia kodo baze į kurią jis keliamas. Tiek iš kiekvieno programuotojo ateinantis kodas, tiek tie kodai sujungti kartu turi būti be klaidų.

Kada kuriamos Programinės Įrangos (PĮ) kodo apimtis yra didesnė nei žmogus pajėgus ištestuoti rankiniu būdu, pasitelkiamos automatinio testavimo priemonės. Tai padeda vykdyti kaip galima visapusiškesnę programinio kodo kokybės kontrolę.

Pagrindinės testavimo charakteristikos:

- Kokybės įvertinimas:
  - ✓ Klausimas: „kada turi būti sustota?“. Naudojamas “klaidų tankumo” matas;
  - ✓ Klaidų kiekis / Sistemos apimtis;
  - ✓ Kiek liko neatrasta klaidų, remiantis “klaidų išsiskaidymu”;
- Rizikos valdymas:
  - ✓ Metodai, naudojami rizikos valdyme, nustato kiek svarbus yra (gali būti) konkretus defektas.
  - ✓ Vienas iš metodų “Risk-driven method” – kiekvienas komponentas vertinamas pagal griežtumo (kas atsitiks jei komponentas blogai veikia) ir galimumo (tikimybė, kad komponentas neveiks) kriterijus.
- Populiariausi matai:
  - ✓ Efektyvumas, tai rastų ir sėkmingai pašalintų defektų skaičiaus ir esamų defektų skaičiaus santykis
  - ✓ Produktyvumas, tai defektų skaičius rastas per duotą laiką

Testavimai, kurie atliekami pabaigus kurti PĮ, prieš išsiunčiant užsakovui, vykdomi nepriklausomų testuotojų grupių. Projekto testavimo fazę įprasta naudoti kaip projekto laiko atsargos buferį vėlavimams kompensuoti, taip testavimui skirtas laikas labai sutrumpėja. Rekomenduojama vadovautis kita metodika ir pradėti vykdyti testavimą nuo projekto pradžios iki jo pabaigimo.

Testinius atvejus generuojantis testavimo įrankis būtinas siekiant gauti teisingą ir gerai veikiantį kodą, nes toks korektiškumo tikrinimas nevykdomas paprastose derinimo programose (angl. *Debugger*) pagal nutylėjimą.

Šiame darbe apžvelgsime programos kuriamos ekstremalaus programavimo būdu automatinį testavimą, pagrindines problemas ir jų sprendimo būdus.

Darbo tikslas ištirti automatinio testavimo specifiką ekstremalaus programavimo kūrimo procese. Tam naudojamas .TEST ir integruotas naudojamos IDE įrankis, kuriame kuriamos kodo modulių testavimo taisyklės. Šios taisyklės buvo kuriamos prieš kuriant DisksDB sistemą. Atlikus kodo testavimą, buvo sugeneruoti šablonai, kurie rankiniu būdu buvo supildyti duomenimis, reikalingais atlikti kodo modulių testus. Testuojant kodą įrankiai pateikė testavimo metrikas, kurios buvo išanalizuotos. Šios analizės rezultatai taip pat pateikti šiame darbe.

Atsižvelgus į analizės rezultatus, buvo pakeistas DisksDB kodas. Daugiausia dėmesio buvo skirta kritiniams pranešimams.

Darbo uždaviniai:

1. Apžvelgti kodo modulių testavimo metodus;
2. Atlikti kodo modulių testavimą;
3. Suprojektuoti pasirinktos programinės įrangos architektūrą;
4. Atlikti programinės įrangos testavimą, naudojant automatinio testavimo įrankį;
5. Apžvelgti rezultatus ir užfiksuoti problemines kodo modulių testavimo vietas.

Problema. Pagrindinis dažnai pasitaikantis kuriamų sistemų trūkumas – kokybės stoka. Nedidelėms sistemoms ją užtikrinti nėra itin sunku, tačiau jei kalbama apie dideles sistemas, kokybės užtikrinimo klausimas yra itin aktualus ir svarbus, kadangi jos užtikrinimu reikia pradėti rūpintis ankstyvoje projektavimo fazėje.

Sistemos sutrikimų taisymas vėlyvoje kūrimo fazėje, įmonei gali atnešti didžiules išlaidas, todėl svarbu laiku nustatyti galimus sutrikimus. Testavimas yra vienas iš būdų užtikrinančių programinės įrangos kokybę. Tačiau žmogui tai atlikti gali būti sudėtinga, nes jis dažnai susiduria su tokiomis kliūtimis kaip, laiko apribojimai, ribota aptarnavimo apimtis. Kompiuteriai gali atlikti daugelį užduočių, su kuriom žmogus nesusitvarko.

Kita svarbi problema, kuri dažniausiai pasitaiko didelėse kompanijose - tai parašyto kodo vientisumas. Tai reiškia, kad programuotojams rašant programinį kodą, turi būti laikomasi bendrų taisyklių, kurios nustatytos kompanijos viduje.

Sukurta sistema „DisksDB“ šių problemų tai pat neišvengė. Tačiau ši sistema nėra didelė, o jos klaidų taisymas nesukeltų didelių finansinių nuostolių, tačiau tai pareikalautų papildomų laiko sąnaudų.

100% kodo padengimas ir sudėtingų duomenų tipų testavimas naudojant testinius atvejus generuojančius įrankius.

2 Programinės įrangos kūrimas ir testavimas

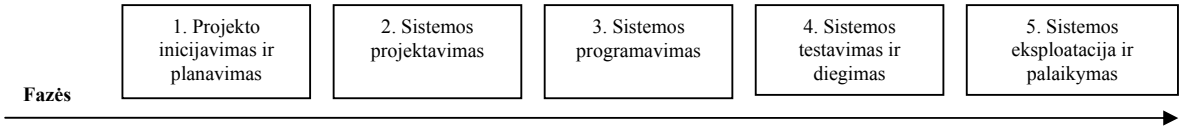
Analizės tikslas yra pažinti ekstremaliame programavime (**XP**) naudojamo programos kodo modulių testavimo specifiką. Surasti kokios priemonės naudojamos automatizuoti šį procesą.

2.1 Agile

Pirmosios programinės įrangos kūrimo metodikos buvo net ne metodikos, o tik būdas organizacijoms iš kompiuterinių technologijų išgauti naudą. Tačiau kai buvo daugiau žinoma apie programinės įrangos kūrimą, atsirado tam tikri būdai valdyti ir nuspėti programinės įrangos kūrimo kaštus. Kelis dešimtmečius dominavo krioklio metodas [7].

Krioklio PĮ kūrimo metodas nusako, kad sistemos turi būti sukurtos nuosekliai vykdant visų etapus, pradedant reikalavimų analize ir baigiant įdiegtos sistemos palaikymu (žr. *1 pav.*). Tai reiškia, kad sudėtingos sistemos turi būti sukuriamos iš pirmo karto, todėl išimtinai Krioklio modeliu grindžiamuose projektuose vadovaujamasi principu „viskas turi buti gerai iš pirmo karto“ (angl. *everything is right the first time*)

Šiuo metu Krioklio modelis naudojamas sistemoms kurti, kur reikalavimai sistemai yra gerai žinomi ir apibrėžti dar prieš pradedant patį sistemos kūrimo procesą.



1 pav. *Nuoseklios PĮ kūrimo fazės, pagal tradicinį krioklio modelį*

Pagrindiniai Krioklio modelio trūkumai:

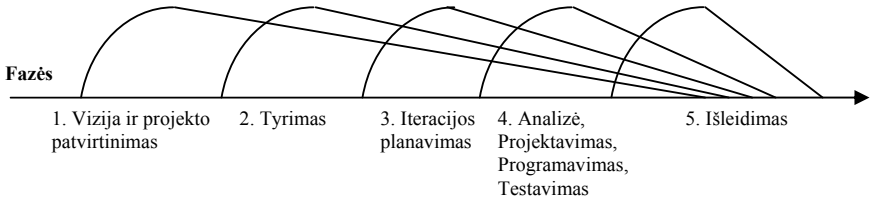
- Brangus ir ilgai trunkantis procesas (jei pasikeičia užsakovų reikalavimai);
- Didelės dokumentacijos apimtys;
- Nėra užtikrinamas grįžtamasis ryšys (angl. *feedback*) tarp sistemos kūrimo etapų;
- Sudėtingas projekto valdymas. Planavimo, analizės ir projektavimo metu kuriama tik dokumentacija, todėl sunku kontroliuoti projekto vykdymo terminus;
- Maksimaliai ilgas laiko intervalas tarp projekto inicijavimo ir sistemos sukūrimo bei įdiegimo;
- Užsakovas suprantamas tik kaip žinių šaltinis vartotojo poreikiams ir reikalavimams sistemai nustatyti ankstyvajame analizės etape;
- Užsakovas gali įvertinti sistemą tik atlikus jos diegimą;

- Tik realizavus ir įdiegus sistemą galima identifikuoti jos trūkumus ir klaidas – šiame projekto vykdymo etape tai atliekama su maksimaliom išlaidom ir papildomo laiko sąnaudom.

Vietoj griežtų (negalima praleisti ar keisti eilės tvarką iš anksto numatytų projektavimo ar realizavimo žingsnių), vienakrypčių (nėra galimybės sugrįžti į prieš tai buvusius projektavimo/realizavimo etapus), inkrementinių (kuriamos prototipų versijos iki gaunamas galutinis produktas) PĮ kūrimo technologijų reikėjo lankstesnių, pigesnių įvykdyti (kur galima būtų tiksliau įvertinti laiko apimtis ir kaštus projektui atlikti), pigesnių vykdyti pakeitimus PĮ kūrimo būdų, tai buvo paskata naudoti kitus metodus. Šie metodai buvo apibūdinami kaip „smulkieji metodai“ (angl. *lightweight*), vėliau juos imta vadinti „Agile metodais“.

Galimybė pakartotinai grįžti į ankstesnes PĮ kūrimo fazes ženkliai pagerina projekto efektyvumą. Šis grįžimo procesas vadinamas inkrementinis ir iteracinis kūrimas (angl. *Incremental and Iterative Development (IID)*). Kūrimo gyvavimo ciklas yra suskaidomas į inkrementus arba iteracijas, o kiekviena iteracija turi tradicinio krioklio PĮ kūrimo modelio fazes.

Kiekviena fazė tai sluoksnis, kuris tęsiasi visą gyvavimo ciklą (*2 pav.*). Reikalavimų, kūrimo ir diegimo ciklas kiekvieną kartą yra pereinamas pagal nustatytas gaires. Šis lygiagretus PĮ kūrimas suteikia galimybę eksperimentuoti bandymų ir klaidų metodu bei siekti efektyvių naujovių



2 pav. *Persidengiančios PĮ kūrimo fazės, pagal Agile metodologiją*

Agile vertybės

Individualizmas ir bendravimas procesuose ir priemonėse (angl. *Individuals and interactions over processes and tools*). Tradiciniuose PĮ kūrimo metoduose linkstama prie procesų, kuriuose yra atskiriamos rolės. Šie procesai akcentuoja roles, o ne individus, atliekančius tam tikros rolės užduotis. Kad projektas būtų sėkmingai vykdomas, yra tikima, kad jei koks nors asmuo nusprendė palikti projektą, jo rolę gali užimti koks nors kitas. Gamybos procese toks mąstymas gali būti patogus, tačiau kūrybiniame darbe, tokiame kaip PĮ projektavimas, yra akcentuojamas individualizmas ir asmeninė kompetencija dėl to yra sunku pakeisti individą. Agile metoduose toks rolių atskyrimas nėra priimtinas. Juose pabrėžiami

individualūs asmenys bei tarpininkavimas tarp jų. Tačiau, kad projektas sėkmingai būtų pradėtas, reikalaujami procesų aprašymai bei organizacijos sudėtis.

Išsami veikiančios PĮ dokumentacija (angl. *Working software over comprehensive documentation*). dokumentuoti reikalavimai, analizė ar planas gali būti naudingi kūrėjų darbui bei padėti numatyti perspektyvas, o ištestuotas ir be klaidų veikiantis programinis kodas parodo kokia yra kūrėjų komanda, kūrimo procesas bei kokia kylančių problemų, kurias reikia išspręsti, priežastis. Todėl yra tvirtinama, kad veikianti PĮ yra vienintelė patikima priemonė nusakyti komandos greitį ir trūkumus bei suteikia įžvalgą į tai, ką komanda iš tikrųjų turėtų kurti. Dėl to kad Agile metoduose naudojami tam tikro mąsto dokumentai ir planai, vyrauja nuomonė, kad „Agile metodai dažnai atrodo mažiau suplanuoti negu iš tikrųjų yra“.

Užsakovo bendradarbiavimas derinant sutarties sąlygas (angl. *Customer collaboration over contract negotiation*). Pabrėžiamas glaudus bendravimas tarp PĮ kūrimo komandos ir užsakovo. Bendravimas siejamas su bendrumu, draugiškumu, vieningų sprendimų priėmimu, bendravimo sparta bei ryšiu su individų sąveika. Dėmesys į užsakovo bendradarbiavimą parodo draugiškus ryšius tarp organizacijų bei profesijų. Manoma, kad našus ir artimas bendradarbiavimas padeda išvengti sutarčių, o jeigu sutartis yra rizikingoje situacijoje, geri santykiai gali išgelbėti. Patenkintas užsakovas yra pagrindinis rodiklis PĮ kūrime pagal Agile metodus. Tuo atveju jei PĮ kūrimas užsakovui nėra prioritetas arba jom nepakanka žinių apie dalykinę sritį, tuomet vietoj Agile metodų geriau naudoti tradicinius.

Atsakas į vykdomo plano pakeitimus (angl. *Responding to change over following plan*). naujos sistemos reikalavimai pilnai nebus žinomi, kol vartotojas jos nepanaudojo. Yra žinoma, kad PĮ kūrime yra būdingas ir neišvengiamas neapibrėžtumas. Dažniausiai reikalavimai keičiasi dėl sistemos neapibrėžtumo ir interaktyvumo bei dėl nepastovios verslo ir technologijų aplinkos. Šie pakeitimai įtraukiami į PĮ kūrimo procesą. Agile metodų planavimo fazėje yra naudojami reikalavimų keitimo mechanizmai. Dėl to sistemos kūrimo komandos nuolat lanksčiai turi keisti planą pagal esamą situaciją.

Agile principai

1. Klientas turi būti patenkintas teikiamais rezultatais;
2. Galima keisti reikalavimus, net ir vėlyvoje kūrimo stadijoje;
3. Dažnai pateikti veikiančias PĮ versijas, kas kuo trumpesnį laiko tarpą (maždaug nuo kas dviejų savaičių iki kas dviejų mėnesių);
4. Užsakovai ir PĮ kūrėjai kasdien turi dirbti drauge viso projekto metu;
5. Projektą vykdyti motyvuotų žmonių komandoje. Suteikti jiems aplinką ir pagalbą, kurios jiems reikia ir jais pasitikėti;

6. Pats efektyviausias ir veiksmingiausias būdas perteikti informaciją tarp komandos narių yra bendravimas akis į akį;
7. Veikianti PĮ yra pirminis judėjimo į priekį rodiklis;
8. Agile procesai palaiko nuoseklų PĮ kūrimą. Rėmėjai, kūrėjai ir vartotojai neribotai turi palaikyti pastovų tempą
9. Pastovus dėmesys į kokybę, techninį meistriškumą ir gerą projektavimą padidina lankstumą;
10. Paprastumas – minimizuoti atliekamo darbo kiekį yra būtina. Kūrėjai turi įgyvendinti tik tas savybes, apie kurias buvo kalbėta su užsakovais.
11. Geriausios struktūros reikalavimai ir planai gimsta organizuotose komandose;
12. Pastoviais laiko intervalais komanda suderina savo veiksmus ir nusprendžia kaip darbą paversti efektyvesniu.

1 lentelė. *Aglie manifesto vertybių ir principų priklausomybė [4]*

Vertybės	**Principai**											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Individualizmas ir bendravimas procesuose ir priemonėse	+	-	-	+	+	+	-	+	-	-	-	+
Išsami veikiančios PĮ dokumentacija	+	-	+	-	-	-	+	-	+	+	+	-
Užsakovo bendradarbiavimas derinant sutarties sąlygas	+	+	-	+	-	-	-	+	-	-	-	-
Atsakas į vykdomo plano pakeitimus	+	+	+	-	-	-	+	-	-	-	-	(+)

2.2 Agile PĮ kūrimo metodų apžvalga

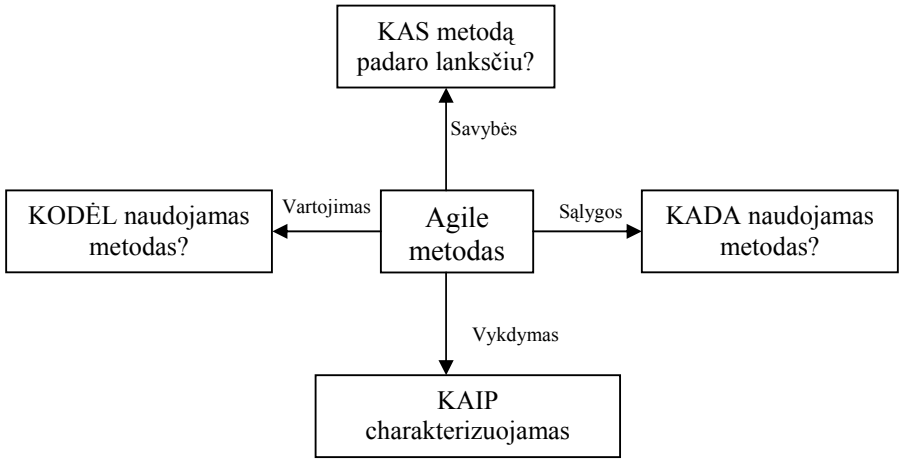
Agile metodus naudoja daugelis įmonių. Kurių veikla susijusi su PĮ kūrimu. Šie metodai pritaikyti prie dažnai besikeičiančių reikalavimų bei griežtų grafikų. Taip pat didelis dėmesys skiriamas užsakovo patenkinimui bei klaidų kiekio mažinimui. Žemiau pateikti pagrindiniai Agile metodai (*2 lentelė.*). Visi šie metodai paremti agi principais bei vertybėmis bei kiekvienas turi savo gyvavimo ciklą.

2 lentelė. *Trumpas Agile pagrindinių metodų programinės įrangos kūrimo ciklų aprašymas[1][2]*

Metodas	Fazė	Aprašymas
XP	Tyrimas	Einamos iteracijos scenarijus
	Iteracijų planavimas	Scenarijų prioritizavimas, pastangų ir resursų įvertinimas
	Versijos išleidimo iteracijos	Analizė, projektavimas, programavimas, testavimas
	Gamyba	Intensyvus testavimas
	Palaikymas	Techninis palaikymas, versija vartojimui
SCRUM	Pasiruošimas	Vizijos, pradinio produkto darbų sąrašo (angl. backlog) ruošimas, pastangų įvertinimas, pradinio modelio ir prototipo sukūrimas, reikalavimų prioritizavimas.
	Kūrimas	Iteracinis programos kūrimas (angl. Sprints), iteracijų planavimas ir analizė, rezultatų demonstracijos
	Pridavimas	Sistemos testavimas, integracijos testavimas, dokumentacija, diegimas, mokymai.
	Bendro modelio sudarymas	Reikalavimų dokumentavimas (panaudojimo atvejai, funkcinis specifikavimas), srities suskaidymas, eigos scenarijų ir suskaidytų sričių objektinių modelių sudarymas komandose.
FDD	Savybių sąrašo sudarymas	Paruošiamas sistemos savybių ir funkcijų sąrašas bei vykdoma jų patikra.
	Planavimas pagal savybes	Savybių suskirstymas pagal prioritetus ir priklausomybes bei priskyrimas programuotojams, terminų nustatymas.
	Projektavimas ir kūrimas pagal savybes	Savybių grupės išrinkimas iš savybių rinkinio, savybių realizavimas iteraciniu būdu (projektavimo patikra, programavimas, kodo modulių testavimas, integracija, kodo patikra).
	Galimybių tyrimas	Įvertinamos sistemos galimybės
DSDM	Operacijų tyrimas	Analizuojamos svarbios operacijų ir technologijų charakteristikos
	Funkcinis modelio iteracijos	Analizė funkcionalumo prioritizavimas, nefunkciniai reikalavimų ir rizikos įvertinimas
	Projektavimo ir kūrimo iteracija	Sistemos kūrimas ir testavimas
	Diegimas	Mokymai
	Mąstymas	Projekto iniciavimas, adaptyvus ciklo planavimas
ASD	Bendradarbiavimas	Lygiagretus komponentų kūrimas
	Mokymasis	Klientų atsiliepimai, testavimas, versijos išleidimas

2.3 Agile PĮ kūrimo metodų palyginimas

Agile PĮ kūrimo metodus galima palyginti pagal tai kokios metodų savybės [3] bei kodėl, kaip ir kada jie yra naudojami (*3 pav.*).



3 pav. *Agile metodų palyginimo kriterijai*

Pasirinkus kurį nors metodą, norisi gauti kokią nors naudą bei žinoti kodėl jis buvo pasirinktas. Žemiau pateikiamos Agile metodų savybės, kurios ir nulemia metodų pasirinkimą (*3 lentelė.*).

3 lentelė. *Agile metodų palyginimas pagal teikiamus privalumus*

Teikiami privalumai	ASD	AM	Crystal	DSDM	XP	FDD	PP	Scrum
Pristatymo terminų svarba	+	-	+	+	-	+	+	+
Reikalavimų svarba	+	+	+	+	+	+	-	+
Kokybės svarba	+	-	-	-	-	-	+	-
Galutinio vartotojo patenkinimas	-	-	-	+	-	-	-	-
Nerami aplinka	-	+	-	+	+	-	+	+
Galima išorinė pagalba	+	-	+	+	-	-	-	+
Gautas produktyvumas	-	+	-	-	+	-	+	-

Lankstumo parametrais nurodoma kas metodą padaro lanksčiu (*4 lentelė.*). Visi parametrai pagal kuriuos lyginami metodai atitinka bendrą Agile metodologijos idėją.

4 lentelė. *Agile metodų palyginimas pagal lankstumą*

Lankstumas	ASD	AM	Crystal	DSDM	XP	FDD	PP	Scrum
Trumpos iteracijos	-	+	-	-	+	+	+	+
Bendradarbiavimas	+	+	+	+	+	-	+	-
Orientuotas į komandos narius	+	+	+	+	+	-	-	-
Pertvarkymo vykdymas	-	+	-	+	+	-	+	-
Testavimo vykdymas	+	-	-	+	+	-	+	+
Pakeitimų integravimas	+	+	-	+	+	-	+	+
Supaprastinta metodika	-	+	-	-	+	+	+	+
Galimybė keisti funkcinius reikalavimus	+	+	-	+	+	-	+	+
Galimybė keisti nefunkcinius reikalavimus	+	-	-	-	-	-	-	-
Galimybė keisti darbo planus	-	+	-	-	+	-	+	-
Galimybė keisti komandos narius	+	+	+	-	+	-	-	-
Pokyčių indeksavimas	-	-	-	-	+	-	-	-
Reaguojama į	Gairę	Iteraciją	Gairę	Iteraciją	Iteraciją	Projekto pradžią	Iteraciją	Gairę
Žinių dalijimasis	Aukštas	Aukštas	Aukštas	Žemas	Aukštas	Žemas	Žemas	Žemas

Pagal bendras aplinkos sąlygas nustatoma, kada tikslinga naudoti Agile metodą (*5 lentelė.*).

5 lentelė. *Agile metodų palyginimas pagal aplinkos sąlygas*

Sąlygos	ASD	AM	Crystal	DSDM	XP	FDD	PP	Scrum
Projekto dydis	Didelis	Mažas	Didelis	Didelis	Mažas	Didelis	Mažas	Didelis
Projekto sudėtingumas	Aukštas	Žemas	Aukštas	Aukštas	Žemas	Aukštas	Žemas	Aukštas
Projekto rizikingumas	Aukštas	Žemas	Aukštas	Aukštas	Žemas	Aukštas	Žemas	Aukštas
Komandos dydis	Didelė	Maža	Maža	Didelė	Maža	Didelė	Maža	Maža
Bendravimas su užsakovais	Aukštas	Aukštas	Žemas	Aukštas	Aukštas	Žemas	Žemas	Žemas
Bendravimas su galutiniais vartotojais	Žemas	Žemas	Žemas	Aukštas	Žemas	Žemas	Žemas	Žemas
Bendravimas tarp komandos narių	Žemas	Aukštas	Aukštas	Aukštas	Aukštas	Žemas	Aukštas	Žemas
Atnaujinimų integravimo laipsnis	Aukštas	Žemas	Žemas	Aukštas	Aukštas	Žemas	Žemas	Žemas
Komandos organizavimas	Hierarchine	Asmeninė	Asmeninė	Hierarchija	Asmeninė	Hierarchija	Hierarchija	Asmeninė

Metodai charakterizuojami pagal jo vykdymą bei rezultatus (*8 lentelė.*). Metodo procesas gali būti charakterizuojamos pagal Agile PĮ kūrimo veiklas (*6 lentelė.*) bei nurodymų ir taisyklių abstrakcijos lygį (*7 lentelė.*).

6 lentelė. *Agile metodų palyginimas pagal PĮ kūrimo veiklas*

Veiklos	ASD	AM	Crystal	DSDM	XP	FDD	PP	Scrum
Projekto paleidimas	-	-	-	+	-	-	-	-
Reikalavimų apibūdinimas	+	+	-	+	+	+	+	+
Modeliavimas	+	+	+	+	+	+	+	+
Kodavimas	+	+	+	+	+	+	+	+
Kodo modulių testas	+	-	+	+	+	+	+	+
Integravimo testas	+	-	+	+	+	+	+	+
Sistemos testas	+	-	+	+	+	+	+	+
Pripažinimo testas	+	-	-	+	-	-	-	-
Kokybės kontrolė	+	-	-	-	-	+	-	-
Sistemos naudojimas	-	-	-	+	-	-	-	-

7 lentelė. *Agile metodų palyginimas pagal nurodymų ir taisyklių abstrakcijos lygį*

Abstrakcijos lygis	ASD	AM	Crystal	DSDM	XP	FDD	PP	Scrum
Projekto valdymas	+	-	+	+	-	+	-	+
Proceso aprašymas	+	-	+	+	+	+	-	-
Veiklų ir rezultatų taisyklės ir nurodymai	-	+	-	-	+	+	+	-

8 lentelė. *Agile metodų palyginimas pagal PĮ kūrimo veiklų rezultatus*

Rezultatai	ASD	AM	Crystal	DSDM	XP	FDD	PP	Scrum
Projekto modeliai / eskizai	-	+	-	+	-	+	+	+
Programos kodas su komentarais	+	+	+	+	+	+	+	+
Vykdomieji failai	+	+	+	+	+	+	+	+
Kodo modulių testas	+	-	+	+	+	+	+	+
Integracijos testas	+	-	+	+	+	+	+	+
Sistemos testas	+	-	+	-	+	+	+	+
Pripažinimo testas	+	-	-	+	-	-	-	-
Kokybės ataskaitos	+	-	-	-	-	-	-	-
Vartotojo dokumentacija	-	-	-	+	-	-	-	-

2.4 Ekstremalaus programavimo metodas

Ekstremalus programavimas (XP) yra PĮ kūrimo stilius orientuotas į programavimo technikas, aiškią komunikaciją ir komandinį darbą, kuris leidžia atlikti tuos darbus ir išspręsti PĮ kūrimo problemas, kurių anksčiau nebuvo įmanoma išspręsti. Ekstremalus programavimas susideda iš:

- PĮ kūrimo filosofijos, kuri paremta pagrindinėmis vertybėmis.
- Viena kitą papildančių PĮ kūrimo proceso veiklų [6][9].
- Komandos, kuri vadovaujasi bendromis taisyklėmis, dirba laikydamasi esminių principų ir praktikų[6].

2.4.1 Problemos

Suaktyvėjus verslui kuriant programinę įranga (PĮ) pradėta susidurti su sunkiomis problemomis. Žemiau pateikiamos pagrindinės problemos, su kuriomis pakankamai dažnai susiduriama, bei kaip šios problemos spendžiamos XP metodo kūrimo metu.

Atsiliekama nuo tvarkaraščio. Iki numatytų terminų nepavyksta sukurti programinės įrangos ar bent jos dalies, nes nebuvo gerai įvertinta projekto apimtis ir reikalavimų sudėtingumas.

XP sprendimas: vykdomi trumpi versijos išleidimo ciklai, kurių kiekvienas trunka iki mėnesio, todėl sumažinama atsilikimo rizika. Kiekviena versija kuriama nuo vienos iki keturių savaičių trunkančiomis iteracijomis, kurių metu realizuojamos užsakovo nurodytos savybės bei iš užsakovo priimamos pastabos bei reikalavimai. Kiekviena iteracija yra suskaidyta į užduotis, trunkančias nuo vienos iki trijų dienų. Vykdydama užduotį, komanda sprendžia iškilusias problemas. Galiausiai, XP metode pirmiausia yra realizuojamos svarbiausios savybės, todėl jei nespėjama, tai lieka nerealizuotos nelabai svarbios savybės.

Projekto nutraukimas. Po keleto planavimo klaidų užsakovai nebepasitiki projektu ir jį nutraukia.

XP sprendimas: užsakovui yra pasiūloma realizuoti nedidelę PĮ versija, kurioje būtų galima pamatyti dalykinę sritį. Taip yra užtikrinama, kad projektas bus vykdomas teisinga linkme ir įgis žymiai didesnę vertę.

PĮ greitai sensta. Net jeigu ir PĮ sėkmingai buvo sukurta ir pradėta naudoti versle, tačiau po keleto metų klaidų kiekis ar reikalingų pakeitimų kaštai pradeda augti taip sparčiai, kad PĮ turi būti pakeista kita.

XP sprendimas: viso PĮ kūrimo metu yra sudarinėjamas ir nuolat papildomas visapusiškas testų rinkinys. Šie testai yra vykdomi po kiekvieno pakeitimo (kelis kartus per dieną). Tokiu būdu yra užtikrinama PĮ kokybė. Siekiama, kad sistema būtų palaikoma geriausioje būsenoje ir kad nebūtų atidėliojamas problemų sprendimas.

Dažnos klaidos. PĮ klaidų kiekis toks didelis, kad niekas iš tikrųjų ta programa nesinaudoja.

XP sprendimas: vykdomas testavimas tiek iš kūrėjų (modulių testavimas) pusės tiek iš kliento (savybių, reikalavimų testavimas)

Dalykinės srities nesupratimas. PĮ sėkmingai sukuriama ir atiduodama verslui, tačiau ji neišsprendžia nei vienos verslo problemos.

XP sprendimas: klientas kviečiamas į komandą. Projekto specifikacija kūrimo metu nuolat tobulinama.

Dalykinės srities kaita. PĮ sėkmingai sukuriama ir atiduodama verslui, tačiau vietoj problemų, kurių išsprendimui buvo kuriama PĮ, atsirado kitos, labiau svarbios problemos.

XP sprendimas: versijos išleidimo ciklas yra trumpinamas, kad būtų išvengta didelių pasikeitimų vienoje versijoje. Versijos kūrimo metu užsakovas gali dar nerealizuotą funkcionalumą pakeisti nauju. Taip kūrėjų komanda net nepajus ar jie dirba su nauju ar senu funkcionalumu.

Nereikalingas funkcionalumas. Visuomet galima padaryti daug įdomių PĮ funkcionalumo patobulinimų, kuriems reikia skirti daug laiko, tačiau jie niekuomet neatneša verslui apčiuopiamos naudos.

XP sprendimas: griežtai nustatyta, kad realizuojamos yra tik aukščiausią prioritetą turinčios užduotys.

Specialistų kaita. Dažniausiai visi geri specialistai po keleto metų pradeda nekęsti kuriamos PĮ ir palieka komandą.

XP sprendimas: iš PĮ kūrimo komandos narių yra reikalaujama prisiimti atsakomybę įvertinant bei atliekant savo darbą. Taip pat yra teikiama informacija, apie tai, kiek laiko praėjo, kad jie galėtų, pagal nustatytas taisykles, iš naujo įvertinti savo darbus, kas yra skatinama. Taip yra sumažinama nepasitenkinimo tikimybė, jei paprašoma atlikti sunkiai įveikiamą užduotį. Kiekvienas asmuo yra skatinamas bendrauti su likusia komanda, kas neleidžia jaustis vienišam, ir taip sumažinamas nepasitenkinimas darbu. Taip pat yra sukurtas specialistų kaitos modelis, t.y. nauji komandos nariai palaipsniui skatinami priimti vis daugiau ir daugiau atsakomybės, bei jiems nuolat teikiama kitų komandos narių pagalba.

2.4.2 Ekstremalaus programavimo vertybės

Ekstremalaus programavimo metodas yra paremtas vertybėmis. Šios vertybės kiekvienos kūrimo fazės metu yra iškeliamos, ir jos nurodo kaip reikėtų derinti komandos ir asmenines vertybes.

Paprastumas. Yra daroma tik tai kas yra reikalinga ir kas yra prašoma ir nieko daugiau. Siekiant galutinio tikslo yra daromi maži paprasti žingsneliai kurie padeda išvengti galimų klaidų. Taip yra maksimizuojama investicijų iki šio momento vertė.

Bendravimas. Kiekvienas yra komandos narys ir su juo kasdien turi būti bendraujama akis į akį. Dirbama kartu nuo pat sistemos reikalavimų rinkimo iki galutinio kodo paleidimo. Taip galima priimti geriausius sprendimus ir tinkamai išspręsti iškilusias problemas sukuriant geriausią produktą.

Grįžtamasis ryšys. Kiekvienos iteracijos metu pateikiama veikianti PĮ versija. Šiai versijai yra teikiami pasiūlymai, pagal kuriuos daromi reikiami pakeitimai. PĮ kūrimo procesas yra pritaikytas nuolatiniams siūlymams bei jų realizavimui.

Pagarba: Kiekvienas komandos narys yra vertinamas taip, kaip jis yra to nusipelnęs. Kiekvienas prisideda tuo kuo gali. PĮ kūrėjai ir užsakovai vertina vieni kitus. Kiekvienas turi prisiimti atsakomybę, bei imti iniciatyvos vykdant savo darbus.

Drąsa: Visuomet yra pateikiama reali situacija. Nėra dokumentuojami pasiteisinimai kai kas nors nepavyksta, kadangi planuojamas tikslas turi būti pasiektas. Nė vienas komandos narys niekada nedirba vienas. Vyksta nuolatinis prisitaikymas prie pokyčių.

2.4.3 XP metodo veiklos

Planavimas (angl. *The Planning Game*) – skubiai nusprendžiama kas bus padaryta iki nustatyto termino pabaigos ir ką daryti toliau atsižvelgiant į prioritetus ir technines galimybes. Jei plano neišeina įgyvendinti, planas keičiamas.

Nedidelės versijos (angl. *Small releases*) – užsakovui dažnai pateikiamos per trumpą laiką sukurtos naujos sistemos versijos. Tokiu būdu laiku gaunamas grįžtamasis ryšys, kuris įtakoja sistemos tolimesnį vystymą.

Metafora (angl. *Metaphor*) – būdas priminti sistemos vystymo komandai kaip veikia sistema

Paprasta konstrukcija (angl. *Simple design*) – sistema bet kurioje vietoje turi būti kiek galima paprastesnė ir tik tokia koka buvo planuota.

Pertvarkymas (angl. *Refactoring*) – Vykdant iteracinį procesą sistema yra pertvarkoma nekeičiant funkcionalumo: pašalinami pasikartojantys kodo fragmentai, pagerinami informacijos srautai, supaprastinama arba suteikiama lankstumo. Kodo modulių testai palengvina pertvarkymą, kadangi jie patvirtina, kad padaryti kodo pakeitimai nepakeitė funkcionalumo

Testavimas (angl. *Testing*) – kad sistemos kūrimas tęstųsi, pastoviai yra rašomi kodo modulių testai, kurie turi veikti nenutrūkstamai. Testuotojai padeda užsakovams rašyti testus, kurie parodo, kad sistemos funkcionalumas yra pabaigtas. XP metodo testavimas vykdomas visiškai kitaip negu kituose metoduose, kuriuose testavimas vykdomas kūrimo pabaigoj arba programuotojams nėra leidžiama testuoti savo parašyto kodo. XP metode yra pabrėžiama, kad testavimas pradedamas labai anksti ir skatinama, kad programuotojai patys testuotų savo kodą.

Programavimas poromis (angl. *Pair programming*) – kuriamos sistemos kodas yra rašomas poromis, t.y. du programuotojai sėdi šalia vienas kito prie to paties kompiuterio (kol vienas programuoja, kitas tuo metu duoda pastabas, peržiūri rezultatus ir ieško klaidų). Darbas poromis užtikrina palankią aplinką bendraujant tarp komandos narių, kadangi dirbant poroje

mokomasi iš vienas kito, o apsikeitimas poromis įtakoja tolimesnį žinių sklidimą. Poros taip pat gali būti sudaromos analizuojant bei projektuojant.

Bendras kodas (angl. *Collective code ownership*) – bet kuris komandos narys gali keisti bet kurią sistemos kodo vietą. Bendras sistemos kodas yra saugomas atskiroje saugykloje, į kurią kiekviena programuotojų pora perkelia savo kodo dalį. Bendrą kodą vienu metu integruoti, testuoti ir paleisti pakeitimus gali tik viena programuotojų pora, todėl yra naudojama tam tikra užrakinimo sistema.

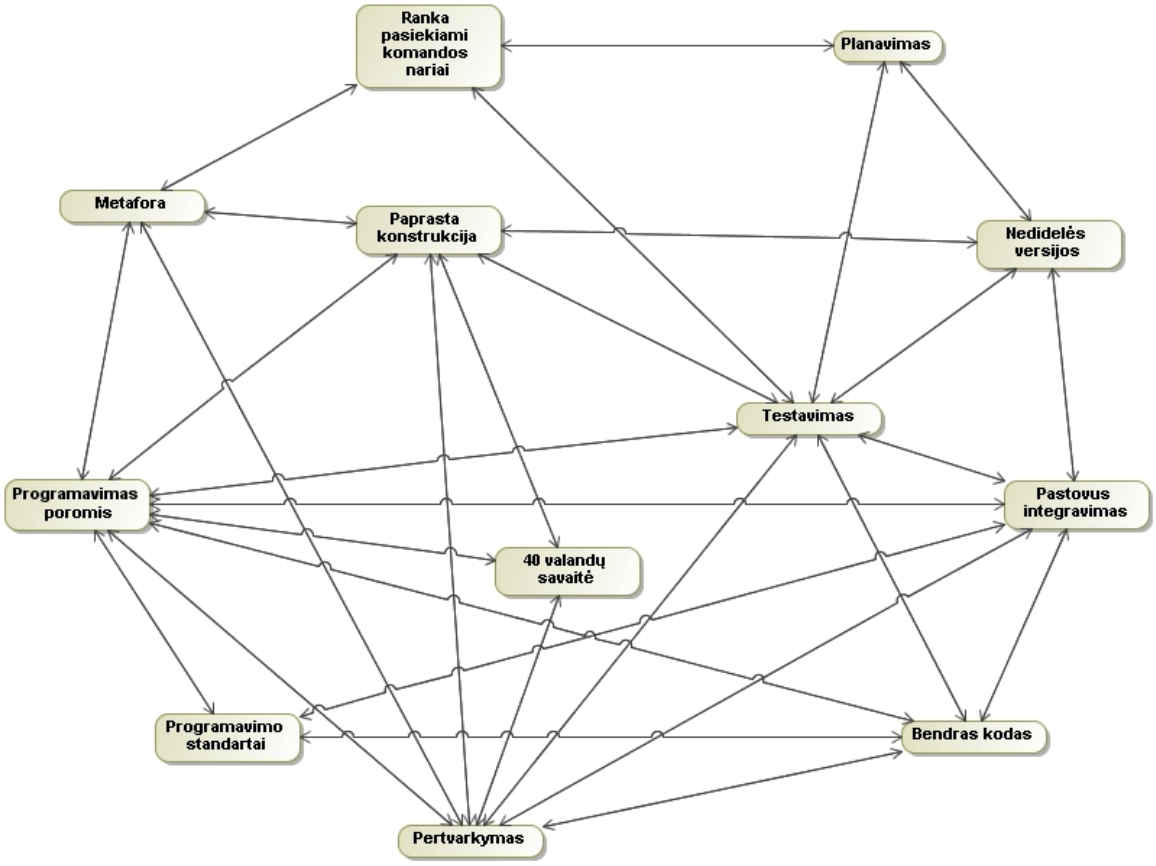
Pastovus integravimas (angl. *Continuous integration*) – kiekvieną kartą kai yra užbaigiama užduotis yra suintegruojama sistema t.y kelis kartus per dieną. Pastovus integravimas sumažina vienas kitam prieštaraujančių pakeitimų tikimybę.

40 valandų savaitė (angl. *40 hour week*) – svarbu dirbti ne daugiau kaip 40 valandų per savaitę (optimalus darbo tempas). Niekada negalima dirbti viršvalandžių antrą savaitę iš eilės, kadangi jie siejami su „projekto mirties maršu“, kai darbuotojai patiria spaudimą ir daug streso.

Ranka pasiekiami komandos nariai (angl. *On-site customer*) – komandos narys turi būti pasiekiamas bet kuriuo realiu laiku, kadangi XP metode labai svarbus komandinis darbas ir greitas grįžtamasis ryšys. Rekomenduojamas komandos dydis – tarp 2 ir 12 žmonių į kurią įeitų užsakovas, kuris teiktų reikalavimus, nustatytų prioritetus ir prižiūrėtų projektą, programuotojai, testuotojai, kurie užsakovui padėtų nustatyti pripažinimo testus, analitikai, kurie užsakovui padėtų nusakyti reikalavimus, projekto vadovas, kuris padėtų komandai nenukrypti nuo grafiko ir palengvintų darbo procesą bei vadybininkas, kuris teiktų komandai darbinę medžiagą, palaikytų komunikavimą ir derintų darbus. Visos šios rolės nebūtinai turi būti atskiri individai. Kiekvienas asmuo turi turėti atvirą darbo vietą šalia vienas kito.

Programavimo standartai (angl. *Coding Standards*) – programuotojai rašo programinį kodą naudodamiesi taisyklėmis, kurios padeda sumažinti besikartojantį darbą bei suvienodinti sistemos kodą taip, kad atrodytų, jog jis visas yra parašytas vieno kompetentingo asmens.

Kaip matome kiekviena veikla yra pakankamai paprasta, tačiau nė viena iš aukščiau paminėtų XP metodo veiklų (galbūt išskyrus testavimą) negali veikti atskirai. Jos viena kitai padeda išlaikyti pusiausvyrą. Sąveikaudamos tarpusavyje jos tampa sudėtingesnės. *4 pav.* parodytos visos veiklos, o tarp jų nurodytos rodyklės parodo, kad dvi veiklos sustiprina viena kitą. [4][6]



4 pav. *XP metodo veiklos papildo viena kitą*

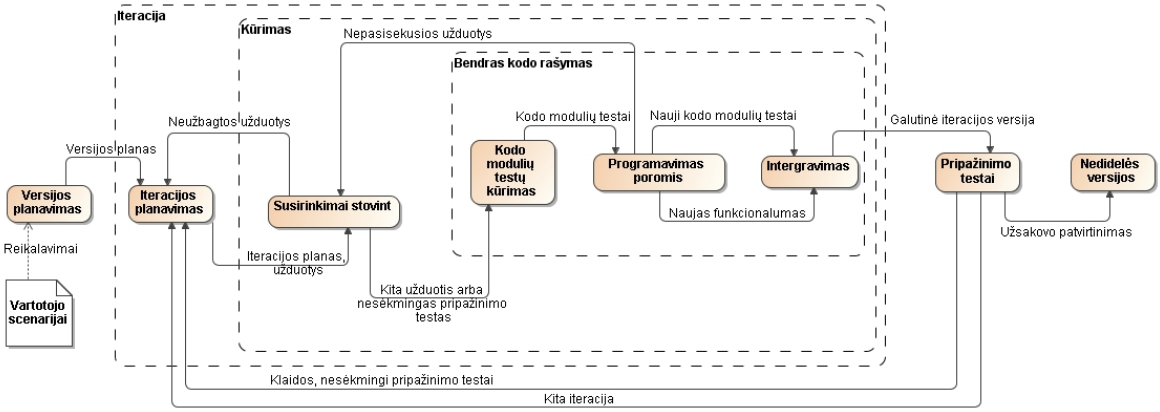
9 lentelė. *XP metodo veiklų sąsaja su Aglie manifesto principais [4]*

XP metodo veiklos	Agile manifesto principai											
	1	2	3	4	5	6	7	8	9	10	11	12
Planavimas	+	+	-	+	-	-	-	-	-	-	-	-
Nedidelės versijos	+	-	+	-	-	-	-	-	-	-	-	-
Simbolika	+	-	-	-	-	-	-	-	-	-	-	-
Paprasta konstrukcija	+	-	-	-	-	-	-	-	+	+	-	-
Pertvarkymas	+	-	-	-	-	-	-	-	+	+	-	-
Testavimas	+	+	-	-	-	-	+	-	+	-	-	-
Programavimas poromis	+	-	-	-	+	+	-	-	+	-	-	-
Bendras kodas	+	-	-	-	-	-	-	-	+	-	-	-
Pastovus integravimas	+	+	+	-	-	-	-	-	-	-	-	-
40 valandų savaitė	+	-	-	-	-	-	-	+	-	-	-	+
Ranka pasiekiami komandos nariai	+	-	-	+	-	+	-	-	-	-	+	+
Programavimo susitarimas	+	-	-	-	-	-	-	-	+	-	-	-

2.4.4 XP metodo gyvavimo ciklas

Ekstremalaus programavimo procesą [6] [8] galima suskaidyti į planavimo, projektavimo, kodavimo ir testavimo lygiagrečius procesus, kurie susideda iš tam tikrų taisyklių ir praktikų. Gali pasirodyti, kad tam tikros taisyklės ar praktikos nėra reikšmingos, tačiau jų visų visuma ir tarpusavio sąryšiai formuoja stiprią PĮ kūrimo proceso metodologiją, kuri puikiai susidoroja su kintančiais reikalavimais. XP procesą [12] atvaizduojanti schema pateikta *5 pav.* Naudojantis XP metodologiją pirmiausia pradedama nuo planavimo proceso renkant vartotojo siužetus,

analizuojant architektūrinius aspektus, vėliau užsakovai, vadybininkai ir programuotojai suderina PĮ leidimo planą, kurį turi patvirtinti ir su juo sutikti visi planavime dalyvaujantys asmenys ir vėliau pradedamas PĮ kūrimo procesas nuo iteracijos planavimo.



5 pav. *Ekstremalaus programavimo (XP) gyvavimo ciklo schema*

Vartotojo scenarijai (angl. *User Stories*). Tai užsakovo paruošti dokumentai, kur kiekviename jų naudojant jo terminologiją maždaug trimis sakiniais yra nurodyta ką sistema turėtų daryti. Jie atlieka panašias funkcijas kaip reikalavimai, panaudojimo atvejai ir scenarijai. Juose taip pat galima apibūdinti vartotojo sąsają. Jie naudojami versijos planavime laiko įvertinimui, pripažinimo testams.

Vartotojo scenarijuose informacija turi būti pateikta tokiu detalumo lygiu, kad jos pakaktų teisingai įvertinti scenarijaus realizavimo laiką. Taip pat juose turi būti vengiama konkrečių technologijų, duomenų bazės ar algoritmų aprašymų. Realizavimo etape, kūrimo komanda akis į akį susitinka su užsakovu ir detaliau aptaria sistemai keliamus reikalavimus.

„Idealus“ scenarijaus kūrimo laikas yra 1, 2 arba 3 savaitės. Idealus laikas tai laikas per kurį yra realizuojamas scenarijuje aprašytas funkcionalumas, jeigu niekas neblaško, neskiria kitų užduočių ir jeigu tiksliai yra žinoma, ką reikia daryti. Ilgiau nei trys savaitės reiškia, kad scenarijų reikia skaidyti, trumpiau nei viena savaitė reiškia, kad reikia apjungti kelis panašius scenarijus. Versijos planavime geriausia kai yra naudojama maždaug 80 (±20) vartotojo scenarijų.

Tam, kad būtų patvirtinta, kad vartotojo scenarijus buvo teisingai realizuotas sukuriamas vienas ar keli automatizuoti pripažinimo testai.

Versijos planavimas (angl. *Release Planning*). Versijos planavimo metu yra kuriamas versijos planas. PĮ kūrimo komanda vartotojo scenarijus įvertina diegimo laiko atžvilgiu. Šiame žingsnyje svarbiausia kuo geriau įvertinti scenarijus, kadangi vėliau dėl to gali atsirasti problemų. PĮ kūrimo komandai atlikus vertinimus, užsakovai suskirsto scenarijus pagal prioritetus ir nusprendžia, kuriems reikia skirti didžiausią prioritetą, kad jie būtų įdiegti pirmiausia. Po to vartotojo scenarijai dažniausiai išdėliojami ant didelio stalo ir PĮ kūrimo

komanda ir užsakovai grupuoja scenarijus, kurie turėtų būti įdiegti pirmojoje ar sekančiose versijose. Galima versijos planą sudarinėti atsižvelgiant į laiką arba į scenarijų grupes. Pirmu atveju žiūrima kiek vartotojo scenarijų gali būti įdiegta iki numatytos datos, o antruoju kiek prireiks laiko įdiegti visus pasirinktos grupės scenarijus.

Tačiau pagrindinė versijos planavimo idėja yra projektą įvertinti keturiais kintamaisiais:

- apimtimi – kiek darbo turi būti padaryta;
- ištekliais – kiek yra žmonių;
- laiku - kada projektas ar versija bus baigta
- kokybe – kokiu kokybės lygiu sistemas bus ištestuota

Visų šių keturių kintamųjų neįmanoma suvaldyti. Todėl, kad vieno kintamojo pasikeitimas įtakoja kito pasikeitimus.

Versijos planas naudojamas viso projekto metu kuriant iteracijų planus, kiekvienai iteracijai, taip pat priimant techninius bei verslo sprendimus.

Iteracija (angl. *Iteration*). Naudojant iteracijas PĮ kūrimo procesas tampa lankstesnis. Visas kūrimo procesas suskaldomas į 1-3 savaičių trukmės iteracijas, kurių trukmė nekeičiama viso kūrimo proceso metu. Šis pastovumas planavimą XP metode padaro paprastesnį ir patikimesnį.

Negalima planuoti tolimesnių užduočių. Dėl šios priežasties, kiekvienos iteracijos pradžioje vykdomas iteracijos planavimas. Tokiu būdu realizuojami patys svarbiausi, aktualiausi ir nuolat besikeičiantys vartotojo reikalavimai.

Kiekvienos iteracijos pabaigos data turėtų būti vertinama griežtai – jei atrodo, kad iki iteracijos pabaigos bus nesuspėta atlikti visų darbų, tuomet sušaukiamas kitas iteracijos planavimo susitikimas, kurio metu atmetamos kai kurios užduotys.

Didžiausias dėmesys turėtų būti skiriamas pačioms svarbiausioms užduotims, o ne neužbaigtoms užduotims.

Iteracijos planavimas (angl. *Iteration Planning*). Iteracijos planavimas vykdomas kiekvienos iteracijos pradžioje, kurios metu sudaromas iteracijos planas. Kiekviena iteracija trunka nuo vienos iki trijų savaičių. Kiekvienai iteracijai užsakovas parenka jam pačius svarbiausius vartotojo scenarijus iš versijos plano. Atrenkami nesėkmingi pripažinimo testai, kurie taip pat bus taisomi. Užsakovo parinktų vartotojo scenarijų apskaičiuota trukmė pridedama prie bendro projekto atlikimo greičio.

Vartotojo scenarijai ir nesėkmingi pripažinimo testai išskaidomi į programavimo užduotis, kurios užrašomos ant kortelių panašiai kaip vartotojo scenarijai. Skirtumas tas, kad vartotojo scenarijai yra parašyti užsakovo kalba, o užduotys PĮ kūrėjų kalba. Šios užduočių kortelės yra detalizuotas iteracijos planas.

Kiekvienas programuotojas pasirenka užduotį ir įsivertina jos atlikimo terminą. Tai svarbu padaryti, kadangi iteracijos metu, programuotojai nėra keičiami.

Kiekviena užduotis tęsiasi maždaug vieną, dvi arba tris „idealaus“ programavimo dienas. Užduotys, kurios trunka trumpiau negu vieną dieną, gali būti sujungtos, o tos, kurios trunka ilgiau nei tris dienas – išskaidomos.

Bendras iteracijos idealaus programavimo greitis (sudėjus visų užduočių atlikimo greitį) neturi viršyti praėjusių iteracijų atlikimo greičio. Priešingu atveju užsakovas turėtų dalį vartotojo scenarijų atidėti vėlesniam laikui. Jei greitis per mažas, vartotojo scenarijai turi būti papildomai pridedami.

Dažnai yra baiminamasi dėl vartotojo scenarijų atidėjimo. Šioje vietoje reikėtų prisiminti kodo modulių testavimo ir pertvarkymo svarbą. Taip pat reikėtų vengti funkcionalumo, kuris nebuvo planuotas, kadangi tai gali sulėtinti darbą.

Susirinkimai stovint (angl. *Stand Up Meeting*). Pagrindinis tokio susirinkimo tikslas yra bendravimas tarp komandos narių. Jis rengiamas kiekvieną rytą. Jo metu išsakomos problemos bei sprendimai. Šiame susirinkime dalyvauja visi komandos nariai. Jie sustoja ratu, tokiu būdu išvengdami ilgų diskusijų. Šie susirinkimai gali būti rengiami bet kur, pavyzdžiui, prie kompiuterio kur galima peržiūrėti programos kodą arba išbandyti pasiūlytus sprendimus.

Susirinkimo metu kūrėjai pateikia bent po tris dalykus: kas buvo atlikta vakar, kas ruošiamasi atlikti šiandien bei kokios problemos iškilo.

Bendras kodo rašymas (angl. *Collective Code Ownership*). Bendras kodo rašymas leidžia visiems komandos nariams keisti bet kurią kodo vietą, taisyti klaidas ir pertvarkyti bet kurią kodo dalį, t.y. už visos sistemos kūrimą yra atsakinga visa komanda, o ne vienas pagrindinis asmuo kuris vystytų turimą viziją. Tačiau bet kokia sudėtinga sistema negali tilpti vienos asmens galvoje.

Kiekvienas programuotojas savo kodo daliai sukuria kodo modulių testus, kurie, kartu su kodu padedami į saugyklą, kurioje bet kas gali taisyti bet kurį kodo fragmentą. Prieš pateikiant programos versiją užsakovui, visi testai turi pereiti kodą. Taip yra užtikrinama, kad bus realizuotas visas reikiamas PĮ funkcionalumas, o kuriam nors komandos nariui pakeitus ar pertvarkius kodą, jis nebus pakeistas.

Bendras kodo rašymas yra patikimesnis kodo rašymo būdas negu vadovaujant vienam asmeniui. Ypač jei tas asmuo nusprendžia palikti projektą.

Kodo modulių testų kūrimas (angl. *Unit Tests*). Tai vienas iš esminių XP metodo aspektų. Tačiau XP metode šie testai yra šiek tiek kitokie. Kad galima būtų rašyti modulių testų rinkinius, pirmiausia reikia susikurti arba atsisiųsti modulių testų bazę (angl. *Framework*). Po to reikėtų ištestuoti visas sistemos klases. Šio testo metu dažniausiai yra praleidžiami paimantys iš

grąžinantys metodai (angl. *Getter and Setter Methods*). Ir galiausiai prieš programuojant, reikėtų sukurti testus.

Kai pirmiausia yra kuriami testai, vėliau yra lengviau rašyti patį programinį kodą. Laikas, kurį užima kodo modulių testų bei programinio kodo rašymas yra toks pat kaip kad užimtų vien tik rašant programinį kodą. Kodo modulių testų kūrimas padeda programuotojui suprasti, kas turėtų būti padaryta bei ar padarytas visas reikiamas funkcionalumas. Testai padeda griežtai laikytis reikalavimų. Darbo metu, šie testai teikia tiesioginį atsaką.

Dažniausia kai kurias sistemas yra sudėtinga ištestuoti. Tos sistemos dažniausia buvo kurtos pirmiausia rašant programinį kodą, o tik po to testuojant. Testavimą dažniausiai atlieka visiškai kita komanda. Tačiau pirmiausia kuriant testus, didelės įtakos turės noras ištestuoti viską, kas yra naudinga užsakovui, o sistema bus sukurta taip, kad ją bus lengviau testuoti.

Šie testai turėtų būti saugomi saugykloje kartu su testuojamu kodu, kuris negali būti pateikiamas jei jam nėra sukurtų testų. Todėl, su kiekvienu nauju funkcionalumu, tie testai turėtų būti atnaujinami.

Automatinai testai per projekto gyvavimo laiką gali sutaupyti nemažai laiko ir pinigų, todėl jie turėtų būti kuriami nors jie ir pareikalauja šiek tiek brangaus laiko. Kuo sunkiau sukurti testą tuo labiau jis yra reikalingas. Modulių testai taip pat neturėtų būti kuriami paskutinius tris projekto mėnesius, kadangi be testų, kūrimas užsitęsia ir išeikvoja tuos paskutinius tris mėnesius ir dar daugiau. Be to, geriems testų rinkiniams taip reikia laiko išsivystyti, kadangi surasti visas galimas spragas užima laiko. Todėl, kad laiku būtų išbaigtas modulių testų rinkinys, jį reikia kurti dabar.

Modulių testai saugo, programinį kodą nuo netyčinių pakeitimų, kurie yra galimi apjungiant atskirus kodo fragmentus į bendrą visumą arba vykdant kodo pertvarkymą. Taip yra užtikrinama, kad galutinėje versijoje veiktų visas realizuotas funkcionalumas.

Turint universalų kodo modulių testų rinkinį su grįžtamaisiais (angl. *Regression*) ir patikrinimo (angl. *Validation*) testais, galima dažnai atlikti kodo integravimą. Jei suintegravus atliktus pakeitimus ar naują funkcionalumą ir paleidus vėliausią testų rinkinio versija, testai yra neįvykdomi, reiškia vėliausiai suintegruoti pakeitimai yra nesuderinami su komandos pateikta sistemos versija. Tokiu atveju yra atliekami papildomi pataisymai, išsprendžiantys nesudėtingas nesuderinamumo problemas. Tokiu būdu nesudėtingų problemų sprendimas kas kelias valandas užima mažiau laiko negu sprendimas sudėtingų problemų artėjant nustatyto termino pabaigai.

Galimas atvejis, kad klaidos pasitaiko ne tik programiniame kode, bet ir testuose. Jos nustatomos sprendžiant ir taisant atsiradusias problemas. Klaidų tikimybę testuose galima sumažinti testus kuriant prieš programinio kodo sukūrimą.

Programavimas poromis (angl. *Pair Programming*). Visas PĮ kodas yra sukurtas programuotojų dirbančių poromis prie vieno kompiuterio. Taip yra pagerinama PĮ kokybė nesueikvojant daug laiko t.y. du žmonės dirbdami prie vieno kompiuterio realizuoja tiek pat funkcionalumo, kaip kad dirbtų kiekvienas atskirai, tačiau dirbant tokiu būdu kokybė yra geresnė. Pagerinus kokybę, yra sutaupoma nemažai projekto lėšų.

Programavimas poromis tai socialiniai įgūdžiai, kuriuos išmokti, užima šiek tiek laiko. Vardan bendro tikslo iš abiejų partnerių turi būti siekis dirbti bendradarbiaujant. Geriausia programuotojų pora yra ta, kuri žino kada pasakyti „pirmiausia išbandom tavo idėją“. Nereikia tikėtis, kad viskas pavyks iš pirmo karto. Geriausia, jei kas nors iš komandos parodytų kaip tai yra daroma.

Dažnas kodo integravimas (angl. *Integrate Often*). Jeigu įmanoma programuotojai savo sukurtą programinį kodą turėtų integruoti ir į kodo saugyklą pateikti kas kelias valandas. Ilgiausia kiek galima pakeisto kodo nepateikti į saugyklą, yra diena. Pastovus integravimas dažnai padeda išvengti nesuderinamumo bei kodo suskaidymų, kai programuotojai vienas su kitu nebendrauja apie tai, kas turėtų būti iš naujo panaudota. Kiekvienas turėtų naudoti paskutinę programos versiją. Pakeitimai neturi būti padaryti neatnaujintam kodui, kas priešingu atveju gali įtakoti integracijos problemų atsiradimą.

Kiekviena programuotojų pora yra atsakinga už savo kodo integravimą. Vienu metu savo kodą gali integruoti tik viena pora.

Integracija tai „mokėk dabar arba vėliau mokėsi daugiau“ veikla. T.y. jei programa integruojama nedidelėmis dalimis viso projekto metu, integravimas neužima savaičių darbo, kai artėja projekto pabaiga.

Pripažinimo testai (angl. *Acceptance tests*) yra kuriami iš vartotojo scenarijų. Iteracijos metu vartotojo scenarijai, atrinkti iteracijos planavimo metu, yra padengiame pripažinimo testais. Kai vartotojo scenarijus buvo sėkmingai realizuotas, užsakovas aprašo scenarijų testavimą. Vienas scenarijus turi turėti tiek pripažinimo testų, kad būtu užtikrintas funkcionalumo veikimas.

Pripažinimo testai yra juodos dėžės testai. Kiekvienas pripažinimo testas parodo lauktą sistemos rezultatą. Už pripažinimo testų korektiškumą yra atsakingi užsakovai. Taip pat jie peržiūri šių testų rezultatus ir nusprendžia, kurie neįvykę testai yra svarbesni. Pripažinimo testai prieš išleidžiant versiją taip pat naudojami kaip grįžtamieji testai. žinimo testai. Tai reiškia, kad kiekvienoje iteracijoje turi būti kuriami nauji pripažinimo testai.

Nedidelės versijos (angl. *Small Releases*). Kūrėjų komanda, sistemos iteracijos versijas turi dažnai pateikti užsakovui. Kai kurios komandos PĮ vartojimui pateikia kiekvieną dieną. Ilgiausia kiek galima nepateikti vartojimui naujos PĮ versijos yra savaitė ar dvi. Kiekvienos

iteracijos pabaigoj turi būti ištestuota, veikianti ir paruošta vartojimui PĮ versija, kuri parodoma užsakovams. Jie nusprendžia ar galima šią PĮ pateikti vartojimui.

Versijos planavimo susitikime yra planuojami nedideli dalykinės srities funkcionalumo moduliai, kurie gali būti anksti įdiegti užsakovo aplinkoje. Yra labai svarbu laiku gauti naudingas pastabas, kad sistemos kūrimo metu būtų galima į jas atsižvelgti. Kuo ilgiau delsiama pateikti svarbias sistemos savybes sistemos vartotojams, tuo mažiau laiko lieka jų taisymui.

2.5 Testavimo metodai

Agile metodologijų grupės, ekstremalaus programavimo metode, yra naudojamas testavimu paremto (angl. *Test Driven*) PĮ kūrimo modelis. Pirmiausia programuotojai sudaro kodų modulių testinius atvejus (angl. *Test Cases*). Šie testai ilgainiui tampa netinkamais naudoti, tačiau tai yra numatyta iš anksto. Kuriamos PĮ kodas toliau yra tikrinamas nuolatos papildomais testinių atvejų rinkiniais (angl. *Test Suites*). Testinių atvejų rinkiniai yra nuolatos atnaujinami naujai rastomis sutrikimo sąlygomis ir ribinėmis reikšmėmis, vėliau jie yra integruojami su visais naujai sukurtais grįžtamaisiais testais (angl. *Regression Tests*).

Testavimas gali būti vykdomas skirtinguose lygiuose:

Programos kodo modulių testavimas (angl. *Unit Testing*) [13]. Tikrinami smulkiausi PĮ komponentai – funkcijos ir procedūros. Kiekvienas kodo modulis yra tikrinamas ar gerai įgyvendinta tai, kas buvo numatyta projektavimo etape. Objektiškai orientuotoje aplinkoje programinių kodo modulių testavimas vykdomas klasių lygyje, o minimalus klasės ištestavimas apima konstruktorių ir destruktorių patikrinimą.

Integracijos testavimas (angl. *Integration Testing*). Atskleidžiami sąsajų (angl. *Interface*) ir tarpininkavimo tarp integruotų komponentų trūkumai. Vis didesnės PĮ komponentų grupės tol integruojamos ir testuojamos (kaip atitinka architektūrinio projektavimo reikalavimus), kol PĮ pradeda veikti kaip vieninga sistema.

Grįžtamasis testavimas (angl. *Regression Testing*). Naudojamas patikrinti, ar programinėje įrangoje atlikti pakeitimai nesąlygoja klaidingo sistemos veikimo ir pateikia laukiamus rezultatus. Po kiekvieno PĮ modifikavimo yra pakartotinai vykdomi seni testai, prie kurių modifikuotoms dalims pridedami naujai sukurti testiniai atvejai. Seni testai, neatitinkantys naujų programinės įrangos galimybių, nebenaudojami. Regresinio testavimo tikslas yra patikrinti naujų programinės įrangos galimybių funkcionavimo teisingumą ir įsitikinti, kad nauji PĮ pakeitimai nesukėlė jau ištaisytų klaidų.

Pripažinimo testavimas (angl. *Acceptance Testing*). Tikrinama, kiek keliamus reikalavimus atitinka pilnai integruota sistema.

Sistemos integravimo testavimas. Tikrinama, ar sistema yra integruota į kokias nors išorines ar trečiosios šalies (angl. *Third Party*) sistemas nusakytas sistemos reikalavimuose.

Vartotojo sąsajos testavimas (angl. *User Interface Testing*). Tikrinama ar sąsaja atitinka numatytus sąsajos reikalavimus, ar naudotojui yra patogu dirbti su programa. Šio testo tikslas įsitikinti, kad naudotojo sąsaja atitinka reikalavimus ir yra patogi naudojimui. Sąsaja gali būti testuojama tiek rankiniu, tiek automatiniu būdu, panaudojant specialius įrankius. Testavimo metu yra tikrinama navigacijos korektiškumas, duomenų įvedimas ir atliekami įvairių mygtukų paspaudimai.

Prieš išsiunčiant galutinę PĮ versiją, papildomai yra atliekami **alfa** arba **beta** testavimai:

Alfa testavimas gali būti simuliacinis ar realus veikimo testavimas atliekamas vartotojų, užsakovų arba nepriklausomos testuotojų grupės programos kūrimo vietoje. Alfa testavimas naudojamas standartinei (angl. *off-the-shelf*) PĮ, kaip pripažinimo testavimas (angl. *Acceptance Testing*).

Beta testavimas vykdomas pabaigus alfa testavimą. PĮ beta versijos pateikiamos platesniam vartotojų ratui, esant reikalui PĮ beta pateikiama viešai siekiant gauti išsamesnę vartotojų reakciją į gaminį (angl. *Feedback*).

Testuoti programinio kodo modulius galima rankiniu būdu arba tą gali automatiškai atlikti speciali programa.

Baltos dėžės testavimas – kada testuotojas turi priėjimą prie vidines duomenų struktūras ir algoritmus aprašančio programinio kodo.

Baltos dėžės testavimo tipai:

- Kodo padengimas – sugeneruojami testai patenkinti keliamus kodo padengimo lygio reikalavimus. Pavyzdžiui, testų projektuotojas gali sukurti testą įvykdyti visus teiginius programoje bent vieną kartą.
- Mutacijų (angl. *Mutation*) testavimo metodai
- Klaidų įvedimo (angl. *Fault Injection*) metodai
- Statinio testavimo metodai - baltos dėžės testavimas apima visą statinį testavimą

Programinio kodo pilnumo įvertinimas

Baltos dėžės testavimo metodai gali būti naudojami įvertinti testinio rinkinio (angl. *Test Suite*) pilnumą, kuris buvo sukurtas naudojant juodos dėžės testavimo metodus. Tai leidžia programuotojų komandai patikrinti įvairias retai tikrinamas sistemos dalis ir užtikrina, kad svarbiausi funkciniai taškai yra ištestuoti.

Dažniausiai naudojami programinio kodo padengimo būdai:

- Funkcinis padengimas – pranešantis apie įvykdytas funkcijas
- Formuluočių padengimas (angl. *Statement*) – pranešantis apie įvykdytų eilučių skaičių atlikus testą.

Grąžinamos padengimo metrikas pateiktos procentinėmis išraiškomis.

Testuojant programą baltos dėžės metodu yra sudaromi testiniai atvejai, o testuojamai programai keliami tokie tikslai:

- 100% kodo padengimas - kiekvienos aktyvios programos kodo eilutės (aktyvaus kodo eilutėse yra galimas įvykdyti programinis kodas, į jas neįeina programos komentarai) įvykdymas bent vieną kartą
- praėjimas visomis galimomis loginėmis programos šakomis bent vieną kartą

Programinio kodo moduliais (angl. *Code Unit*) vadinami smulkiausi vykdomi kodo vienetai – funkcijos ir procedūros. Kadangi naudojame juodos dėžės metodą, žinome paduodamus įėjimus, tikimės rezultatų, generuojami yra tokie testiniai atvejai, kad šie įvykdytų aukščiau paminėtas sąlygas.

Jei testuojami yra paprasto tipo duomenys, tai parenkamos atitinkamos ribinės reikšmės: -1,0,1, jeigu tikrinama trigonometrinės funkcija arcsin testiniam atvejui parenkama ribinė reikšmė: –1.

Jei užduotis yra pereiti visomis medžio šakomis yra sudaromas planas kokia eilės tvarka bus kviečiamos programos funkcijos, procedūros, klasės, pavyzdžiui:

1. execute class1.function1(-1)
2. execute class1.function2(1)
3. execute class2.function1(0)
4. execute class2.function2(arcsin(-1))

Sugeneruoti testiniai atvejai yra apjungiami į testų rinkinį. Pabaigtas testų rinkinys yra tikrinamas nuo klaidų, tam naudojamas klaidos įvedimo metodas – jis įvairiuose testuoti reikalingo programos kodo vietose atsitiktinai įveda klaidas ir tikrina ar naujai sukurtas testų rinkinys jas aptiks, ar pereis visomis loginėmis programos šakomis ir ar yra tinkamas naudojimui. Jei naudojimui tinkamas, juo galima testuoti programą, kuriai jis ir buvo parašytas.

Programos kodo modulių testavimo problema atsiranda testuojant sudėtingus duomenų tipus. Jei duomenų tipas ne sveikas skaičius, o klasė ar struktūra. Kaip galime elgtis, jeigu programos testavimui naudojamas automatinis testus generuojantis įrankis? Kaip galime padėti įrankiui išspręsti šią užduotį?

Orakulas. Orakulo problema. Orakulas, tai programų kūrimo specialistų naudojamas mechanizmas nustatantis ar testuojamas produktas patikrinimą išlaikė ar ne.

Orakulas turi tris savybes ir vykdo jas patikimai:

- Generator - generatorius, pateikia spėjamus arba laukiamus kiekvieno testo rezultatus
- Comparator – palyginimo mechanizmas, palygina rezultatus kurių tikimės su realiai gautais.
- Evaluator – įvertinimo mechanizmas, vertina palyginamų rezultatų panašumą, ar jis pakankamas, kad tenkintų keliamus reikalavimus.

Orakulu gali būti žmogus arba programa. Programinio kodo testavimo algoritmų tikslas yra korektiškai veikti tikrinant įvairias programas. Pageidaujama, kad testavimo algoritmai būtų universalūs, tačiau jokių išlygų netaikoma jų veikimo teisingumui. Testuojant programą ne visada yra žinomas jos vidinis veikimo algoritmas. Šis atvejis vadinamas juodos dėžės testavimu, kai žinomas į sistemą įeinantis duomenų srautas ir gaunami rezultatai, kuriuos prognozuoja orakulas.

Viena iš esminių orakulų problemų yra tai, kad jie gali aptarnauti tik mažą įėjimų ir išėjimų poaibį susijusį su bet kuriuo testu. Testuotojas gali tikslingai priskirti kintamiesiems reikšmes, tačiau visi programos kintamieji turi reikšmes pagal nutylėjimą. Neužimtos atminties kiekis, derinimo ir programos nustatymai gali taip pat paveikti testų rezultatus. Testų rezultatų vertinimas naudojančių testinius įėjimus, priklauso nuo nepilnų duomenų, kurie gali būti ir neteisingi.

Bet kuris orakulo funkcionalumas gali būti automatizuotas. Pavyzdžiui, galima prognozuoti programos testų rezultatus, pagal elgesį prieš tai išleistos programos versijos. Naudojant standartines funkcijas ir specialius modelius prognozes galime atlikti ir pagal konkurentų programą. Testavimo rezultatai gali būti gaunami rankiniu būdu arba naudojant įrankį, kuris paduoda įėjimo srautus į testuojamą programą ir pasiima išėjimo srautus, taip pat galima naudoti įrankį apjungiantį automatinį ir rankinį testavimą. Prognozuoti galime iš specifikacijų, gamybinių nurodymų ar kitų informacijos šaltinių, kurie reikalauja žmogų vertinti informaciją tam, kad galima būtų daryti prognozę.

Orakulo problema - orakulas tikisi atitinkamų testavimo rezultatų.

Juodos dėžės testavimas - kada nežinoma tikrinamos programos vidinė sandara: klasės, veikimo algoritmai ir funkcijos.

Jeigu testavimui naudojamas juodos dėžės metodas, tai žinomos testuojamą kodą aprašančios aukštesnio abstrakcijos lygio specifikacijos. Šios specifikacijos formalizuojamos pagalbinėm programom tiek, kad galima būtų joms pateikti įvesties srautą ir gauti rezultatus. Toks pat įvesties srautas paduodamas testuojamam programiniam kodui. Gautieji rezultatai palyginami tarpusavyje ir nustatomas programinio kodo teisingumo lygis.

2.6 Programinio kodo modulių testavimas

Kodo modulių testavimo procesą sudaro testų planavimo, testų rinkinio realizavimo bei kodo modulio patikrinimas naudojant sukurtus testus. Kodo modulio testavimo metu lyginamas esamas funkcionalumas su reikalaujamu t.y. aprašytu reikalavimų dokumentacijoje.

Žemiau aprašytas PĮ kodo modulių testavimo IEEE standartas [5]. Jis apibūdina sistemišką ir dokumentuotą kodo modulių testavimo procesą, kurį sudaro fazių, veiklų bei užduočių hierarchija.

2.6.1 Testų planavimo fazė

2.6.1.1 Bendrų aspektų, išteklių bei tvarkaraščio planavimas

Planavimo sąnaudos:

- Projekto planai;
- Programinės įrangos reikalavimų dokumentai.

Planavimo užduotys:

Bendrų kodo modulių testavimo aspektų nustatymas. Nustatomos probleminės vietos į kurias bus orientuotas testavimas. Apibūdinami charakteristikų atrinkimo (pvz., savybės, kurios turi būti ištestuotos), testo konstrukcijos ar testo realizavimo (pvz., testų rinkiniai, kuriuos privaloma naudoti) apribojimai. Nustatomi esantys įėjimo, išėjimo ir būsenų duomenų šaltiniai (pvz., testiniai arba rezultatų dokumentai, testinių duomenų generatoriai). Taip pat nustatomi bendri duomenų patvirtinimo būdai bei būdai, kurie bus naudojami rezultatų registravimui, rinkimui, atrinkimui, bei patvirtinimui. Apibrėžiami nustatymai taikomosios programos, kuri tiesiogiai siesis su testuojamais kodo moduliais.

Pilnumo reikalavimų nustatymas. Nustatomos sritys (pvz., savybės, procedūros, būsenos , funkcijos, duomenų charakteristikos, komandos), kurios bus padengtos kodo modulių testų rinkiniu, taip pat nustatomas kodo padengimo laipsnis, reikalaujamas kiekvienai sričiai. PĮ kūrimo metu testuojami kodo moduliai turi būti padengti testiniais atvejais.

Nutraukimo reikalavimų nustatymas. Nustatomi kodo modulių testavimo proceso įprastinio nutrakimo reikalavimai. Nutraukimo reikalavimai turi atitikti pilnumo reikalavimus. Nustatomos sąlygos, kurios gali įtakoti neįprastą kodo modulių testavimo proceso arba kokios nors vykdomos stebėjimo procedūros nutraukimą.

Išteklių reikalavimų nustatymas. Įvertinami ištekliai reikalingi testų rinkinio sudarymui, pirmajam paleidimui bei tolimesnių testavimo operacijų kartojimui. Atsižvelgiama į techninę įrangą, priėjimo laiką (pvz., paskirtas kompiuterio laikas), susisiekimą arba sisteminę PĮ, testavimo įrankius, testinius dokumentus, formas bei kitus ištekius. Taip pat įvertinamas resursų apimties poreikis. Nustatomi ištekliai reikalingi pasiruošimui bei atsakingos komandos. Taip pat

vykdomas šių išteklių parengimas, į kurį įeina, resursų, reikalaujančių nemažai laiko juos paruošti (pvz., pritaikyti testavimo įrankiai), parengimas. Nustatomos komandos už kodo modulių testavimą ir klaidų taisymą. Nustatomi darbuotojų reikalavimai, įskaitant jų kvalifikaciją, kiekį bei trukmę.

Bendro tvarkaraščio sudarymas. Pagal turimus išteklius bei testų galimybes visai kodo modulių testavimo operacijai yra sudaromas tvarkaraštis.

Planavimo rezultatai:

- Bendra kodo modulio testų planavimo informacija;
- Kodo modulių testo išteklių reikalavimai;
- Užbaigimo bei nutraukimo reikalavimai.

2.6.1.2 Testuojamų savybių apibrėžimas

Apibrėžimo sąnaudos:

- Kodo modulių reikalavimų dokumentacija;
- PĮ architektūros projektavimo dokumentacija – jeigu reikia.

Apibrėžimo užduotys:

Funkcinių reikalavimų analizė. Analizuojama kiekviena funkcija nurodyta kodo modulių reikalavimų dokumentacijoje. Užtikrinama, kad kiekviena funkcija turėtų unikalų identifikatorių. Jei reikia, pareikalaujama reikalavimų išaiškinimo.

Papildomų reikalavimų ir susijusių procedūrų nustatymas. Nustatomi kiti reikalavimai (pvz., atlikimas, bruožai ar konstrukcijos apribojimai) susiję su PĮ ypatybėmis, kurios būtų efektyviai ištestuotos modulių lygyje. Nustatomos visos procedūros (vykdomos arba vykdančios) susijusios tik su testuojamu moduliu. Užtikrinama, kad kiekvienas papildomas reikalavimas ar procedūra turėtų unikalų identifikatorių. Jei reikia, pareikalaujama reikalavimų išaiškinimo.

Modulių būsenų apibrėžimas. Jei modulių reikalavimų dokumentacija nusako PĮ būsenas (pvz., neaktyvus, pasiruošta priimti, vykdoma), yra apibrėžiama kiekviena būsena bei galimas būsenų pasikeitimas. Užtikrinama, kad kiekviena būsena bei galimas būsenų pasikeitimas turėtų unikalų identifikatorių. Jei reikia, pareikalaujama reikalavimų išaiškinimo.

Įėjimo ir išėjimo duomenų charakteristikų nustatymas. Nustatomos testuojamo modulio įėjimo ir išėjimo duomenų struktūros. Kiekvienai struktūrai nusakomos charakteristikos tokios kaip atėjimo greičiai, formatai, reikšmių sritys bei ryšiai tarp lauko reikšmių. Kiekvienai charakteristikai nusakomos galimos ribos. Užtikrinama, kad kiekviena charakteristika turėtų unikalų identifikatorių. Jei reikia, pareikalaujama reikalavimų išaiškinimo.

Testuojamų elementų atrinkimas. Atrenkamos savybės, kurias reikia testuoti. Taip pat testavimui atrenkamos susijusios procedūros, būsenos, būsenų pasikeitimai bei duomenų

charakteristikos. Parenkami klaidingi bei teisingi įėjimo duomenys. Jei neprasminga yra vykdyti pilną kodo testavimą, derėtų vadovautis kodo modulių testavimo aprašu pasirenkant reikalingus atlikti testus. Nustatoma nepasirinktų elementų rizika. Į kodo modulių testų sudarymo specifikacijos, testuojamų elementų skyrių, suvedamos pasirinktos savybes, procedūros, būsenos, būsenų pasikeitimai bei duomenų charakteristikos.

Apibrėžimo rezultatai:

- Testuojamų elementų sąrašas;
- Modulių reikalavimų išaiškinimo poreikiai.

2.6.1.3 Bendro plano tobulinimas

Tobulinimo sąnaudos:

- Testuojamų elementų sąrašas;
- Pagrindinė kodo modulių testų planavimo informacija.

Tobulinimo užduotys:

Metodikos tobulinimas. Nustatomi esami testiniai atvejai ir testinės procedūros, kurios turėtų būti naudojamos, bei specialūs metodai, naudojami duomenų tvirtinimui, rezultatų rinkimui, registravimui, atrinkimui bei tvirtinimui. Į kodo modulių testų sudarymo specifikacijos, metodo tobulinimo skyrių, suvedami tobulinimo principai.

Specialių resursų reikalavimų nustatymas. Nustatomi ir paruošiami specialūs ištekliai, reikalingi kodo modulio testavime (pvz., PĮ kuri tiesiogiai sąveikauja su moduliu). Į kodo modulių testų sudarymo specifikacijos, metodo tobulinimo skyrių, suvedami specialių išteklių reikalavimai.

Detalaus tvarkaraščio sudarymas. Sudaromas PĮ palaikymu, specialiaisiais ištekliais bei kodo modulių galimybių ir integracijos tvarkaraščiais paremtas kodo modulių testavimo tvarkaraštis. Į kodo modulių testų sudarymo specifikacijos, metodo tobulinimo skyrių, suvedamas tvarkaraštis.

Tobulinimo rezultatai:

- Patikslinta kodo modulių testų planavimo informacija;
- Kodo modulių testų specialių resursų reikalavimai.

2.6.2 Testų rinkinio realizavimo fazė

2.6.2.1 Testų rinkinio projektavimas

Projektavimo sąnaudos:

- Kodo modulių reikalavimų dokumentacija;

- Testuojamų elementų sąrašas;
- kodo modulio testų planavimo informacija;
- kodo modulių kūrimo dokumentacija;
- testų specifikacijos iš praeitų testavimų (jei tokie buvo).

Projektavimo užduotys:

Testų rinkinio architektūros sudarymas. Sukuriamas hierarchiškas testavimo objektų rinkinys, paremtas sąlygomis, kurias nurodo atrinkti susiję elementai (pvz., procedūros, būsenų pasikeitimai bei duomenų charakteristikos) bei testuojamomis savybėmis, taip, kad kiekvienas žemesnio lygio objektas galėtų būti tiesiogiai ištestuotas keleto testavimo atvejų. Parenkami tinkami testavimo atvejai. Testinių atvejų grupių identifikatoriai susiejami su žemiausio lygio objektais. Į kodo modulių testų sudarymo specifikacijos, testų identifikavimo skyrių, suvedama objektų hierarchija bei su ja susieti testavimo atvejų identifikatoriai tvarkaraštis.

Testų procedūrų detalizavimas (jei reikia). Kodo modulių reikalavimų dokumentacija, testo planavimo informacija bei testinių atvejų specifikacija nusako kodo modulių testų procedūras dėl to, sumažėja detalių specifikacijų poreikis. Atrenkamos jau sukurtos testų procedūros, kurios galėtų būti pakeistos, kad jas būtų galima panaudoti arba panaudoti jų nekeičiant. Papildomame kodo modulių testų sudarymo specifikacijos skyriuje arba atskirame procedūrų specifikavimo dokumente yra apibrėžiamos papildomos procedūros. Jei nėra akivaizdi sąsaja tarp testavimo atvejų ir procedūrų, yra sukuriama juos susiejanti lentelė, kuri įrašoma į kodo modulių testų sudarymo specifikaciją

Testinių atvejų specifikavimas. Specifikuojami nauji testavimo atvejai, kurie gali būti susiejami su jau sukurtomis specifikacijomis. Papildomame kodo modulių testų kūrimo specifikacijos skyriuje arba atskirame dokumente yra pateikiamos specifikacijos (tiesiogiai arba per nuorodas).

Testinių atvejų specifikacijų rinkinio pagal projektavimo informaciją atnaujinimas (jei reikia). Jei reikia atnaujinama testų rinkinio architektūra remiantis kodo modulių planu. Peržvelgiamos atrinktų algoritmų charakteristikos bei vidinės duomenų struktūros. Nustatomi vidinių duomenų kontroliniai srautai bei pakeitimai, kurie turėtų būti registruojami. Įvertinamos registravimo problemos, kurios gali iškilti (pvz., registruojant kontrolinį srautą sudėtinguose algoritmuose ar fiksuojant vidinių duomenų struktūrų (pvz., stekų bei medžių) pakitimus). Jei būtina, pareikalaujama pagerinti modulio struktūrą (pvz., suformuotos duomenų struktūros pašalinimo galimybės), kad būtų palengvintas kodo modulio testavimas. Papildoma testinių atvejų specifikacija naujas testiniais atvejais pagal atsiradusius kodo modulio pakeitimus.

Testų kūrimo specifikacijos užbaigimas. Užbaigiama kodo modulio testų kūrimo specifikacija.

Projektavimo rezultatai:

- Kodo modulio testų kūrimo specifikacija;
- Atskiros testų procedūrų specifikacijos;

- Atskiros testinių atvejų specifikacijos;
- Modulio struktūros patobulinimai.

2.6.2.2 Patobulinto plano ir projekto realizavimas

Realizacijos sąnaudos:

- Kodo modulio testų planavimo informacija;
- Testinių atvejų specifikacijos kodo modulių testų kūrimo specifikacijoje arba atskiruose dokumentuose;
- PĮ duomenų struktūros aprašymai;
- Testavimo resursai;
- Testavimo elementai;
- Galimi testiniai duomenys iš praeitų testavimų;
- Galimi testavimo įrankiai iš praeitų testavimų.

Realizavimo užduotys:

Testinių duomenų perėmimas ir patikrinimas. Perimami esami testavimo duomenys, kurie galėtų būti pakeisti, kad juos būtų galima panaudoti arba panaudoti jų nekeičiant. Generuojami bet kokie trūkstami nauji duomenys. Kad būtų užtikrintas duomenų nuoseklumas ir integralumas, įtraukiami papildomi duomenys. Naudojant PĮ duomenų struktūros specifikacijas, patikrinami visi duomenys (net ir tie, kurie nebuvo pakeisti). Jei nėra akivaizdi sąsaja tarp testavimo atvejų ir duomenų rinkinių, yra sukuriama juos susiejanti lentelė, kuri įrašoma į kodo modulių testų kūrimo specifikaciją.

Specialių resursų perėmimas. Perimami specialių išteklių reikalavimuose nurodyti testavimo resursai.

Testavimo elementų perėmimas. Surenkami testavimo elementai įskaitant galimus aprašus, operacinės sistemos procedūras, valdymo duomenis (pvz., lenteles) bei programas. Perimama PĮ numatyta testo planavimo metu kuri yra tiesiogiai susijusi su testiniu moduliu. Į kodo modulių testų bendros ataskaitos, santraukos skyrių, įrašomi kiekvieno elemento identifikatoriai.

Realizavimo rezultatai:

- Patvirtinti testiniai duomenys;
- Testavimo priemonės;
- Testinių elementų nustatymai;
- Trumpa pradinė informacija.

2.6.3 Kodo modulio patikrinimas

2.6.3.1 Testinių procedūrų vykdymas

Vykdymo sąnaudos:

- Patvirtinti testiniai duomenys;
- Testavimo priemonės;

- Testinių elementų nustatymai;
- Testinių atvejų specifikacijos;
- Testų procedūrų specifikacijos;
- Nesėkmingų testų analizės rezultatai (jei tokie buvo).

Vykdymo užduotys:

Testų paleidimas. Nustatoma testinė aplinka, paleidžiamas testų rinkinys. Į kodo modulių testų bendros ataskaitos, rezultatų santraukos skyrių, įrašomi visi testavimai.

Rezultatų nustatymas. Nustatoma ar kodo modulis perėjo ar neperėjo kiekvieną testavimo atvejį, t.y. ar buvo gautas reikalaujamas rezultatas, kuris buvo nustatytas kiekvieno testavimo atvejo aprašyme. Į kodo modulių testų bendros ataskaitos, rezultatų santraukos skyrių, įrašomi pavykę ir nepavykę rezultatai, o į operacijų santraukos skyrių įrašomi išteklių panaudojimai.

Kodo modulis, neperėjęs testavimo, yra analizuojamas, o klaidos informacija įrašoma į kodo modulių testų bendros ataskaitos, rezultatų santraukos skyrių. Po to parinkus tinkamą atvejį atliekami atitinkami veiksmai:

1 atvejis: *Klaidinga testavimo specifikacija arba testiniai duomenys.* Ištaisomos klaidos, o klaidos pataisymai įrašomi į kodo modulių testų bendros ataskaitos, operacijų santraukos skyrių. Pakartotinai paleidžiami nepavykę testai.

2 atvejis: *Klaida testo procedūros vykdyme.* Pakartotinai paleidžiami netinkamai įvykę testai.

3 atvejis: *Klaida testinėje aplinkoje (pvz., sistemos PĮ).* Atliekamas vienas iš dviejų galimų veiksmų:

a. pataisoma aplinka, į kodo modulių testų bendros ataskaitos, operacijų santraukos skyrių įrašomi klaidos pataisymai ir pakartotinai paleidžiami nepavykę testai;
b. kodo modulių testų bendros ataskaitos, operacijų santraukos skyriuje nurodomas neįprasto kodo modulių testavimo proceso nutraukimas bei priežastis, dėl ko testinė aplinka negali būti pataisyta. Toliau tęsiamas kodo modulio testavimo procesas vykdant patikrinimą nutraukimui.

4 atvejis: *Klaida kodo modulio realizavime.* Atliekamas vienas iš dviejų galimų veiksmų:

a. pataisomas kodo modulis, į kodo modulių testų bendros ataskaitos, operacijų santraukos skyrių įrašomi klaidos pataisymai ir pakartotinai paleidžiami visi testai;
b. kodo modulių testų bendros ataskaitos, operacijų santraukos skyriuje nurodomas neįprasto kodo modulių testavimo proceso nutraukimas bei priežastis, dėl ko kodo modulis negali būti pataisytas. Toliau tęsiamas kodo modulio testavimo procesas vykdant patikrinimą nutraukimui.

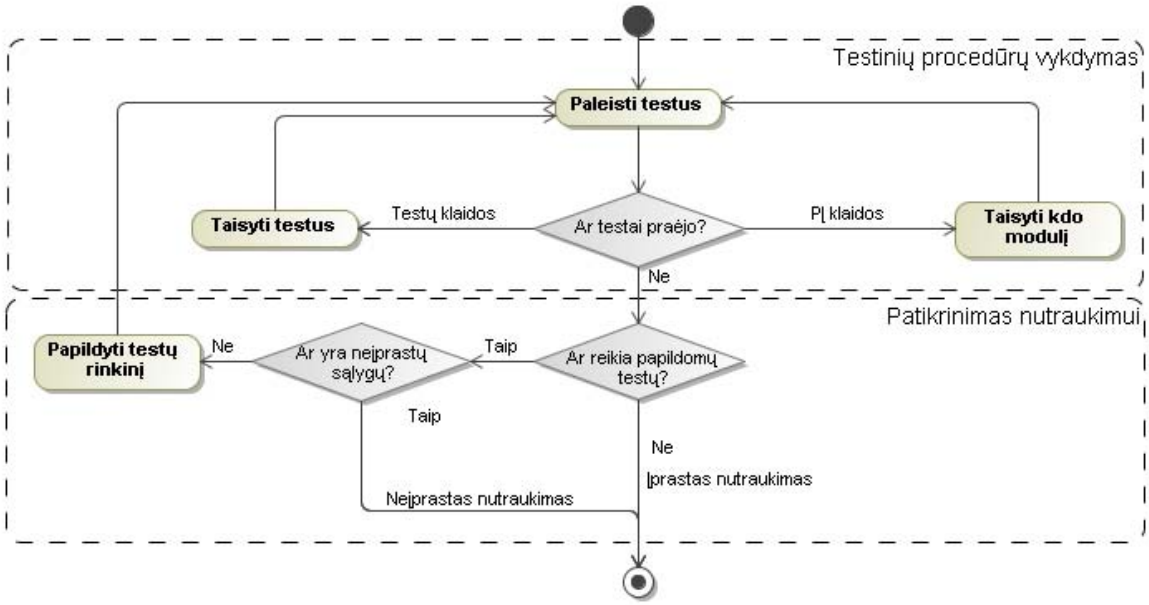
5 atvejis: *Klaida kodo modulio projektavime.* Atliekamas vienas iš dviejų galimų veiksmų:

a. Pataisomas kodo modulio projektas, kodo modulis bei atitinkamai testinių atvejų specifikacija ir duomenys, į kodo modulių testų bendros ataskaitos, operacijų

santraukos skyrių įrašomi klaidos pataisymai ir pakartotinai paleidžiami visi testai;

b. kodo modulių testų bendros ataskaitos, operacijų santraukos skyriuje nurodomas neįprasto kodo modulių testavimo proceso nutraukimas bei priežastis, dėl ko kodo modulio projektas negali būti pataisytas. Toliau tęsiamas kodo modulio testavimo procesas vykdant patikrinimą nutraukimui.

Pastaba: vykdymo ir rezultatų analizės ciklas turi būti kartojamas tol, kol yra patenkinti planavimo etape numatyti nutraukimo reikalavimai (*6 pav.*).



6 pav. *Testinių procedūrų vykdymo ir patikrinimo nutraukimui veiklos*

Vykdymo rezultatai:

- Kodo modulio testų vykdymo informacija (testų rezultatai, testavimų aprašymai, nesėkmingų testų analizės rezultatai, klaidų pataisymai, nepataisytų klaidų priežastys, naudoti resursai);
- Pataisytos testų specifikacijos;
- Pataisyti testiniai duomenys.

2.6.3.2 Patikrinimas nutraukimui

Patikrinimo sąnaudos:

- Užbaigimo bei nutraukimo reikalavimai;
- Kodo modulio testų vykdymo informacija;
- Atskiros testų procedūrų specifikacijos;
- Atskiros testinių atvejų specifikacijos;
- PĮ duomenų struktūros aprašymai.

Patikrinimo užduotys:

Testavimo proceso įprasto nutraukimo patikrinimas. Remiantis reikalavimų išbaigtumu ar nepraėjusių testų istorija, nustatomas papildomų testų poreikis. Jei papildomi testai nereikalingi, kodo modulių testų bendros ataskaitos, operacijų santraukos skyriuje fiksuojamas įprastas nutraukimas. Toliau tęsiamas kodo modulio testavimo procesas vykdant kodo modulio ir testo įvertinimą.

Testavimo proceso neįprasto nutraukimo patikrinimas. Jei yra neįprastų sąlygų (pvz., neištaisyta svarbi klaida, baigėsi laikas), tuomet užtikrinama, kad konkreti situacija įtakojanti nutraukimą, kartu su nepabaigtu testavimu bei kitoms neištaisytomis klaidomis, yra dokumentuota kodo modulių testų bendros ataskaitos, operacijų santraukos skyriuje. Toliau tęsiamas kodo modulio testavimo procesas vykdant kodo modulio ir testo įvertinimą.

Testų rinkinio papildymas. Jei reikalingi papildomi testai, o neįprasto nutraukimo sąlygos nėra patenkintos, yra papildomas testų rinkinys atliekant šiuos žingsnius:

1. Atnaujinama testų rinkinio architektūra (pagal testų rinkinio projektavimo veiklos testų rinkinio architektūros sudarymo užduotį.) bei papildomos testinių atvejų specifikacijos;
2. Pakeičiamos testų procedūrų specifikacijos;
3. Parenkami papildomi testiniai duomenys;
4. Kodo modulių testų bendros ataskaitos, operacijų santraukos skyriuje įrašomi pakeitimai;
5. paleidžiami papildomi testai, t.y. iš naujo vykdomos vykdymo veiklos užduotys (žr. skyrių *2.6.3.1 Testinių procedūrų vykdymas*).

Patikrinimo rezultatai:

- Kodo modulio testų patikrinimo nutraukimui informacija (nutraukimų patikrinimai bei testų rinkinio papildymai);
- Papildomos arba pataisytos testų specifikacijos;
- Papildomi testiniai duomenys.

2.6.3.3 Kodo modulio ir testo įvertinimas

Įvertinimo sąnaudos:

- Kodo modulio testų kūrimo specifikacija;
- Kodo modulio testų vykdymo informacija;
- Kodo modulio testų patikrinimo nutraukimui informacija;
- Atskiros testinių atvejų specifikacijos.

Įvertinimo užduotys:

Testavimo statuso apibūdinimas. Į kodo modulių testų bendros ataskaitos, pakitimų skyrių, įrašomi testų planų ir testų specifikacijų pakitimai bei nurodoma jų priežastis. Neįprasto nutraukimo atveju, nustatomos vietos, kurios liko nepakankamai ištestuotos, o kodo modulių testų bendros ataskaitos,

visapusiškumo įvertinimo skyriuje nurodomos priežastys. Kodo modulių testų bendros ataskaitos, rezultatų santraukos skyriuje nurodomi neįgyvendinti testai bei neįgyvendinimo priežastys

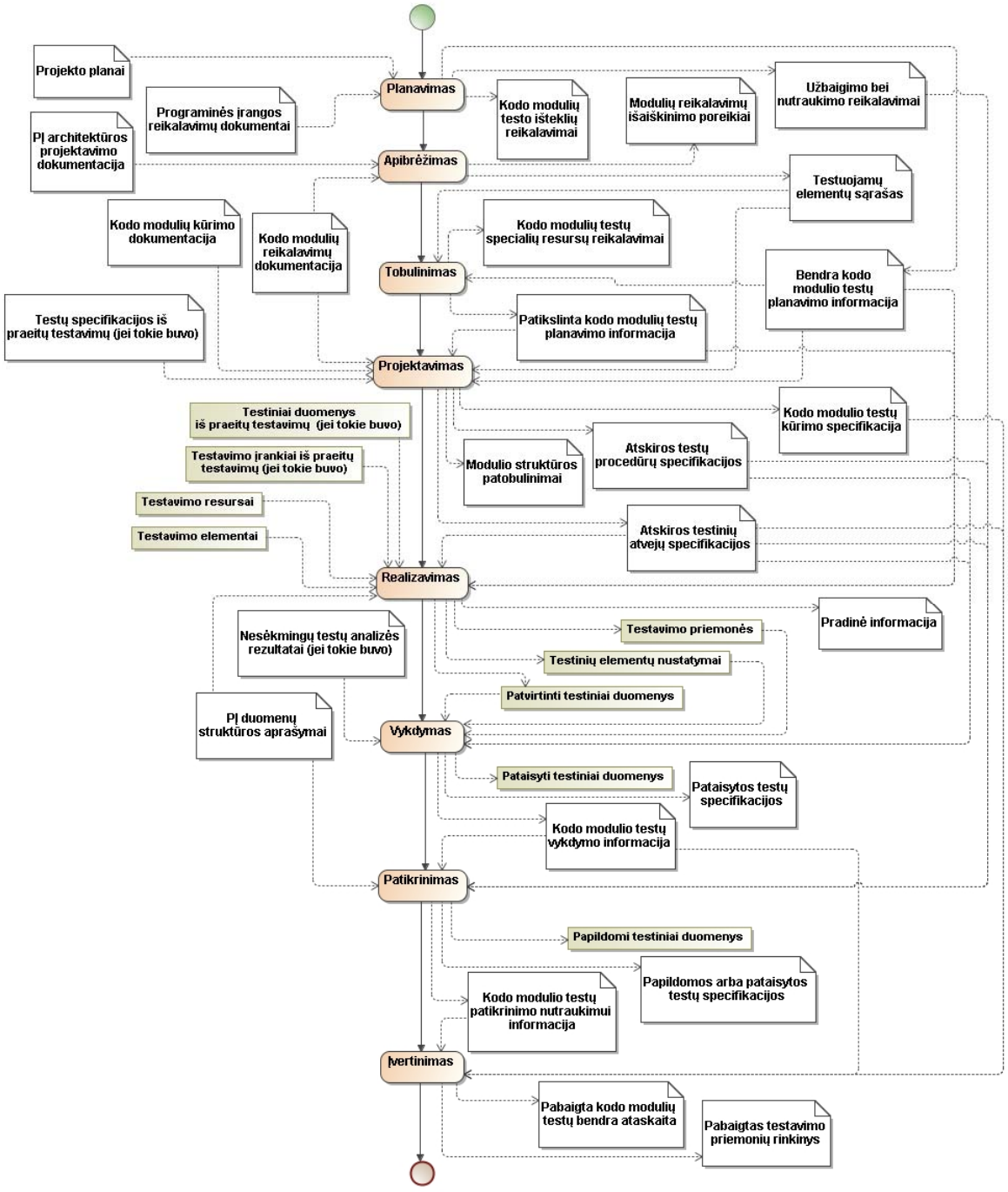
Kodo modulio statuso apibūdinimas. Į kodo modulių testų bendros ataskaitos, pakitimų skyrių, įrašomi testavimo metu nustatyti skirtumai tarp kodo modulio bei jo reikalavimų. Remiantis testavimo rezultatais ir aptiktų klaidų informacija, įvertinamas kodo modulio projektas bei realizacija. Į kodo modulių testų bendros ataskaitos, įvertinimų skyrių, įrašoma įvertinimo informacija.

Kodo modulių testų bendros ataskaitos užbaigimas. Kodo moduliui užbaigiama bendra testų ataskaita.

Testavimo produktų apsaugos užtikrinimas. Užtikrinama, kad testavimo produktai yra surinkti, susisteminti bei saugomi pakartotinam panaudojimui. Į šiuos produktus įeina, testo kūrimo specifikacija, atskiros testinių atvejų specifikacijos, atskiros testų procedūrų specifikacijos, testiniai duomenys, testinių duomenų generavimo procedūros, testavimo valdikliai (angl. *Driver*) ir kamščiai (angl. *Stub*) bei bendra testų ataskaita.

Įvertinimo rezultatai:

- Pabaigta kodo modulių testų bendra ataskaita;
- Pabaigtas testavimo priemonių rinkinys.



7 ***pav.*** *Kodo modulių testavimo veiklų diagrama*

2.7 Parasoft .Test testavimo įrankis

Ekstremaliame programavime nuolatinė kodo peržiūra ir testavimas vykdomi nuo pat projekto pradžios, todėl patogu naudoti automatinius testavimo įrankius tikrinti kuriamą PĮ.

Pasirinktas Parasoft .Test testavimo įrankis. Jis yra skirtas testuoti Microsoft Visual Studio .NET pagrindu parašytas programas. Šiuo įrankiu testavimai atlikti DiskDB programai.

Vartotojų tikslai ir problemos. Vartotojų testavimo programai keliami tikslai:

- Automatinis testinių atvejų generavimas;
- Galimybė nustatyti ir praplėsti įrankio naudojamus taisyklių rinkinius;
- Testavimo metodikų pasirinkimas;
- Automatinis ataskaitų generavimas ir išsamių testų rezultatų ataskaitų patogios peržiūros galimybė.

2.8 Analizės išvados

Įvertinus agile PĮ kūrimo metodikas pasirinktas ekstremalaus programavimo (XP) metodas. Programiniai kodo moduliai bus testuojami automatinėm priemonėm, tai vykdoma baltos dėžės principu.

DiskDB programa bus kuriama naudojant Microsoft Visual Studio .NET technologija, todėl testavimui pasirinktas Parasoft .TEST įrankis.

Ekstremalaus programavimo metodologijos privalumai:

- XP procesas gerai prisitaiko prie besikeičiančių reikalavimų, pateikia greitai nepilno funkcionalumo, tačiau veikiančias programas;
- XP procesas užtikrina aukštą PĮ kokybę nuolatinio PĮ logikos ir funkcionalumo testavimu;
- XP procesas pagerina PĮ kūrimo komandos darbo kokybę;

Trūkumai:

- Beveik nėra dokumentacijos;
- Kai kurių darbuotojų savybės gali tapti problema;
- Netinka kuomet komanda lokalizuota skirtinguose biuruose;
- XP procesas orientuotas daugiau į PĮ programavimą negu į projektavimą. Neįrodyta kaip XP procesas pasiteisintų kuriant PĮ, kurią reikia diegti į kitą, jau egzistuojančią sistemą.

3 DisksDB reikalavimai ir realizacija

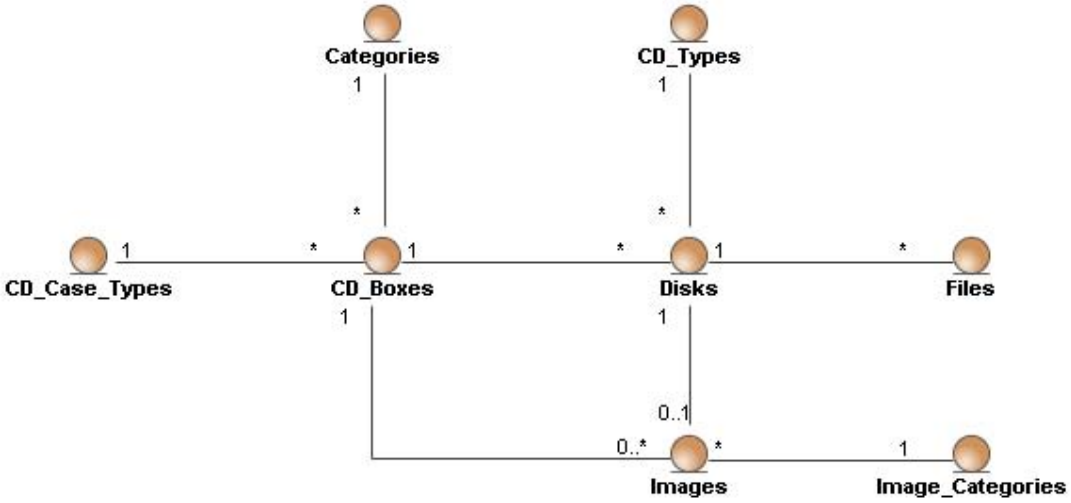
3.1 Nefunkciniai reikalavimai ir apribojimai

Pagrindiniai sistemos reikalavimai yra šie:

- Nepriklausomumas nuo platformos – kuriama sistema turi būti kuo mažiau priklausoma nuo konkrečios platformos. Kliento dalis turi būti nepriklausoma nuo operacinės sistemos: personalinio kompiuterio programa turi veikti Windows XP, 2003, 2000, Linux ir BSD aplinkose, o delninio kompiuterio programa – Windows CE 4.2 ir aukštesnėse aplinkose
- Išplečiamumas – pirminė projekto versija turi būti suprojektuota lanksčiai kadangi ateityje gali būti planuojama šį projektą plėsti.
- Patogus naudojimas – sistema turi būti draugiška vartotojui ir ją diegiančiam bei palaikančiam administratoriui.
- Lengvas integravimas – įrankis turi lengvai integruotis su organizacijoje naudojamomis ir naujai diegiamomis sistemomis.
- Sistema turi būti sukurta .NET palaikančia kalba.

3.2 Dalykinės srities modelis

Kompiuterizuojama dalykinė sritis suskirstoma į keletą objektų apjungtų ryšiais (žr. 8 pav.).



8 pav. *Diskų tvarkymų sistemos objektų diagrama*

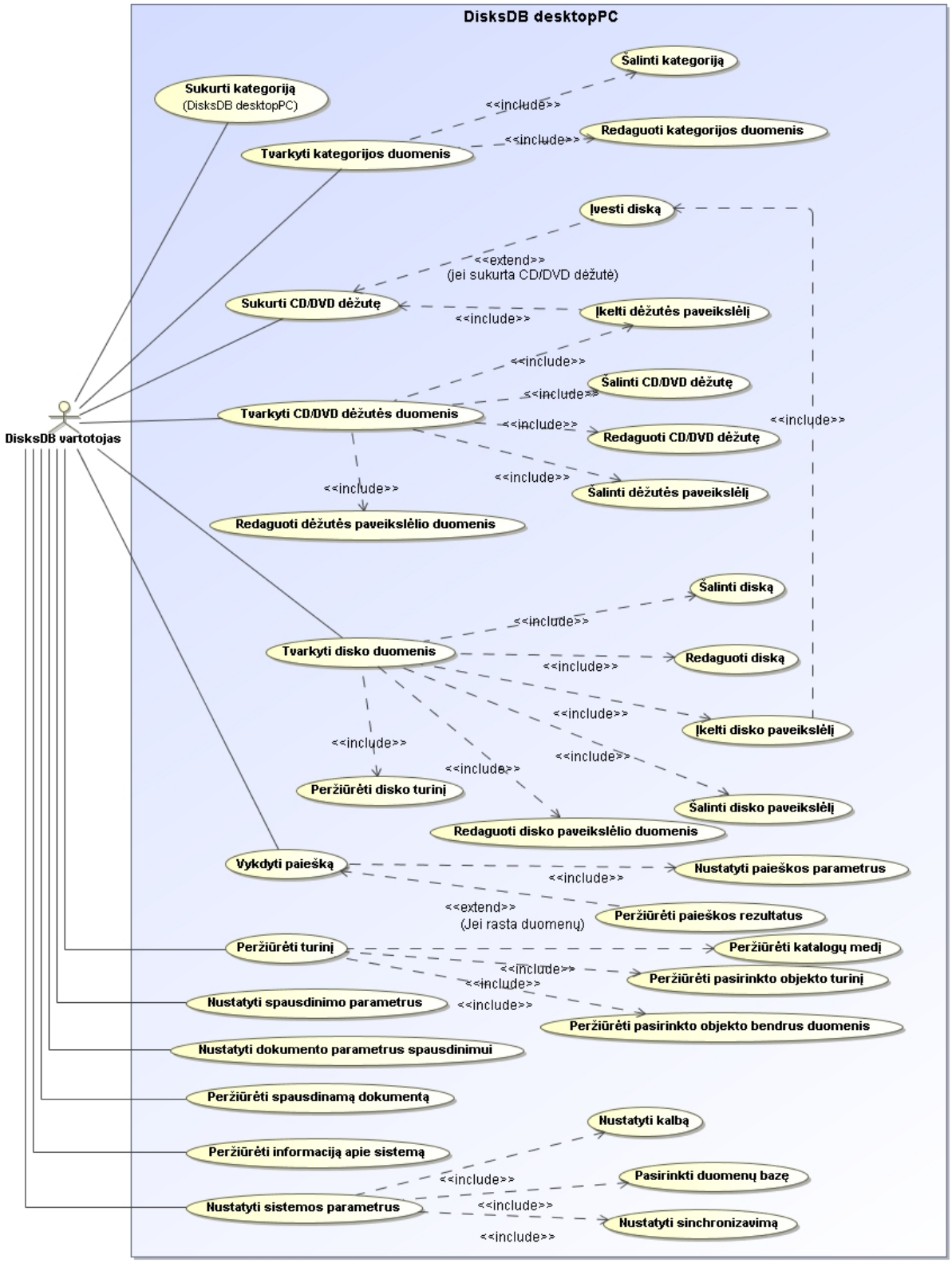
Šie objektai išreiškia pagrindinę kompiuterizuojamą struktūrą, kuri bus naudojama duomenims saugoti.

3.3 DisksDB sistemos funkcijos

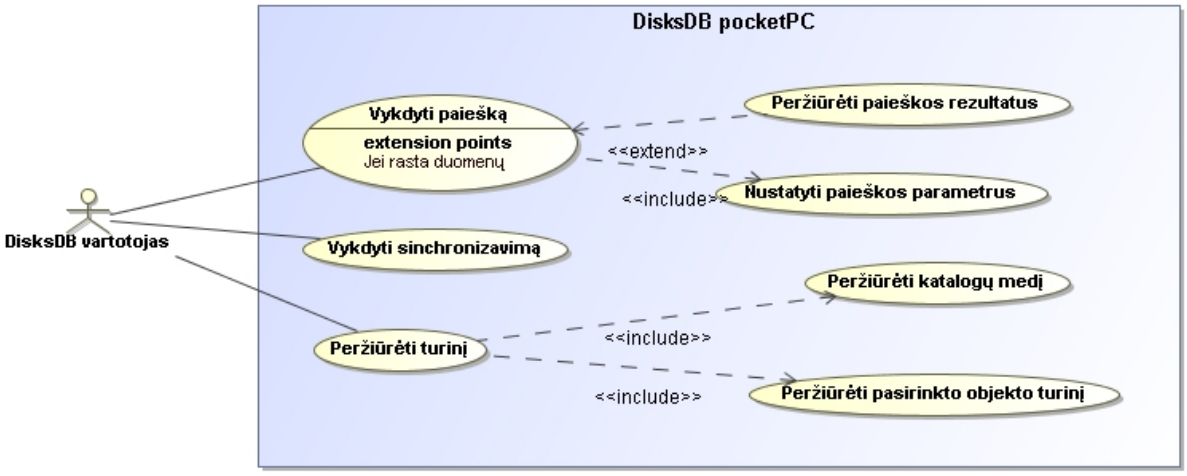
DisksDB sistema bus skirta dviem aplinkom: personalinio kompiuterio aplinkai ir delininio kompiuterio aplinkai. Jos bus atskirtos ir galės veikti nepriklausomai viena nuo kitos, tačiau

tarpusavyje jos galės bendrauti t.y. sinchronizuoti duomenis. Žemiau pateiktos šių aplinkų panaudojimo atvejų diagramos (*9 pav.* ir *10 pav.*).

Šiose schemose DisksDB vartotojas yra asmuo turintis kompaktinių ar DVD diskų organizavimo problemą. Sistema turi padėti ją išspręsti, realizuotomis funkcijomis.



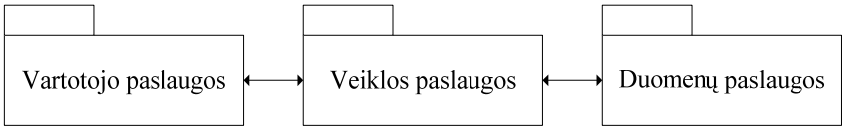
9 pav. *DisksDB desktopPC sistemos panaudojimų atvejų diagrama*



10 pav. *DisksDB pocketPC sistemos panaudojimų atvejų diagrama*

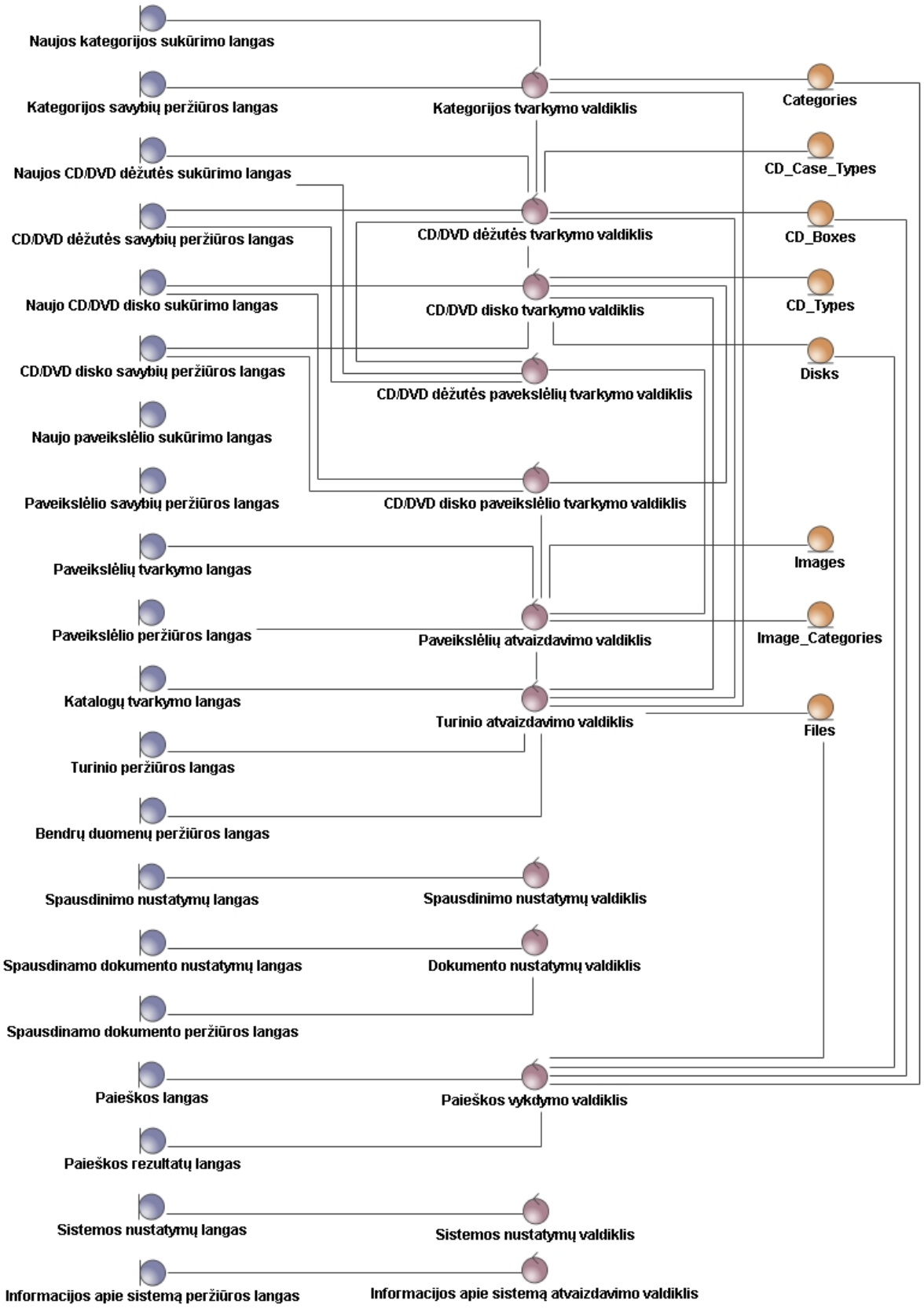
3.4 DisksDB sistemos loginė architektūra

Pagrindinės kuriamos sistemos loginės dalys: vartotojo paslaugos, veiklos paslaugos ir duomenų paslaugos (*11 pav.*). Vartotojo paslaugos apibūdina sistemos grafinę aplinką, kurioje vartotojas į sistemą įveda duomenis bei gauna rezultatus. Veiklos paslaugos atvaizduoja sistemos vidinę struktūrą, kuri parodo duomenų perėjimus tarp metodų ar klasių. Duomenų paslaugos parodo kaip duomenys yra saugomi, iš jos sistema paima duomenis kai vartotojas to pareikalauja.

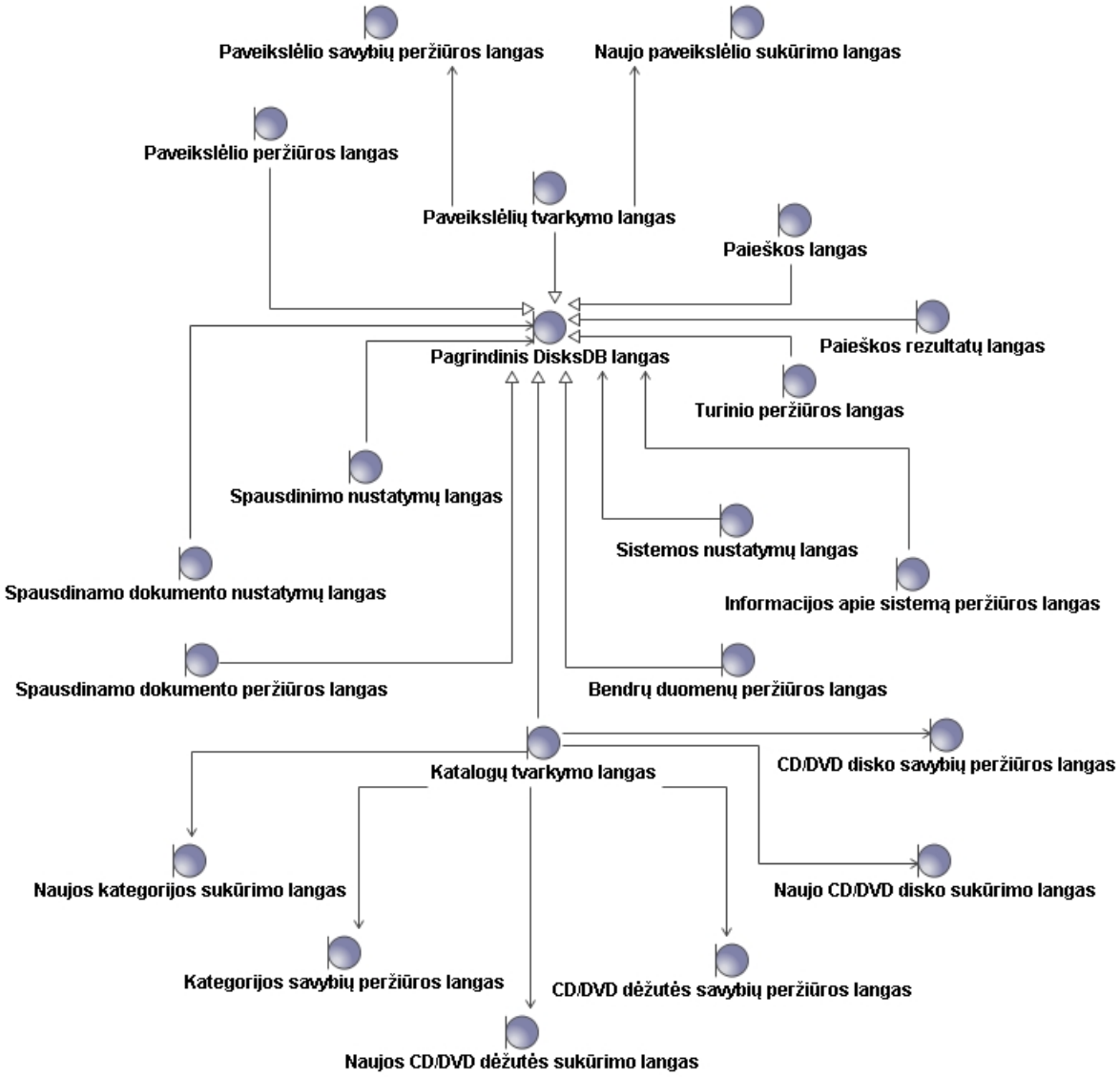


11 pav. *Trijų lygių loginės architektūros principinė schema*

Detalioj DisksDB (*12 pav.*) sistemos informacijoj pateikiami ryšiai tarp šių trijų loginės architektūros dalių. Jie ir parodo pagrindinius informacinius srautus.



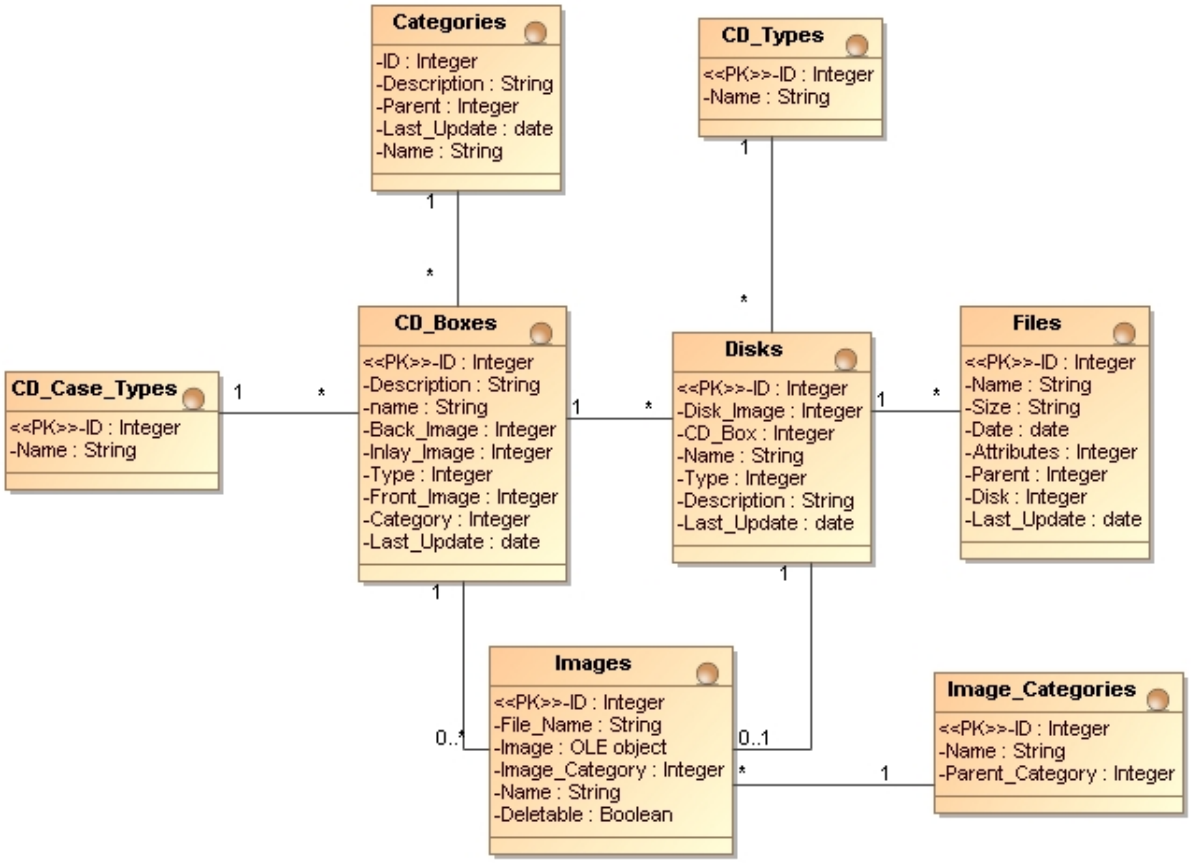
12 pav. *Detali DisksDB sistemos objektų diagrama*



13 pav. *Vartotojo paslaugų objektų tarpusavio ryšiai*

3.5 Duomenų bazės schema

Sistemos duomenų bazę sudaro dalykinės srities objektai sujungti tarpusavio ryšiais. Joje yra saugomi vartotojo įvesti duomenys, kurie vėliau gali būti panaudojami. Naudojant įvairias užklausas galima tuos duomenis

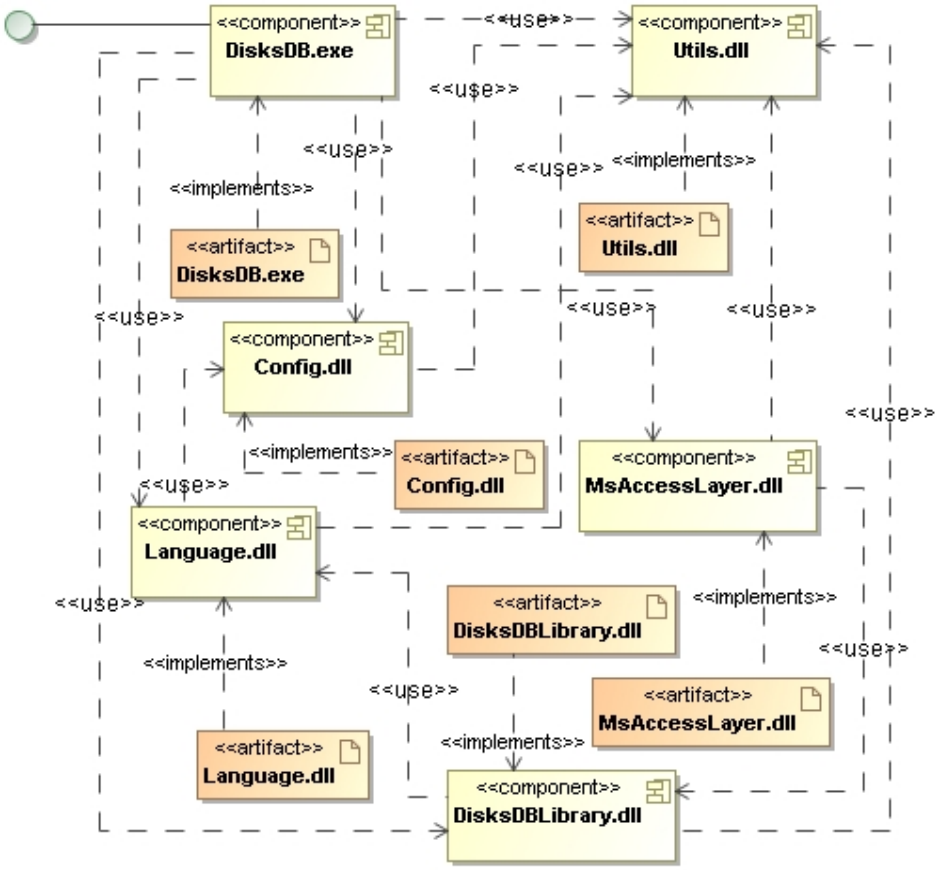


14 pav. *Duomenų klasių diagrama*

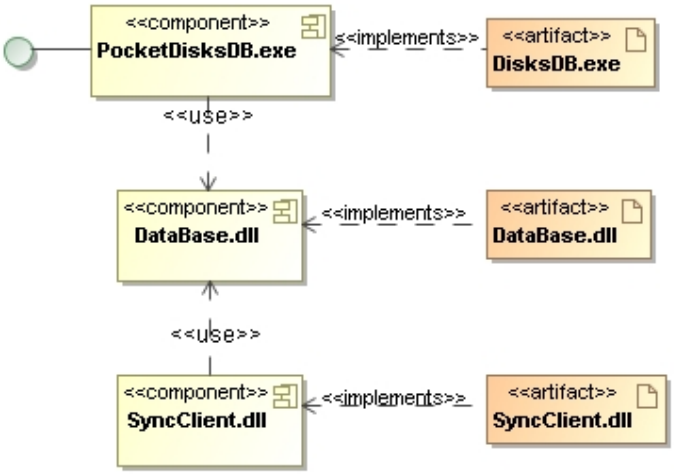
3.6 DisksDB realizacijos modelis

Sistemos komponentų galima pamatyti kaip susiję sistemoje esantys projektai tarpusavyje taip pat galima pamatyti kokiais artefaktais jie bus realizuoti.

Žemiau yra pateikti DisksDB sistemos komponentų diagramos atskirai pocketPC ir atskirai desktoPC aplinkoms (*15 pav.* ir *16 pav.*).

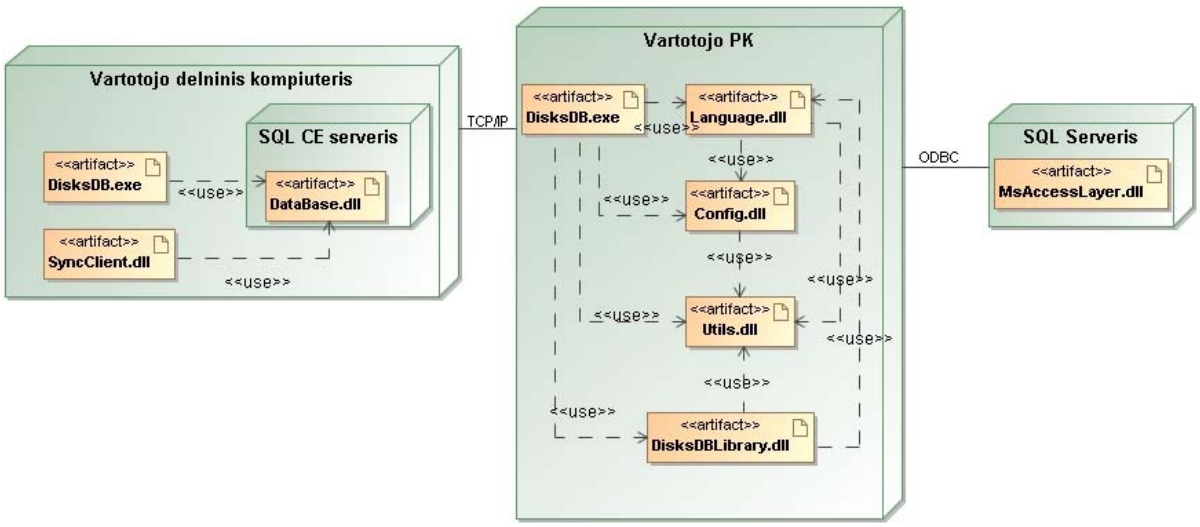


15 pav. *DesktopPC platformos komponentų diagrama*



16 pav. *PocketPC platformos komponentų diagrama*

Sistemos diegimo diagramoje (*17 pav.*) pateiktas bendras sukurtos sistemos modulių diegimas atskirose aplinkose bei jų tarpusavio ryšiai. Jų viduje parodyta kokie artefaktai sudaro sistemą bei atskurus jos modulius.



17 pav. *DisksDB sistemos diegimo diagrama*

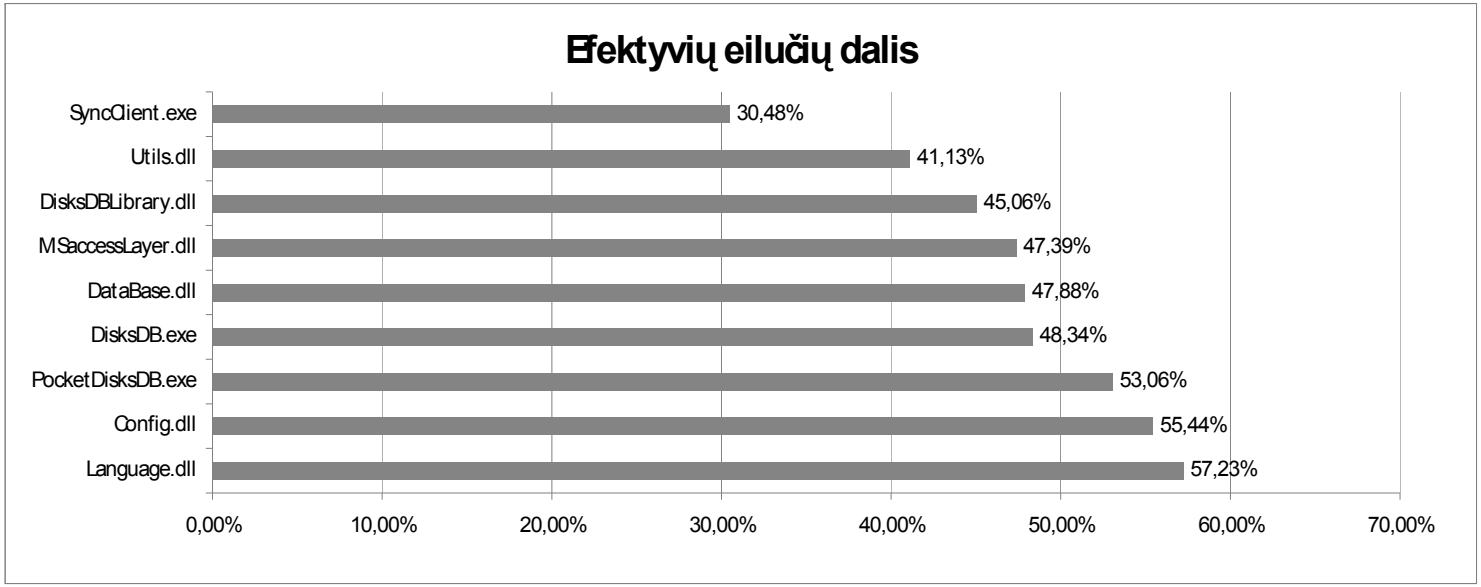
3.7 Paketų parametrai

Sukurtą sistemą sudaro du moduliai PocketPC ir DesktopPC. Šiuos du modulius sudaro paketai, kurie buvo gauti kompiliuojant programinį kodą. Sistemos kūrimo įrankio pagalba, buvo gauti pagrindiniai šių paketų duomenys (*10 lentelė.*). Šioje lentelėje pateikiami kiekvieno paketo failų, klasių vardų sričių ir pan. kiekiai. Taip pat pateikiamas kodo eilučių ir efektyvių eilučių kiekiai, bei efektyvių eilučių dalis.

Efektyvios eilutės tai tos eilutės, kurios nėra tuščios, ar kuriose nėra vien tik metodų skliaustų ar komentarų.

10 lentelė. *Sistemos paketų pagrindinė informacija*

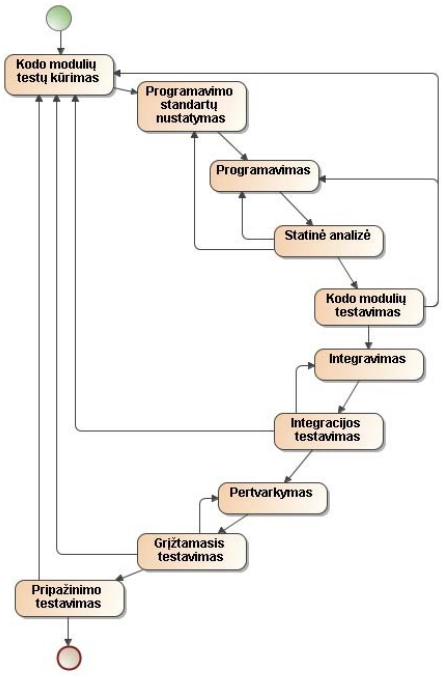
Platforma	Projektas	Failų skai-čius	Vardų sritys	.NET tipai					Eilučių kiekis	Efektyvių eilučių kiekis	Efektyvių eilučių dalis
				Klasių	Struktūrų	Sąrašų	Interfeisų	Delegatų			
DesktopPC	Config.dll	3	1	5	0	0	0	1	772	428	55.44%
	DisksDB.exe	63	4	65	2	0	4	17	13846	6693	48.34%
	DisksDBLibrary.dll	35	1	41	2	1	4	13	7058	3180	45.06%
	Language.dll	11	1	22	0	0	0	4	3175	1817	57.23%
	MSaccessLayer.dll	6	1	8	0	0	0	1	1973	935	47.39%
	Utils.dll	7	1	6	1	1	0	0	693	285	41.13%
PocketPC	DataBase.dll	6	1	4	0	1	1	0	1299	622	47.88%
	SyncClient.exe	2	1	1	0	0	0	0	105	32	30.48%
	PocketDisksDB.exe	9	4	8	0	0	0	4	1208	641	53.06%
	Bendra:	**142**	**15**	**160**	**5**	**3**	**9**	**40**	**30129**	**14633**	**48.57%**



18 pav. *Efektyvių eilučių dalis nuo bendros eolučių sumos paketuose*

4 Eksperimentinis DisksDB sistemos programos kodo modulių tyrimas

DisksDB sistemos kūrimo metu buvo vykdomas įvairių lygių testavimas. Kiekvieno lygio testai buvo kuriami pačioje pradžioje t.y. kodo modulių testų kūrimo fazėje (*5 pav.*). *19 pav.* patektos DisksDB kūrimo metu naudotos testavimo veiklos (jų aprašymas pateiktas skyriuje „*2.5 Testavimo metodai*“).



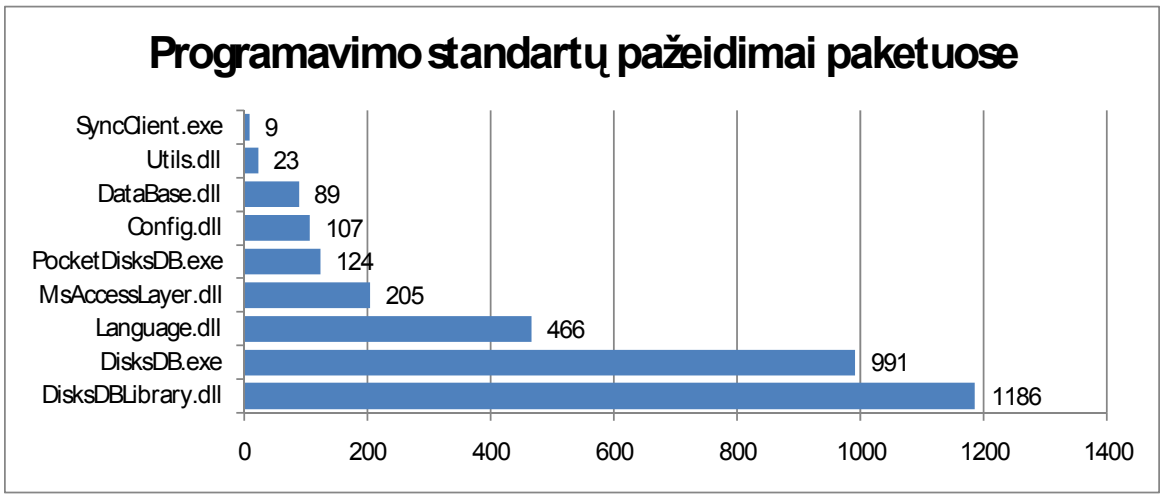
19 pav. *XP metodo testavimo veiklos*

4.1 Pateiktų pranešimų tyrimas

Atliekant DisksDB sistemos statinę analizę buvo naudojamos programavimo taisyklės – programavimo standartai. Šie standartai yra suskirstytas pagal svarbumą.

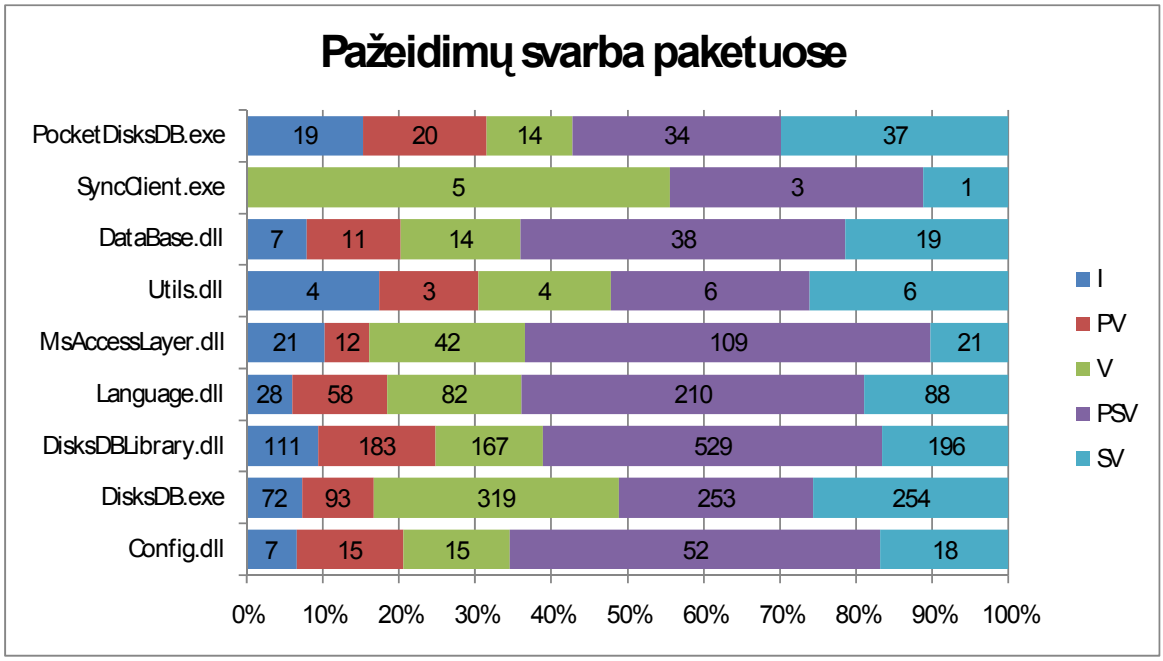
- I (angl. *Informational*) informacinis
- PV (angl. *Possible Violation*) galimas pažeidimas
- V (angl. *Violation*) pažeidimas
- PSV (angl. *Possible Severe Violation*) galimas didelis pažeidimas
- PV (angl. *Severe Violation*) didelis pažeidimas

Žemiau pateiktoje diagramoje pavaizduotas programavimo standartų pažeidimas kiekviename akete atskirai.



20 pav. *Pažeistų standartų kiekiai paketuose*

21 pav. pavaizduotoje diagramoje matyti, kad beveik visuose paketuose buvo padaryti galimi dideli pažeidimai.



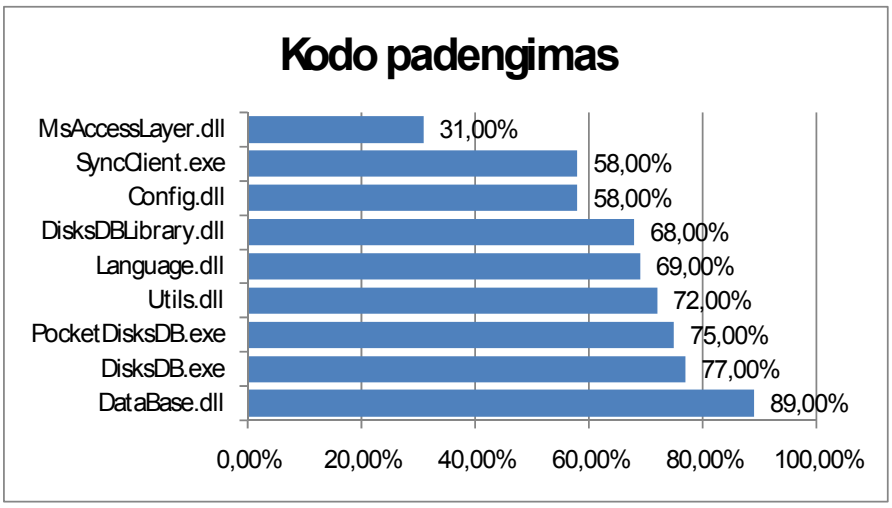
21 pav. *Pažeistų programavimo standartų svarba kiekviename pakete*

Iš pateiktų diagramų galima daryti išvadą, kad kuriant PĮ keliems asmenims, reikėt naudoti programavimo standartus, kadangi kyla rizika, kad atskiri sistemos moduliai gali sunkiai integruotis ar būti sunkiai taisomi.

4.2 Kodo padengimo tyrimas

Sukūrus kodo modulių testus bei PĮ, buvo patikrintas bendras viso bei atskirų paketų kodo padengimas.

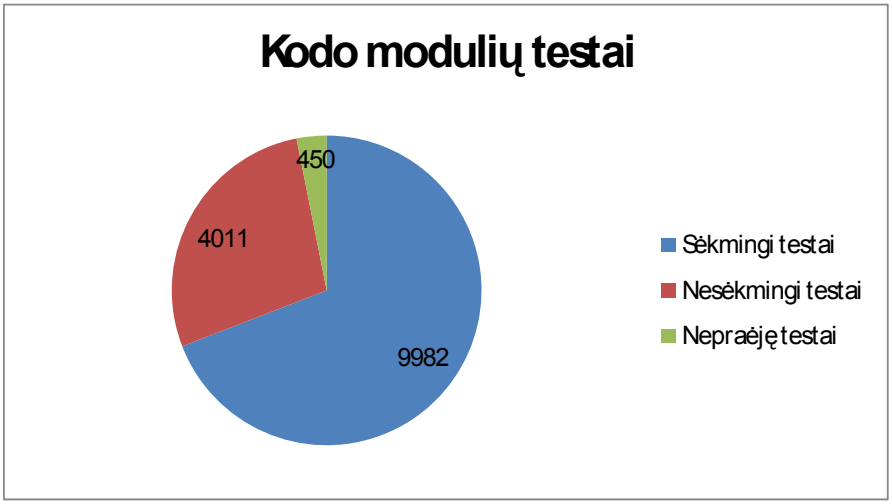
Testavimo metu įvykdytas bendras kodo padengimas 69%, o kiekvieno paketo padengimai atskirai pateikti *22 pav.*



22 pav. *Kodo padengimo palyginimas atskiriems paketams*

4.3 Testų vykdymo tyrimas

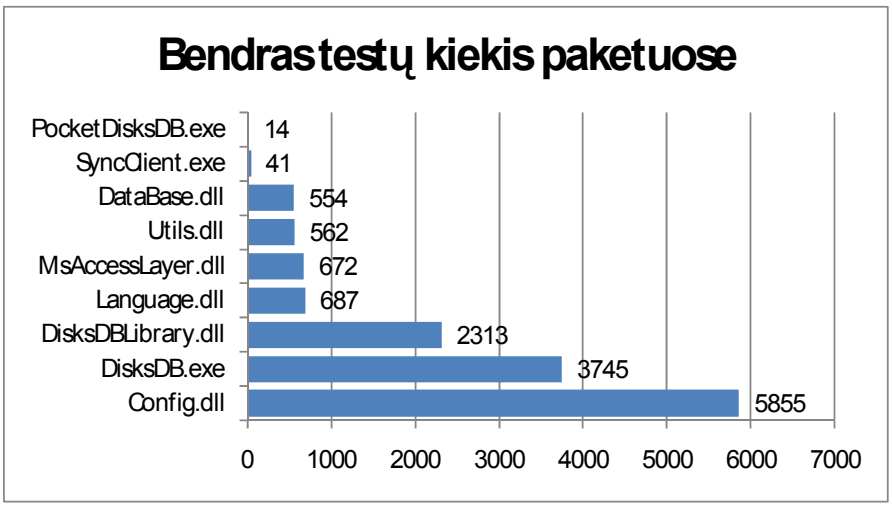
DisksDB kūrimo metu buvo sukurti 14443 kodo modulių testai. Atlikus kodo pertvarkymą ir galutiniame rezultate paleidus kodo modulių testu buvo gauti rezultatai, pateikti žemiau esančiose diagramose.



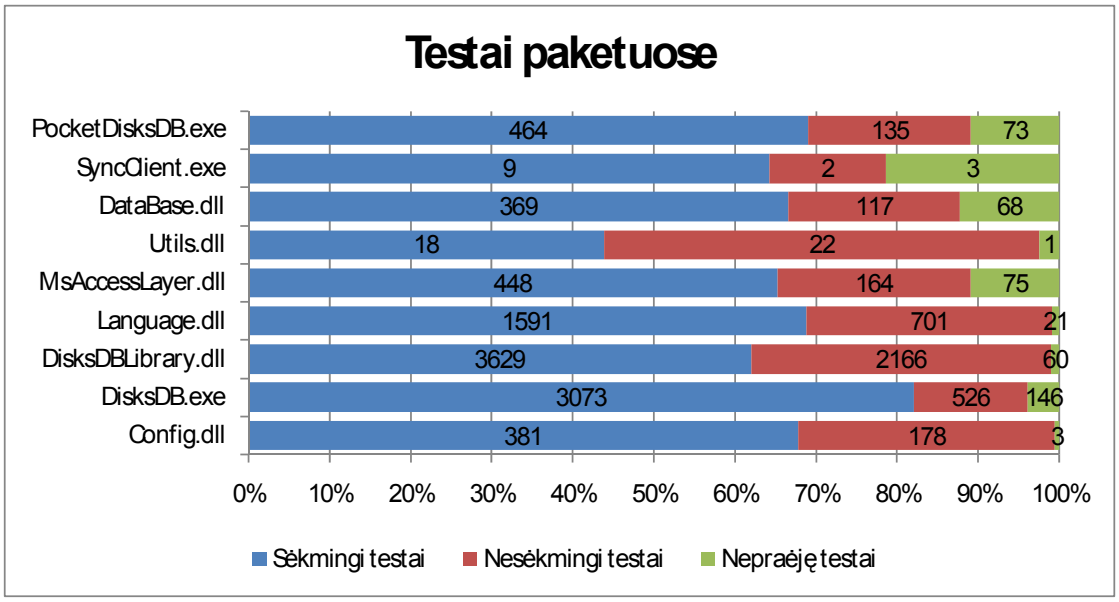
23 pav. *Testinių atvejų skaičiai ištestavus programos kodo modulius*

Kaip matome, be sėkmingų testų buvo ir nesėkmingų bei nepraėjusių testų. Taip įvyko todėl, kad buvo atliktas PĮ pertvarkymas.

Žemiau pateiktos testų kiekių palyginimo (žr. *24 pav.*) bei testų rezultatų pasiskirstymo (žr. *25 pav.*) kiekviename pakete diagramos

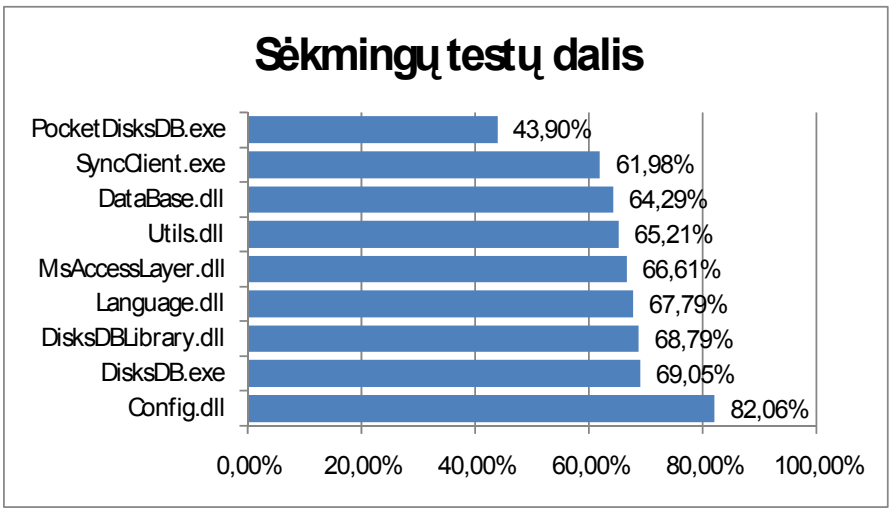


24 pav. *Testų kiekiai paketuose*



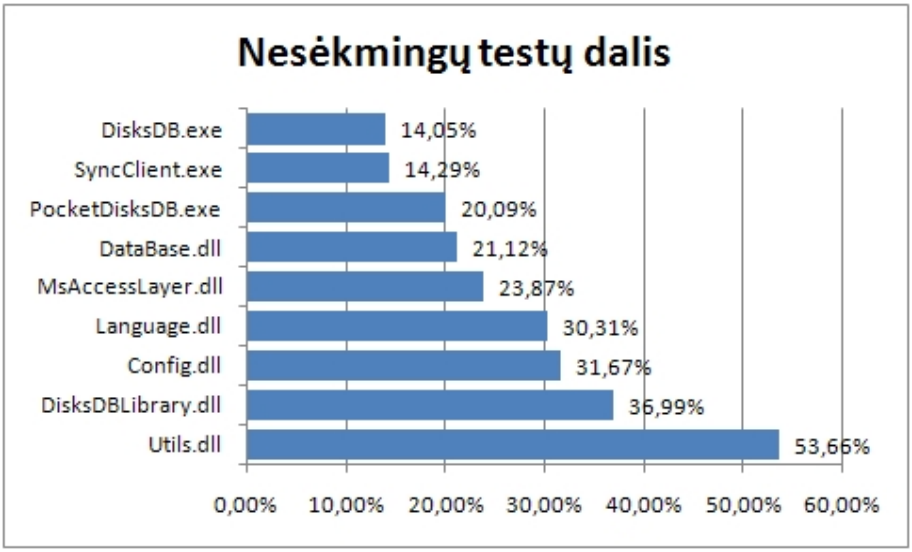
25 pav. *Testų rezultatai paketuose*

Atlikus išsamesnę analizę buvo nustatyta, kad daugiausia sėkmingų testų įvyko Config.dll faile, tuo tarpu PocketDisksDB.exe buvo įvykdyta mažiausiai. Kituose paketuose sėkmingų testų svyravimas nebuvo žymus (žr. *26 pav.*).

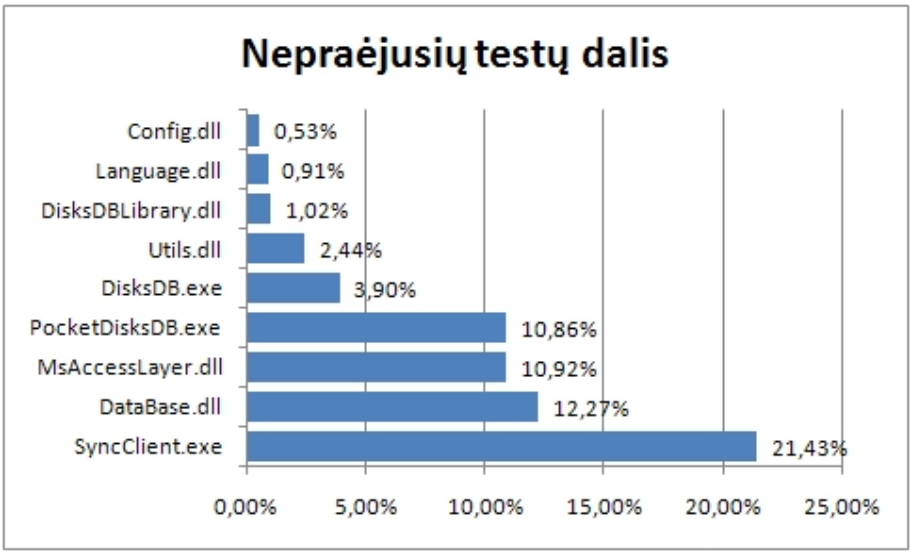


26 pav. *Sėkmingų testų palyginimas paketuose*

Panaši situacija ir su nesėkmingais bei nepraėjusiais testais (žr. *27 pav.* bei *28 pav.*).



27 pav. *Nesėkmingų testų palyginimas paketuose*



28 pav. *Nepraėjusių testų palyginimas paketuose*

5 Išvados

- Buvo suprojektuota ir sukurta diskų tvarkymo sistema skirta darbui ant dviejų platformų;
- Sistema buvo išanalizuota statinės analizės metodo pagalba;
- Atlikta automatinio kodo modulių testavimo rezultatų analizė;
- DiskDB sistemai įvykdytas bendras kodo padengimas 69% dėl nemažo kiekio neefektyvaus programinio kodo (komentarai, skliaustai ir pan.);
- Kai kuriuos testus gali sugeneruoti įrankiai, tačiau jie būna nepilni ir dėl to reikalingas programuotojo ar testuotojo įsikišimas.
- Jeigu testuojant programa patenka į amžiną ciklą ar neaprašytą išimtį testiniai atvejai negali būti korektiškai įvykdyti;
- Dalis testavimo buvo atlikta rankomis dėl ribotų testavimo įrankio galimybių;
- Iš anksto sudaryti kodo modulių testai leidžia turėti teisingesni kodą ir didesnį kodo padengimą, nei sudarant juos jau parašytam kodui;
- Kodo versijavimas, labai palengvina darbą, kada reikia dažnai keistis ar pildyti jau egzistuojančius kodo fragmentus;
- Automatinis kodo testavimas sumažina rizika sugadinti programinį kodą atliekant pertvarkas sename ar prijungiant naują kodo fragmentą;
- Rašyti kodą poromis ir dalintis patirtimi - tai geras būdas pagerinti programuotojų kvalifikaciją.

6 Literatūra

[1] Abrahamsson P., Salo O., Agile Software Development Methods: Review and Analysis, Otamedia Oy, Espo 2002

[2] Bhalerao S., Generalizing Agile Software Development Life Cycle, International Journal on Computer Science and Engineering Vol.1(3), 2009, 222-226

[3] Iacovelli A., Souveyet C., Framework for Agile Methods Classification, [žiūrėta 2010.04.10], prieiga internete: < http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-341/paper9.pdf >

[4] Kalermo J., Rissanen J., Agile Software development in theory and practice, University of Jyvaskyla, 2002 [žiūrėta 2010.03.10], prieiga internete: < http://www.cs.jyu.fi/sb/Publications/KalermoRissanen_MastersThesis_060802.pdf >

[5] IEEE Standard for Software Unit Testing [žiūrėta 2009.01.10], prieiga internete: < http://iteso.mx/~pgutierrez/calidad/Estandares/IEEE%201008.pdf >

[6] Beck K., Extreme Programming Explained: Embrace Change, Addison-Wesley, 2000

[7] Szalvay V., An Introduction to Agile Software Development, Danube technologies, 2004

[8] Extreme Programming (xp): A Gentle Introduction [žiūrėta 2009.01.10] prieiga internete: < http://www.extremeprogramming.org/ >

[9] Rengarajan M. S., Extreme Programing. The Way to Engineer, [žiūrėta 2009.01.10] prieiga internete: < http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci1243425,00.html >

[10] Bertolino A., Software Testing Research: Achievements, Challenges, Dreams

[11] Highsmith A. J., Agile Software Development Ecosystems, Pearson Education, Inc. 2002

[12] Laurance D., Mancl D., Extreme Programming and Agile Processes [žiūrėta 2009.09.10], prieiga internete: < http://princetonacm.acm.org/downloads/extreme_agile.html >

[13] Unit Testing Overview - Article source: CoDe, 2004 [žiūrėta 2009.01.10], prieiga internete: < http://www.code-magazine.com/Article.aspx?quickid=0411031>

[14] Stobie K., Too Darned Big to Test, Microsoft, [žiūrėta 2009.01.10], prieiga internete: <http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=277>

[15] Whittaker A. J., Black Box Debugging, Florida Institute of Technology;

[16] Thompson H. H., Security Innovation, [žiūrėta 2010.05.10], prieiga internete: <http://acmqueue.org/modules.php?name=Content&pa=showpage&pid=107&page=1>

[17] A Course in Black Box Software .Testing Examples of Test Oracles [žiūrėta 2009.01.10], prieiga internete: < http://www.testingeducation.org/k04/OracleExamples.htm>

[18] Ambler W. S., Agile Modeling (AM) Home Page->Agile Modeling and eXtreme Programming (XP) [interaktyvus], [žiūrėta 2009.03.10], prieiga internete: < http://www.agilemodeling.com/essays/agileModelingXP.htm>

[19] Ambler W. S., Agile Modeling (AM) Home Page->Agile Modeling and eXtreme Programming (XP)-> Agile Modeling Throughout the XP Lifecycle [interaktyvus], [žiūrėta 2009.08.10], prieiga internete: < http://www.agilemodeling.com/essays/agileModelingXPLifecycle.htm>

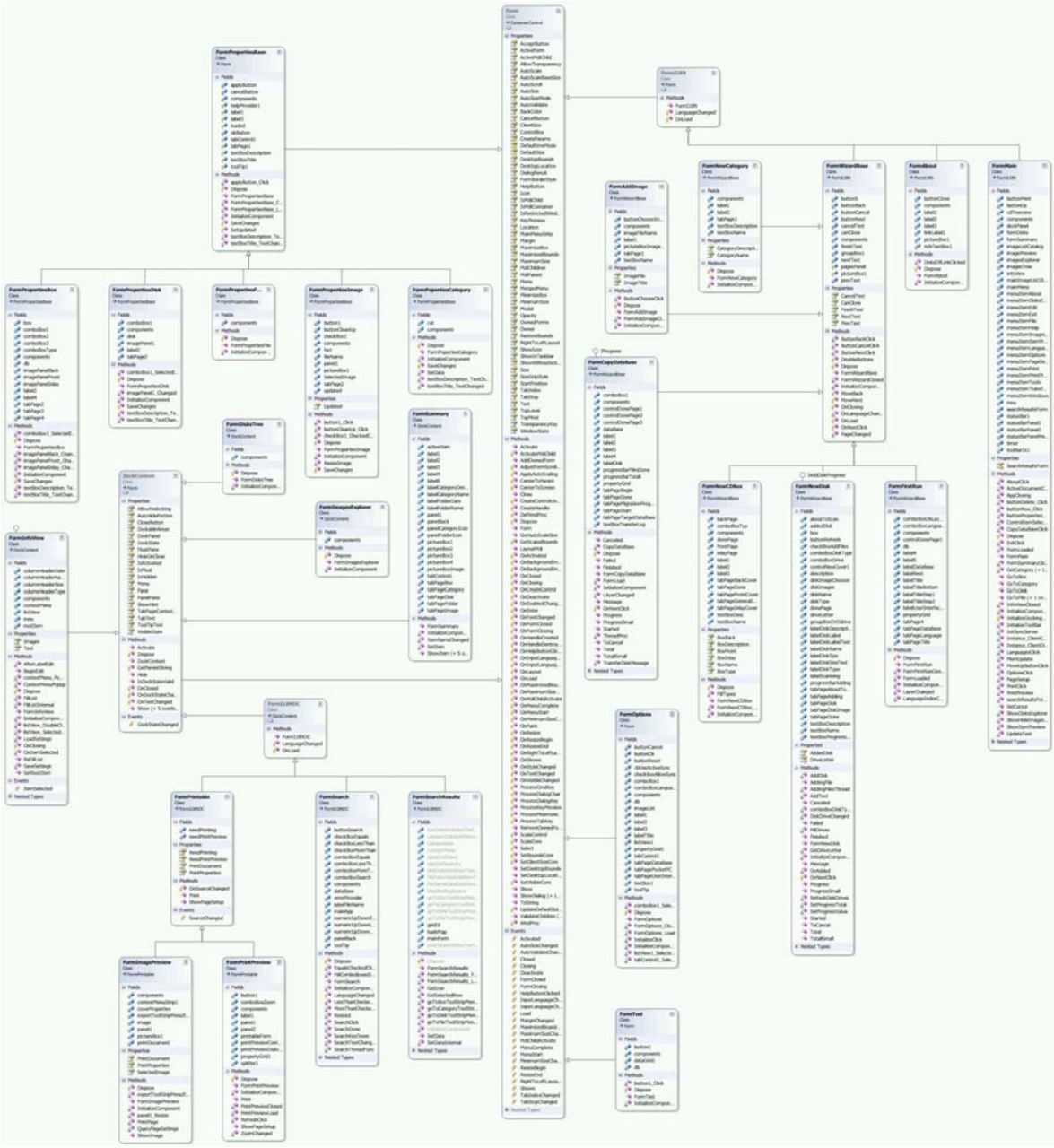
7 Terminų ir sutrumpinimų žodynas

Žemiau pateikiami dokumente naudojami sutrumpinimai ar sąvokos ir jų paaiškinimai:

Naudo-jamas sutrum-pinimas	Angliška sąvoka	Lietuviška Sąvoka	Aprašymas
		Metodologija	Pagrindinių taisyklių rinkinys, kuriuo vadovaujasi programuotojų komanda savo darbo organizavime
	Lightweight Methodology	Supaprastinta metodologija	PĮ kūrime naudojama kiek įmanoma mažiau taisyklių ir praktikų.
AM	Agile Modeling	Agile modeliavimas	Agile praktika paremta metodologija efektyviam PĮ modeliavimui ir dokumentavimui.
XP	Extreme Programming	Ekstremalus (Ribinis) programavimas	Agile PĮ kūrimo metodas, kuris didelį dėmesį skiria komandiniam darbui.
FDD	Feature Driven Development	Funkciniais reikalavimais paremtas programų kūrimas	Agile PĮ kūrimo metodas, kuris yra praktiškas ir orientuotas į klientą bei architektūrą. Funkcionalumas tai nedidelė kliento įvertinta funkcija išreikšta forma <veikla><rezultatas><objektas>. Pavyzdžiui <Suskaičiuoti bendrą pardavimą><Patvirtinti vartotojo slaptažodį><Patvirtinti pardavimų transakciją klientu>
DSDM	Dynamic System Driven Development	Dinaminių sistemų kūrimo metodas	Agile PĮ kūrimo metodas paremtas greito PĮ kūrimo metodu (RAD - Rapid Application Development). DSDM yra iteracinis inkrementinis procesas išsiskiriantis pastoviu vartotojo dalyvavimu procese.
ASD	Adaptive Software Development	Adaptyvus programų kūrimas	Agile PĮ kūrimo metodas aprašantis kūrybinį procesą, kuris numato lankstumo ir praktiškumo galutinio produkto pristatyme.
PP	Pragmatic Programming	Praktiškas programavimas	Agile PĮ kūrimo metodas, kuris remiasi praktiškais rezultatais
	Crystal		Agile PĮ kūrimo metodas, kuriame svarbiausia komandinis darbas, komunikavimas, paprastumas, dažnas pakartotinis proceso nustatinėjimas ir tobulinimas. Crystal skatina dažną veikiančio kodo leidimą, aktyvų užsakovo dalyvavimą procese, adaptyvumą, biuroklatijosi ir kitų blaškančių kliūčių pašalinimą.
	Scrum		Unikalus Agile PĮ kūrimo metodas, kuris remiasi projekto progresu realiame pasaulyje - ne geriausiu spėjimu ar iš piršto laužta prognoze - planuoti ir nustatyti versijų išleidimus.
SLC	System Life Cycle	Sistemos gyvavimo ciklas	Baigtinis rinkinys etapų ir žingsnių, kuriuos sistema gali pereiti savo per savo gyvavimo istorija.
		Idealus programavimo laikas	Tai laikas per kurį yra realizuojamas scenarijuje aprašytas funkcionalumas, jeigu niekas neblaško, neskiria kitų užduočių ir jeigu tiksliai yra žinoma, ką reikia daryti.
	Regression testing	Grįžtamasis testavimas	Procesas, skirtas pakartotinai testuoti programinę įrangą (kuri buvo kažkaip pakeista), užtikrinti, kad neatsirado naujų defektų po paskutinio pakeitimo bei įsitikinti, jog testuojama sistema vis dar atitinka specifikaciją.
		Testavimas	Patikrinamai ar sistema/komponentas vis dar atitinka su juo susijusius reikalavimus.
	Code Unit	Kodo modulis	Smulkiausi vykdomi kodo vienetai – funkcijos ir procedūros.
	Test Stub	Testavimo kamštis	Programos, kurios imituoja neužbaigto PĮ modulio veikimą, o rezultatą duoda tokį kokį duotų tikrasis modulis
	Test driver	Testavimo valdiklis	Tai programinio kodo dalis, kuri perduoda testavimo atvejus, testuojamam kodo moduliui.

8 Priedai

1 priedas. DisksDB pagrindinė klasių diagrama



pav. 29 *Bendroji klasių diagrama*

3 priedas. Parasoft .Test sugeneruotos ataskaitos fragmentas

```
Project: DiskDB   [passed=9140 failed=3757 incomplete=306 / tested=13203 / total=13203]  [IL=69
SRC=N/A (%)]
    Config.dll  [passed=381 failed=178 incomplete=3 / tested=562 / total=562]  [IL=58 SRC=N/A
(%)]
      DisksDB  [passed=381 failed=178 incomplete=3 / tested=562 / total=562]  [IL=58 SRC=N/A (%)]
        Config  [passed=381 failed=178 incomplete=3 / tested=562 / total=562]  [IL=58 SRC=N/A
(%)]
          Config  [passed=39 failed=46 incomplete=0 / tested=85 / total=85]  [IL=33 SRC=N/A (%)]
            Config()  [passed=0 failed=5 incomplete=0 / tested=5 / total=5]  [IL=5 SRC=N/A (%)]
              Config.0.scn  [IL=5 SRC=N/A (%)]
                [Pre-conditions ] statics=DisksDB.Config.Config.appId ="\"
```

```
                [Return         ] {ANY}
                [Post-conditions] statics=DisksDB.Config.Config.appId ="\"
                [Status         ] fail
                [Outcome        ]
                Unexpected exception of type System.ArgumentException during method/constructor
call
                       at System.Data.DataKey.CheckState()
                       at System.Data.DataKey.Create(DataColumn[] columns, Int32[] sortOrders)
                       at System.Data.UniqueConstraint.Create(String constraintName,
DataColumn[] columns)
                       at System.Data.UniqueConstraint..ctor(String name, DataColumn[] columns,
Boolean isPrimaryKey)
                       at DisksDB.Config.ConfigDataTable.InitClass()
                       at DisksDB.Config.ConfigDataTable..ctor()
                       at DisksDB.Config.DataSetConfig.InitClass()
                       at DisksDB.Config.DataSetConfig..ctor()
                       at DisksDB.Config.Config..ctor()

                Post condition failed for return value
                    expected value:     {ANY}
                    exception thrown:   {m_paramName=null}
  void Save()  [passed=10 failed=0 incomplete=0 / tested=10 / total=10]  [IL=64 SRC=N/A (%)]
              Save.0.scn  [IL=64 SRC=N/A (%)]
                [Pre-conditions ] this={dsCfg=null,cfgFileName="a",configFileExists=True}
                [Return         ]
                [Post-conditions] this={dsCfg=null,cfgFileName="a",configFileExists=True}
                [Status         ] pass

              Save.1.scn  [IL=64 SRC=N/A (%)]
                [Pre-conditions ] this={dsCfg={...},cfgFileName="",configFileExists=False}
                [Return         ]
                [Post-conditions] this={dsCfg={...},cfgFileName="",configFileExists=False}
                [Status         ] pass

              Save.2.scn  [IL=64 SRC=N/A (%)]
                [Pre-conditions ] this={dsCfg=null,cfgFileName=null,configFileExists=True}
                [Return         ]
                [Post-conditions] this={dsCfg=null,cfgFileName=null,configFileExists=True}
                [Status         ] pass


IListItem IListItem  [passed=3 failed=5 incomplete=2 / tested=10 / total=10]  [IL=100 SRC=N/A
(%)]
              IListItem get()  [passed=3 failed=5 incomplete=2 / tested=10 / total=10]  [IL=100
SRC=N/A (%)]
                get_IListItem.0.scn  [IL=100 SRC=N/A (%)]
                  [Pre-conditions
] this=DisksDB.UserInterface.ListItem(DisksDB.UserInterface.IListItem)({...})
                  [Return
] {baseObject=null,canBeDeleted=False,canBeRenamed=False,typeName=null,size=0,date={...},hasPrope
rties=False,canContainDisk=False,canContainBox=False,canContainCategory=False,canContainImage=Fal
se,isNodesLoaded=True,disk=null,files={...},db=null}
                  [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-1,BackColor={...},type=null,size=0,date={...},iListItem={...}}
                  [Status         ] pass

                get_IListItem.1.scn  [IL=100 SRC=N/A (%)]
                  [Pre-conditions
] this=DisksDB.UserInterface.ListItem(DisksDB.UserInterface.IListItem)({...})
                  [Return
] {baseObject={...},canBeDeleted=True,canBeRenamed=True,typeName="Category",size=-
1,date={...},hasProperties=True,canContainDisk=False,canContainBox=True,canContainCategory=True,c
anContainImage=False,nodesLoaded=False,db=null,cat={...},trv={...}}
                  [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-1,BackColor={...},type="Category",size=-1,date={...},iListItem={...}}
                  [Status         ] fail
                  [Outcome        ]
                  Post condition failed for this value
                      expected value:     {date={...}}
                      obtained value:     {date={...}}


                get_IListItem.2.scn  [IL=100 SRC=N/A (%)]
```

```
                [Pre-conditions
] this=DisksDB.UserInterface.ListItem(DisksDB.UserInterface.IListItem)({...})
                [Return
] {baseObject=null,canBeDeleted=False,canBeRenamed=False,typeName=null,size=0,date={...},hasPrope
rties=False,canContainDisk=False,canContainBox=False,canContainCategory=False,canContainImage=Fal
se,isNodesLoaded=False,disk={...},files={...},db=null}
                [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-1,BackColor={...},type=null,size=0,date={...},iListItem={...}}
                [Status         ] fail
                [Outcome        ]
                Post condition failed for return value
                    expected value:     {disk={...}}
                    obtained value:     {disk={...}}


              get_IListItem.3.scn  [IL=0 SRC=N/A (%)]
                [Pre-conditions
] this=DisksDB.UserInterface.ListItem(DisksDB.UserInterface.IListItem)({...})
                [Return
] {baseObject=null,canBeDeleted=False,canBeRenamed=False,typeName=null,size=0,date={...},hasPrope
rties=False,canContainDisk=False,canContainBox=False,canContainCategory=False,canContainImage=Fal
se,isNodesLoaded=True,db={...},box={...},trv={...}}
                [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-1,BackColor={...},type=null,size=0,date={...},iListItem={...}}
                [Status         ] incomplete
                [Outcome        ]
                Pre-condition creation exception of type System.ArgumentException
                        at System.Drawing.Icon..ctor(Stream stream)
                        at DisksDB.UserInterface.FileIcons..ctor(ImageList imageList)
                        at DisksDB.UserInterface.TreeViewCatalog..ctor(ImageList imgList)


              get_IListItem.4.scn  [IL=100 SRC=N/A (%)]
                [Pre-conditions
] this=DisksDB.UserInterface.ListItem(DisksDB.UserInterface.IListItem)({...})
                [Return
] {baseObject=null,canBeDeleted=False,canBeRenamed=False,typeName=null,size=0,date={...},hasPrope
rties=False,canContainDisk=False,canContainBox=False,canContainCategory=False,canContainImage=Fal
se,isNodesLoaded=False,disk={...},files={...},db={...}}
                [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-1,BackColor={...},type=null,size=0,date={...},iListItem={...}}
                [Status         ] fail
                [Outcome        ]
                Post condition failed for return value
                    expected value:     {disk={...}}
                    obtained value:     {disk={...}}


              get_IListItem.5.scn  [IL=100 SRC=N/A (%)]
                [Pre-conditions
] this=DisksDB.UserInterface.ListItem(DisksDB.UserInterface.IListItem)({...})
                [Return
] {baseObject=null,canBeDeleted=False,canBeRenamed=False,typeName=null,size=0,date={...},hasPrope
rties=False,canContainDisk=False,canContainBox=False,canContainCategory=False,canContainImage=Fal
se,isNodesLoaded=True,disk=null,files={...},db={...}}
                [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-1,BackColor={...},type=null,size=0,date={...},iListItem={...}}
                [Status         ] pass

              get_IListItem.6.scn  [IL=0 SRC=N/A (%)]
                [Pre-conditions
] this=DisksDB.UserInterface.ListItem(DisksDB.UserInterface.IListItem)({...})
                [Return
] {baseObject=null,canBeDeleted=False,canBeRenamed=False,typeName=null,size=0,date={...},hasPrope
rties=False,canContainDisk=False,canContainBox=False,canContainCategory=False,canContainImage=Fal
se,nodesLoaded=False,db=null,cat=null,trv={...}}
                [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
```

```
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-1,BackColor={...},type=null,size=0,date={...},iListItem={...}}
                  [Status         ] incomplete
                  [Outcome        ]
                  Pre-condition creation exception of type System.ArgumentException
                          at System.Drawing.Icon..ctor(Stream stream)
                          at DisksDB.UserInterface.FileIcons..ctor(ImageList imageList)
                          at DisksDB.UserInterface.TreeViewCatalog..ctor(ImageList imgList)


               get_IListItem.7.scn  [IL=100 SRC=N/A (%)]
                  [Pre-conditions
] this=DisksDB.UserInterface.ListItem(DisksDB.UserInterface.IListItem)({...})
                  [Return
] {baseObject=null,canBeDeleted=False,canBeRenamed=False,typeName=null,size=0,date={...},hasPrope
rties=False,canContainDisk=False,canContainBox=False,canContainCategory=False,canContainImage=Fal
se,isNodesLoaded=False,db={...},box={...},trv={...}}
                  [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-1,BackColor={...},type=null,size=0,date={...},iListItem={...}}
                  [Status         ] fail
                  [Outcome        ]
                  Post condition failed for return value
                      expected value:     {box={...}}
                      obtained value:     {box={...}}


               get_IListItem.8.scn  [IL=100 SRC=N/A (%)]
                  [Pre-conditions
] this=DisksDB.UserInterface.ListItem(DisksDB.UserInterface.IListItem)({...})
                  [Return
] {baseObject={...},canBeDeleted=True,canBeRenamed=True,typeName="Image",size=-
1,date={...},hasProperties=True,canContainDisk=False,canContainBox=False,canContainCategory=False
,canContainImage=False,img={...},imgFact=null}
                  [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-1,BackColor={...},type="Image",size=-1,date={...},iListItem={...}}
                  [Status         ] fail
                  [Outcome        ]
                  Post condition failed for this value
                      expected value:     {date={...}}
                      obtained value:     {date={...}}


               get_IListItem.9.scn  [IL=100 SRC=N/A (%)]
                  [Pre-conditions
] this=DisksDB.UserInterface.ListItem(System.String,System.String,System.Int64,System.DateTime)("
a", null, 9223372036854775807, {...})
                  [Return         ] null
                  [Post-
conditions] this={Font={...},SubItems={...},Text="a",UseItemStyleForSubItems=True,Selected=False,
ForeColor={...},Checked=False,Index=-
1,ImageList=null,Focused=False,Tag=null,ListView=null,Bounds={...},ImageIndex=-
1,StateImageIndex=-
1,BackColor={...},type=null,size=9223372036854775807,date={...},iListItem=null}
                  [Status         ] pass
```