VILNIAUS UNIVERSITETAS
KAUNO HUMANITARINIS FAKULTETAS

INFORMATIKOS KATEDRA

VERSLO INFORMATIKOS STUDIJŲ PROGRAMA

Kodas 62109P101

AUDRIUS PARAŽINSKAS

MAGISTRO BAIGIAMASIS DARBAS

PROGRAMŲ SISTEMŲ JUDRIŲJŲ (AGILE) KŪRIMO METODŲ ANALIZĖ IR TYRIMAS

Kaunas 2008

VILNIAUS UNIVERSITETAS
KAUNO HUMANITARINIS FAKULTETAS

INFORMATIKOS KATEDRA

VERSLO INFORMATIKOS STUDIJŲ PROGRAMA

Kodas 62109P101

AUDRIUS PARAŽINSKAS

MAGISTRO BAIGIAMASIS DARBAS

PROGRAMŲ SISTEMŲ JUDRIŲJŲ (AGILE) KŪRIMO METODŲ ANALIZĖ IR TYRIMAS

Leidžiama ginti_____
Magistrantas_____
(parašas)

Darbo vadovas_____
(parašas)
prof. dr. Rimantas Butleris
(darbo vadovo mokslinis laipsnis, mokslo pedagoginis vardas, vardas ir pavardė)

Darbo įteikimo data_____

Registracijos Nr._____

Kaunas 2008

Turinys

SANTRUMPŲ SĄRAŠAS

DB – duomenų bazė

IS – informacinė sistema

IT – informacinės technologijos

XP – Ekstremalus programavimas (Extreme Programming)

DSDM – Dinaminis sistemos kūrimo metodas (Dynamic System Development Method)

FDD – Savybių „vairuojamas“ kūrimo procesas (Feature Driven Development)

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

SANTRAUKA

PARAŽINSKAS, Audrius. (2008) *Analysis and research of Agile software development methods***.** MBA Graduation Paper. Kaunas: Vilnius University, Kaunas Faculty of Humanities, Department of Informatics. 60 p.

In recent years more and more companies use the Agile methodology to create software. In this paper I analyze the system creation models and Agile methodology. The main object of the paper is the analysis of Agile methods and their application in creating information systems.

Analysis of methods, the system creation methods was represented in the first part of the paper. The advantages and disadvantages of traditional creation methods and Agile methods were described in this part of the paper. The Agile methodology was chosen for the further analysis

According to Agile methodology a new method was introduced in the second part of the paper. It was represented in order to improve and speed up the creation of information systems.

The experiment was made in the third part of the paper. It was carried out in order to check the effectiveness of this new method for creating information systems.

The results of the project. It is important to note that this new method is implemented faster and cost less compared to the traditional system creation methods. The main point of this method is that requirements of the user can change at any time but it will not change the cost and duration of the project.

The project consists of 88 pages, 31 pictures and 25 tables.

ĮVADAS

Dauguma programinės įrangos projektų nepasiteisina, nes problemų atsiranda reikalavimų apibrėžime ir valdymo srityje. Kad to išvengti, reikėtų įtraukti vartotoją į kūrimo procesą, kad galima būtų teisingai suprasti keliamus reikalavimus.

Prieš kuriant programą, yra pasirenkamas tam tikras metodas. Metodo pasirinkimas priklauso nuo projekto dydžio, finansavimo bei užsakovo poreikių. Pavyzdžiui, programinės įrangos kūrimo kaina pagal tradicinius reikalavimų inžinerijos principus yra nustatoma pagal reikalavimus. Pagal reikalavimus nustatomas projekto trukmė. Kadangi reikalavimai gali kisti vidury projekto, todėl kaina gali labai iškilti, nes gali būti taip, kad reikės keisti ne tik programos kodą, bet reikalavimų aprašymus bei projekto modelius.

Tam yra skirtos Agile metodologijos, kurios siūlo kuo mažiau dėmesio skirti dokumentavimui. Programinė įranga kuriama trumpomis iteracijos, o vartotojo poreikiai išsiaiškinami bendraujant tiesiogiai. Tarpinių programinės įrangos versijų vertinimas atliekamas kiekvienos iteracijos pabaigoje (D. Šilingas, 2008).

Darbo objektas – Agile metodologija.

Tyrimo tikslas yra apžvelgti ir išanalizuoti judriųjų programų sistemų kūrimo metodus ir sudaryti metodą programų sistemų kūrimui..

Pagrindiniai uždaviniai, į kuriuos bus siekiama atsakyti tyrimo eigoje, yra:

- Išanalizuoti magistrinio darbo objektą
- Apžvelgti Agile veiklos modeliavimo principus;
- Aprašyti Agile požiūrį į modeliais paremtą IS kūrimą;
- Atlikti Agile metodų analizę;.
- Parinkti tinkamiausią Agile metodą, skirtą informacinėms sistemoms kurti.

- Patikrinti metodo efektyvumą, projektuojant UAB „Kemira Lifosa“ žaliavų ir produkcijos srautų valdymą.

Darbe buvo naudoti tyrimo metodai: duomenų analizės metodas – analizuojami lentelių duomenys, kiti įvairių šaltinių duomenys; lyginamosios analizės metodas - lyginami ir vertinami produktai, panašūs bruožai, savybės; sintezės ir apibendrinimo metodai - apibendrinant informacijos šaltinius, skyrius ir straipsnius, pateikiamos išvados, sprendimai. Šių metodų pagalba buvo siekiama užsibrėžtų tikslų ir uždavinių įgyvendinimo.

Darbas susideda iš šių dalių:

Analitiniame skyriuje aprašyta, kas yra Agile metodologija. Toliau darbe pateikta Agile metodų analizė.. Pateikta detali metodų analizė, taip pat išskirti privalumai ir trūkumai.. Sudaromas metodas informacinėms sistemoms kurti. Eksperimentinėje dalyje patikrinamas metodo efektyvumas, modeliuojant situaciją pagal naują metodą.

Darbą sudaro 88 puslapiai ir 25 lentelės, 31 paveikslai.

1. TEORINIS/ANALITINIS SKYRIUS

1.1 Agile veiklos modeliavimo principai

Prioritetas žmonėms, o ne įrankiams ar technikai

Šią dalį norėčiau pradėti nuo citatos: “The quality of the people on a project, and their organization and management, are much more important factors in success than are the tools they use or the technical approaches they take.” (F. Brooks, 1995) Tokia yra tiesa, nes būtent žmonės kuria vysto ir galų gale naudoja organizacijos veiklos architektūrą. Remiantis šiuo principu išryškėja dar vienas svarbus aspektas – norint sukurti kokybišką IS komunikacija su klientais t.y. būsimais vartotojais yra būtinybė. Geriausia būtų daryti pastovius (kasdieninius) susitikimus su kliento atstovais, tačiau tai padaryti realiai yra labai sudėtinga, galima naudotis šiuolaikinėmis technologijomis kaip video konferencijos ar elektroninis paštas.

Agile veiklos modeliuotojus galėtume įvardinti ir kaip ryšininkus („go to guys“), kurie didina žinių paplitimą organizacijos viduje. Tai gali būti tarp formalios ir neformalios valdžios, tarp komandų, skyrių ar jų narių.

Taigi Agile metodologijos sėkmė labai priklauso nuo žmonių, vadinasi jie turi būti motyvuoti ir atsakingi, neskantant patirties ir įgūdžių.

Kuo Agile veiklos modeliavimas yra kitoks nei tradiciniai? Pagrindinius skirtumus nusako žemiau pateikti principai:

1.Prioritetas žmonėms, o ne įrankiams ar technikai;
2.KISS (Keep It Simple And Stupid);
3.Darbas iteracijomis;
4.„Pasiraitoti rankoves“;
5.Globalus požiūris į sistemą;

6. Sukurti veiklos architektūrą, kuri būtų patraukli vartotojui (W. Cunningham, 2001)

Galime pastebėti, jog dalis principų yra paimti iš Agile manifesto, kuris buvo sukurtas, kaip atsakas į dažnas ir pasikartojančias IS kūrimo problemas bei iš to sekančias nesėkmes. Taigi pateiktų principų suvokimas ir yra Agile veiklos modeliavimo šerdis, todėl toliau referate juos aptarsime plačiau.

KISS principas

Vienas iš esminių akcentų kalbant apie Agile, yra siekimas kiek įmanoma išlaikyti paprastumą. Tai apima tiek programavimą, modeliavimą, testavimą etc. Taigi kuriami veiklos modeliai turėtų būti pakankamai geri ir informatyvūs, tačiau jie neprivalo būti tobuli. Nors tai gali atrodyti, kaip nevisai svarbus aspektas, tačiau ne tokia reta yra "auksinimo" problema, kai siekiant padaryti idealų artefaktą prarandama daug laiko, tačiau bendrame rezultate patobulinimai nėra to verti. Be to, kadangi viskas labai greitai kinta, todėl neįmanoma sukurti tokio modelio, kuris vienodai gerai atspindėtų tam tikrą sritį pakankamai ilgą laiką. Kurdami paprastus artefaktus galime tikėtis, jog jie bus lengvai suprantami, modifikuojami ir nepasens per naktį.

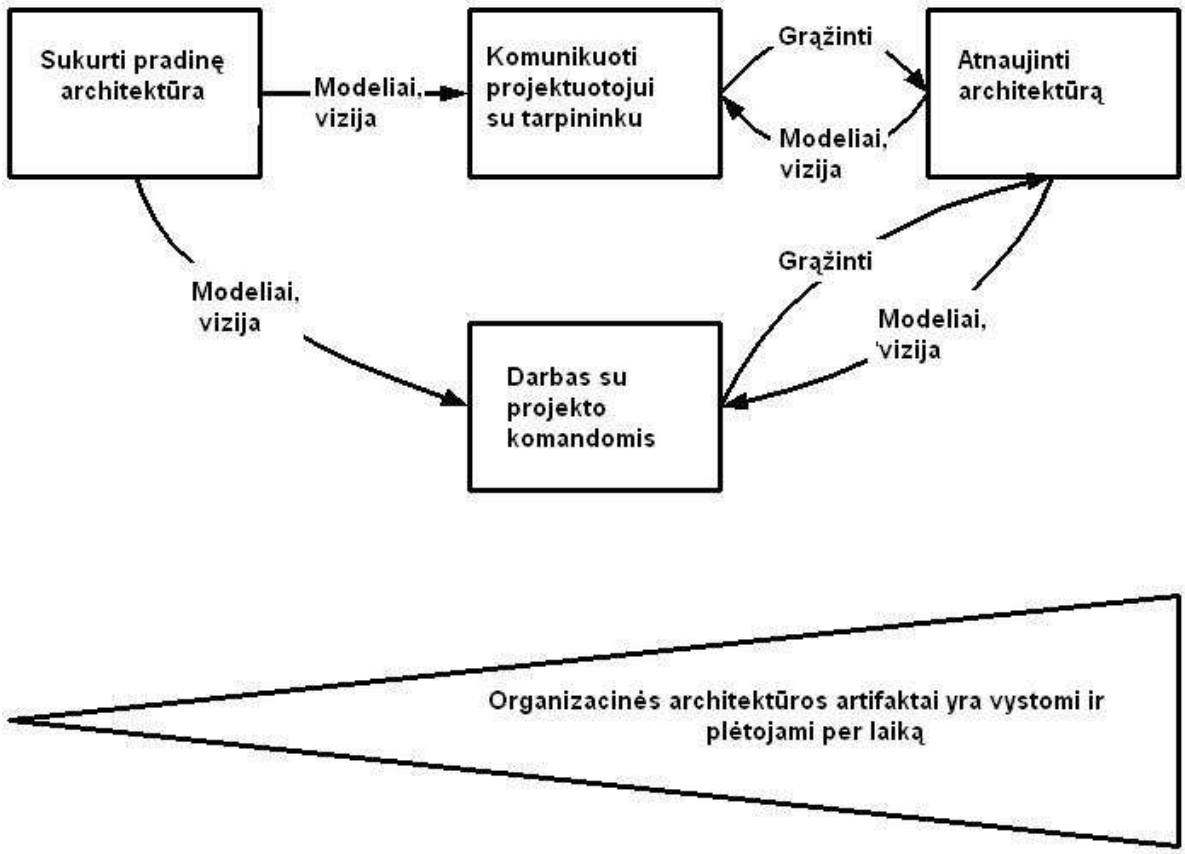
Darbas iteracijomis

Agile veiklos modeliuotojai turėtų dirbti iteraciškai. Tai reiškia, jog sukuriamas paprastas modelis ir jis naudojamas, kol adekvačiai aprašo modeliuojamą procesą, o susidarius naujai situacijai, modelis modifikuojamas, ir taip atliekamas tiek kartų kiek reikia t.y. kol pasiekiamas optimalus ir klientą tenkinantis rezultatas. Yra įprasta sukurti keletą tos pačios srities modelių pvz. klasių diagramos, sekų bei būsenų. Tai leidžia suvokti problemą skirtingais pjūviais ir atsiradus pakeitimams, juos įvykdyti tinkamoje vietoje.

Darbas atliekamas mažais žingsneliais, modeliuojama tiek kiek reikia artimiausiai ateičiai, negalvojant, kokius reikalavimus ar pakeitimus užsakovas pateiks po mėnesio t.y. sprendžiamos šios dienos problemos. Toks požiūris suteikia labai didelį lankstumą ir reagavimo į pokyčius greitį. Savo ruožtu tai gali sukelti kontroliavimo ir koordinavimo problemų, ypač jeigu probleminė sritis yra plati (sudėtinga organizacijos verslo procesų struktūra). Be to, toks požiūris atrodo trumparegiškas ir visada egzistuoja pavojus nepastebėti kažko svarbaus - „jeigu važiuodamas dviračiu žiūrėsi tik po ratais, nepastebėsi posūkio už 10

metrų“. Na, bet šiuos nuogąstavimus stengiamasi paneigti išskiriant „nulinę“ iteraciją, kurią aptarsime vėliau.

Manau tikslinga pateikti grafinę iliustracija, kaip realizuojama Agile metodologija skirta veiklos modeliavimui. Taigi pirmasis etapas yra vizijos ir preliminarių modelių sukūrimas. Būtent preliminarių, kadangi modeliai turi būti pakankamai geri ir leistų eiti pirmyn, tačiau jie neturi būti idealūs.



Pav. 1. Agile Model Driven Development (AMDD) veiklos modeliavimui,

Šaltinis: Ambler S.W. (2006) http://www.agiledata.org/essays/enterpriseArchitecture.html

Kaip matome tik pradinė iteracija nesikartoja, be to su kiekviena iteracija sukuriami nauji artefaktai ir papildomi seni, kad tiksliau atspindėtų naują situaciją, taigi laikui bėgant supratimas apie sistemą pastoviai auga ir gilėja. Vėliau referate AMDD bus aprašytas plačiau.

„Pasiraitoti rankoves“

Agile primygtinai rekomenduoja, kuo greičiau sumodeliuotą architektūrą išbandyti praktiškai t.y. kuo greičiau sukurti veikiančią sistemą. Paprastai siekiama turimas idėjas realizuoti sukuriant mažą prototipą t.y. sukurti nedidelę veikiančia programą, kuri realizuoja bazinius funkcionalumus (šis metodas taip pat naudojamas XP bei RUP metodologijose). Tai reikalinga rizikos valdymui, kadangi leidžia patikrinti ar sumanymai pasiteisina. Gali kilti

klausimas ar verta skirti laiką ir resursus sistemai, kuri bus nepilnai funkcionuojanti? Verta ir štai kodėl:

- Galimybė patikrinti kokios architektūrinės idėjos pasiteisino ir kiek gerai pasiteisino;
- Greitas kūrimas leidžia numatyti naujas problemas ir techninio realizavimo sunkumus;
- Lygiagrečiai atliekamas programavimas ir modeliavimas ženkliai padidina galimybę, jog sistema atitiks probleminės srities poreikius;
- Kuo greičiau veikianti sistema pateikiama vartotojui, tuo greičiau galima tikėtis grįžtamojo ryšio.
- Pelnoma klientų pagarba ir pasitikėjimas, kadangi jie nuo pačios projekto pradžios intensyviai įtraukiami į informacinės sistemos kūrimą bei greitai reaguojama į naujus pageidavimus (S. W. Ambler, 2006)

Paprastai yra priimta, jog Agile projektuotojai yra ir programuotojai. Tai reiškia, jog jie būna vienoje vietoje su visa komanda, taigi gali lengviau koordinuoti veiksmus bei pateikti siūlomus architektūrinius sprendimus.

Globalus požiūris į sistemą

Visų pirma tai reiškia, jog Agile modeliuotojai naudoja daug modelių, kurie suteikia galimybę pažinti sistemą įvairiais pjūviais ir pamatyti pilną vaizdą. Stengiamasi nesivadovauti vien žinomais modeliai ir notacijomis, vietoj to ieškoma būdų, kurie galėtų išreikšti specifines problemas bei poreikius. Tokia filosofija turi tiek teigiamų, tiek neigiamų aspektų pvz. teigimas - įgyjamas lankstumas, neigiamas – sunku susišnekėti, jeigu nėra apibrėžta modelio notacija ir sudarymo principai.

Sukurti veiklos architektūrą, kuri būtų patraukli vartotojui

Agile filosofija yra orientuota į vartotoją t.y. kuo geriau patenkinti jo poreikius. Šis požiūris yra suprantamas, kadangi tik patraukli ir naudinga sistema bus gerai įvertinta ir plačiai naudojama. Kaip to pasiekti – dirbti su užsakovu, tokiu būdu jis gali pats keisti sistemą ir pritaikyti prie konkrečių savo darbo poreikių. Deja, ne toks retas atvejis, kai užsakovas nori kuo mažiau prisidėti prie sistemos kūrimo, įsivaizduodamas, jog tai ne jo rūpestis, kadangi jis

moka pinigus ir tuo viskas pasibaigia. Galima sakyti, jog tokia situacija yra kritinė Agile metodologijai.

Apžvelgėme visus principinius Agile metodologijos aspektus skirtus veiklos modeliavimui, todėl kai ką galime jau dabar apibendrinti. Agile nepateikia konkrečių receptų kaip pasiekti tikslą, tačiau būtinas sąlygas galime išskirti:

1. Vartotojų bendradarbiavimas, modeliuojant veiklos architektūrą;
2. Vizija ir planas kaip ją pasiekti;
3. Modeliai ir dokumentacija apibūdinanti veiklos architektūrą.

Akivaizdu, jog konkretumo čia nerasime, tačiau, todėl tai ir yra Agile. Iš principo norint, kad visa tai veiktų reikia keisti savo mastymą t.y. atsisakyti nuomonės, jog tik detalus ir geras planavimas yra sėkmės pagrindas. Tiesiog neįmanoma to padaryti, kadangi perdaug informacijos, kurią reikia įvertinti, perdaug neprognozuojamų įvykių, kuriuos reikia numatyti. Nors dažnai projektuotojai stengiasi tai padaryti, kas gali būti pražūtinga visai projekto sėkmei.

Daug nefilosofuodami galime išskirti ir pagrindinius Agile metodologijos trūkumai, nors manau jie suvokiami intuityviai šiuos trūkumus (kurie manau yra intuityviai suvokiami)

1. Agile procesai labai priklauso nuo žmonių atsakomybės (žmoniškasis faktorius);
2. Sudėtinga visur pritaikyti KISS principą;
3. Sudėtinga planuoti grafiką ir tvarkaraštį;
4. Sudėtinga valdyti ir koordinuoti veiklas;
5. Sudėtinga sudarinėti ir palaikyti modelius ir dokumentaciją, kuri dažnai keičiasi (S. W. Ambler, 2006)

1.2 Informacinė sistema

Informacinė sistema – tai tam tikra tvarka sutvarkyta dokumentų visumos ir informacinių technologijų, realizuojančių informacinius procesus, sistema. Informacinė sistema – duomenų masyvų visuma, o taip pat techninių, programinių ir metodinių priemonių visuma, skirta informacijos kaupimui, atnaujinimui, korekcijai, panaudojimui ir pašalinimui.

Dažniausiai sutinkamas toks informacinės sistemos apibrėžimas: automatizuota informacinė sistema – aparatinio, programinio ir aparatinio-programinio aprūpinimo visuma,

sukonfigūruota duomenų ar informacijos rinkimui, sukūrimui, perdavimui, platinimui, apdorojimui, saugojimui bei jų valdymui.

Informacinė sistema sudaroma kompiuterinės sistemos pagrindu, ją formuoja 5 komponentai:

- kompiuterinė sistema;
- žmonės;
- procedūros;
- duomenys ir informacija;
- ryšio priemonės (kai kompiuteriai dirba tinkle).

Žmonės - tai svarbiausioji informacinės sistemos dalis:

- kompiuterių profesionalai kuria techninę ir programinę kompiuterių įrangą;
- profesionalūs kompiuterių operatoriai prižiūri ir valdo kompiuterinių sistemų veiklą;
- profesionalūs kompiuterių tarnautojai ir vartotojai kiekvieną dieną įveda didžiulius kiekius duomenų, kurie vėliau bus apdorojami ir paverčiami informacija;
- vartotojai kuria savo specializuotą programinę įrangą;
- vartotojai analizuoja informaciją, gautą kompiuteriu, kad galėtų priimti efektyvius verslo sprendimus;
- vartotojai ir kompiuterių profesionalai priima sprendimus, naudoja ir valdo kompiuterines sistemas, kurios gali turėti įtaką mūsų saugumui ir sėkmingam gyvenimui.

Informacinėje sistemoje yra vykdomos 4 pagrindinės procedūros:

- duomenų įvedimas;
- duomenų apdorojimas;
- informacijos išvedimas;
- informacijos saugojimas. (T. Bilevičienė, 2006)

1.3 Sistemos kūrimo metodų analizė

Programinė įranga Lietuvoje dažniausiai kuriama naudojantis metodais, kurie remiasi dokumentacijos kūrimu. Tai yra reikalavimai yra dokumentuojami, bet nesvarbu kaip išsamiai tai būtų daroma, kūrimo procese beveik visada atsiranda vienokių ar kitokių pasikeitimų. Kiekvienas pakeitimas ir ypač pakeitimo laikas turi didelę įtaką projekto kainai. Kadangi kuo

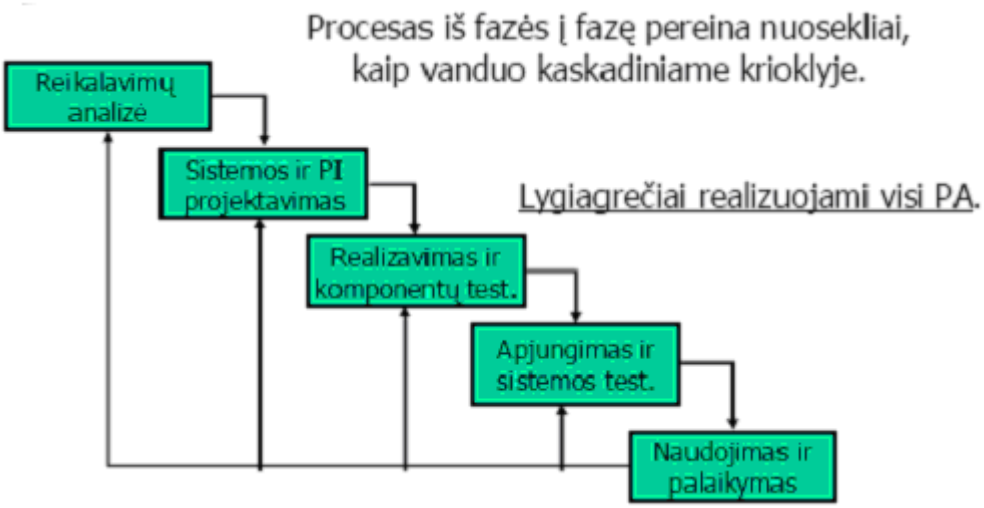
vėliau bus daromi pakeitimai, tuo labiau tai įtakos darbų tvarkaraščius bei projekto apimtį, kas reiškia didesnę kainą.

Yra daug sistemos kūrimo metodikų. Sistemos pobūdis turi įtakos kokį modelį naudoti, nes skirtingi modeliai nevienodai tinka įvairaus tipo sistemoms kurti. Tinkamos metodikos pasirinkimui įtakos turi: organizacinė aplinka ir taikomųjų uždavinių rūšis.

1.3.1 Krioklio modelis

Krioklio modelis – tai klasikinis modelis, kai programa yra kuriama nepertraukiamai, etapas po etapo. Jį tikslinga taikyti, kai:

- vartotojo reikalavimai yra aiškūs,
- uždavinys yra tradicinis,
- aiškus sprendimo algoritmas,
- uždavinys nėra sudėtingas.



Pav. 2 Krioklio (kaskadinis) modelis

Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

1 lentelė

Krioklio modelio analizė

Trūkumai	Privalumai
1. Nelankstus projekto skirstymas į stadijas. 2. Yra sunkumų, kai reikia taisyti klaidas arba reaguoti į vartotojo reikalavimų pasikeitimą. 3. Vykstant procesui, sunku daryti pakeitimus,	1. Jei nėra daug grįžimų, nedidelės darbo sąnaudos. 2. Procesas aiškus, lengvai koordinuojamas. 3. Galimybė suskirstyti kolektyvą į skyrius ir

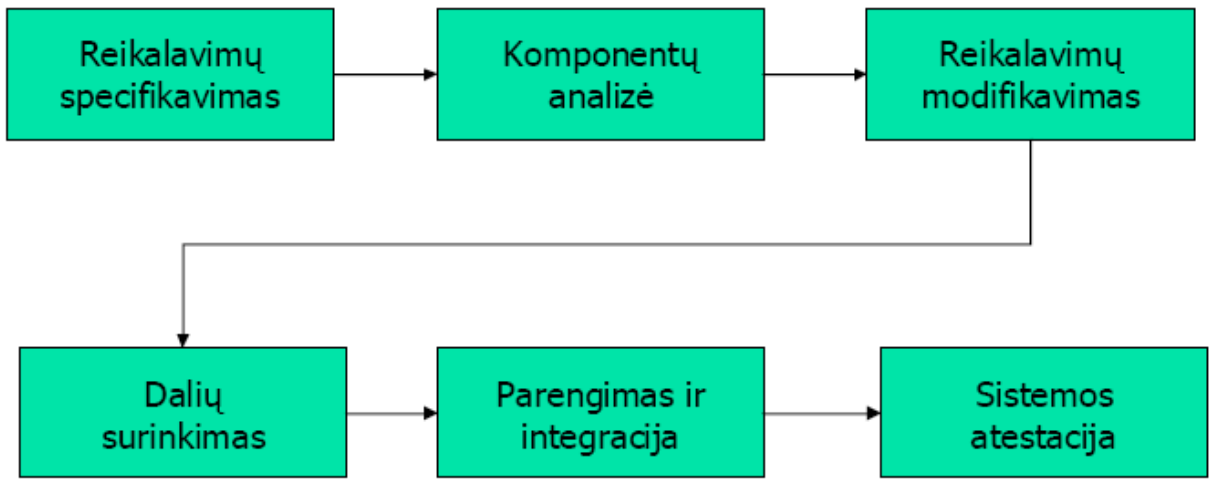
paprastai jie atliekami po paskutinės fazės, grįžtant į kurią nors fazę. 4. Negalima vartotojams pateikti dalinio programos varianto. 5. Eksploatacijos pradžia galima tik realizavus visus panaudojimo atvejus, kas atitinka ilgiausią kelią tinkliniame grafike.	atlikti valdymo kontrolę.

Šaltinis: sudaryta autoriaus pagal Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

Krioklio modelis taikomas tik tokiais atvejais, kai reikalavimai ir jų įgyvendinimas yra labai gerai suprasti. Kūrimas prasideda nuo koncepcijos, po to pereinama prie projektavimo, realizavimo, testavimo, įdiegimo, gedimų pašalinimo ir baigiasi eksploatacija ir palaikymu. Kiekviena kūrimo fazė vyksta pagal griežtą tvarką, jokių persidengimų ar iteracijų. (A. Targamadzė, 2005)

1.3.2 Pakartotinio panaudojimo modelis

Pasaulyje yra atrasta ir suskurta daug naudingų dalykų bei sistemų, tad ne visada yra tikslinga atradinėti kažką naujo, kai tai gali būti ypač sudėtinga. Tikslingiau būtų panaudoti tai kas jau išrasta ir geriausiai gali patenkinti tavo poreikius. Pakartotinio panaudojimo modelio esmė ir yra ta, kad uždaviniams spręsti yra panaudojamos jau esamos sistemos, posistemės ar programos. Arba į vieningą sistemą integruojamos skirtingų sistemų sukurtos dalys, kad veiktų kaip nauja sistema, iškilusiems uždaviniams spręsti. (A. Targamadzė, 2005)



Pav. 3. Pakartotinio panaudojimo modelis

Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

2 lentelė

Pakartotinio panaudojimo modelio analizė

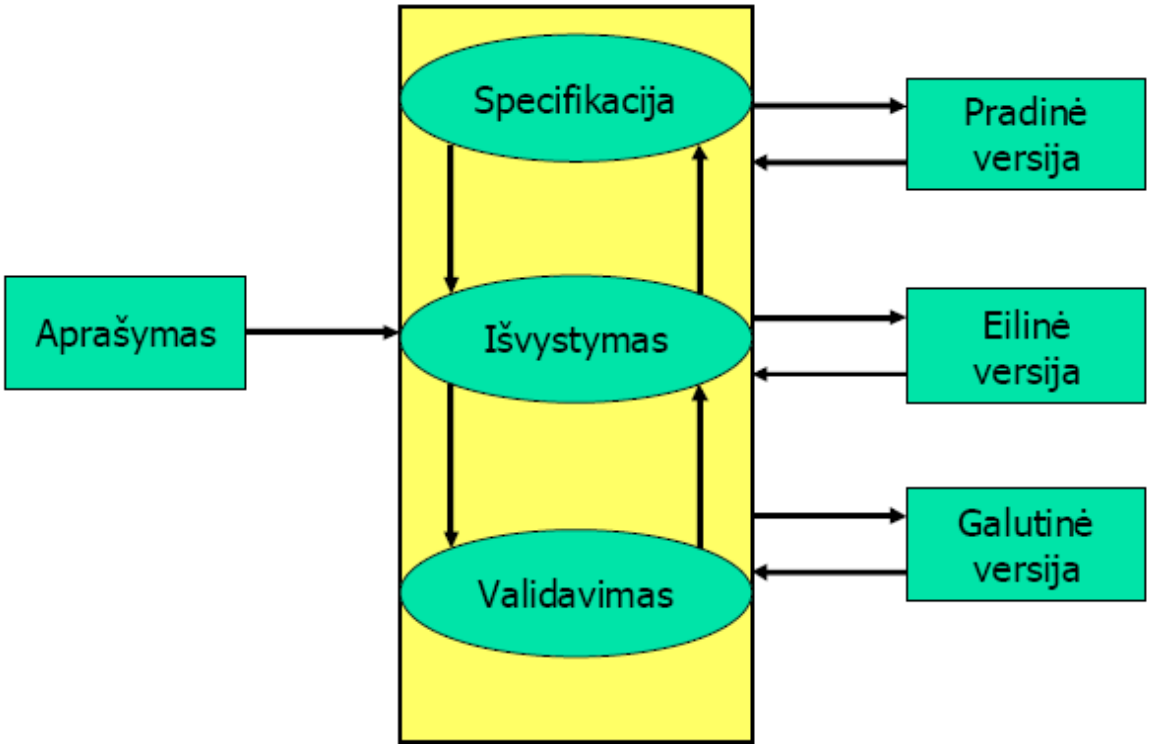
Trūkumai	Privalumai
1. Reikia gerai orientuotis produktų rinkoje. 2. Reikalingi specialistai, gebantys aukščiausiu lygiu įsisavinti svetimus produktus, dažnai neturint išeities kodus. 3. Sudėtinga posistemių sąveika, ne visuomet efektyvūs interfeisai tarp posistemių, skirtingi darbo su posistemėmis stiliai.	1. Išnaudojamas kitų sukurtas geriausias produktas. Gali būti, kad tie kiti geriau išmano siauresnę taikomąją sritį. 2. Mažesnis PS kūrimo darbo imlumas, ji sukuriama greičiau ir pigiau. 3. Taip lengviau išlaikyti progreso smaigalyje.

Šaltinis: sudaryta autoriaus pagal Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

1.3.3 Evoliucinis modelis

Evoliuciniame modelyje projektas vystosi nuosekliai, pradedant panaudojimo atvejo poaibio realizavimu ir nuosekliai prijungiant naujus panaudojimo atvejus arba jų grupes. Modelio esmė – darbas su užsakovu nuo pradinės specifikacijos iki galutinės sistemos versijos. PS reikalavimų supratimas auga realizavimo eigoje. Tikslinga pradėti nuo aiškių ir svarbių panaudojimo atvejų. Labai svarbus yra išdirbėjo bendravimas su vartotoju, gera psichologinė aplinka. (A. Targamadzė, 2005)

Pagrindinė evoliucinio modelio idėja – veikiančių kuriamos sistemų prototipų naudojimas jos dinaminėms savybėms modeliuoti.



Pav. 4. Evoliucinis modelis

Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

3 lentelė

Evoliucinio modelio analizė

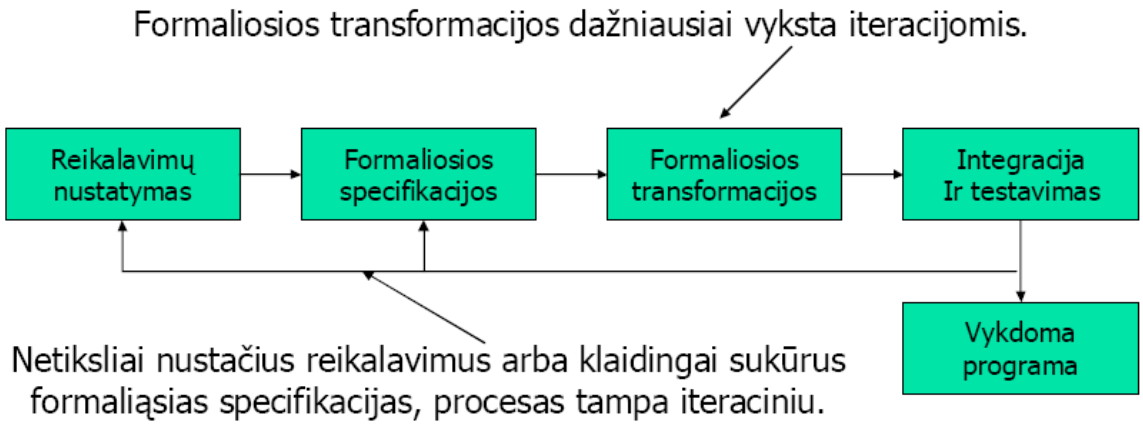
Trūkumai	Privalumai
1. "Suveltas procesas". 2. PS gaunasi blogai struktūrizuota. 3. Ne visada galima įkalbėti vartotoją dirbti su nepilnai suprojektuota sistema.	1. Atitinka mūsų "suveltą" gyvenimą. 2. Nereikalauja aukštos kvalifikacijos sisteminių analitikų – brangiausių specialistų. 3. Efektyvu mažoms sistemoms, nors taip kuriamos ir didelės sistemos, kurios yra per sudėtingos, kad būtų galima jas aprėpti. 4. Šis būdas realizuojasi natūraliai, einant per sistemos gyvavimo etapus.

Šaltinis: sudaryta autoriaus pagal Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

Šis modelis gali būti veiksmingai naudojamas tik turint prototipams konstruoti skirtas instrumentines priemones. Kadangi prototipus modifikuoti tenka daug kartų, daryti tai rankiniu būdu yra per brangu ir reikia labai daug laiko (A. Targamadzė, 2005)

1.3.4 Formalusis modelis

Sistemos specifikavimas šiame modelyje traktuojamas kaip atitinkamos probleminės srities teorijos formulavimas. Teorija yra formuluojama etapais, kol pereinama prie abstrakčių formaliosios logikos sąvokų ir formuluojama formali dalykinės srities teorija. Ši teorija kartu yra ir formali kuriamosios sistemos reikalavimų specifikacija. Specifikacija nusakoma formalia dalykinės srities teorija. Kuriamoji sistema yra vienas iš tos teorijos modelių, todėl sistemos korektiškumas yra garantuotas automatiškai (A. Targamadzė, 2005)



Pav. 5. Formalusis sistemų kūrimas

Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

4 lentelė

Formaliojo modelio analizė

Trūkumai	Privalumai
1. Procesas labai formalus ir sudėtingas. 2. Sunku formaliai aprašyti sistemas. 3. Reikia aukštos kvalifikacijos specialistų. 4. Nepavyksta visko gauti vien formalių transformavimų kelių, tenka įdėti ir rankinio darbo, ypač – projektuojant "žmogus-sistema" interfeisą	1. Iš pat pradžių būtina gerai suprasti sistemos reikalavimus ir jos veikimo ypatybes. 2. Vėlesnis procesas nėra reikalaujantis daug darbo, jis greitas. 3. Gana patogi reinžinerija. 4. Tinka nedidelėms sistemoms bei sistemoms, keliančioms aukštus reikalavimus patikimumui.

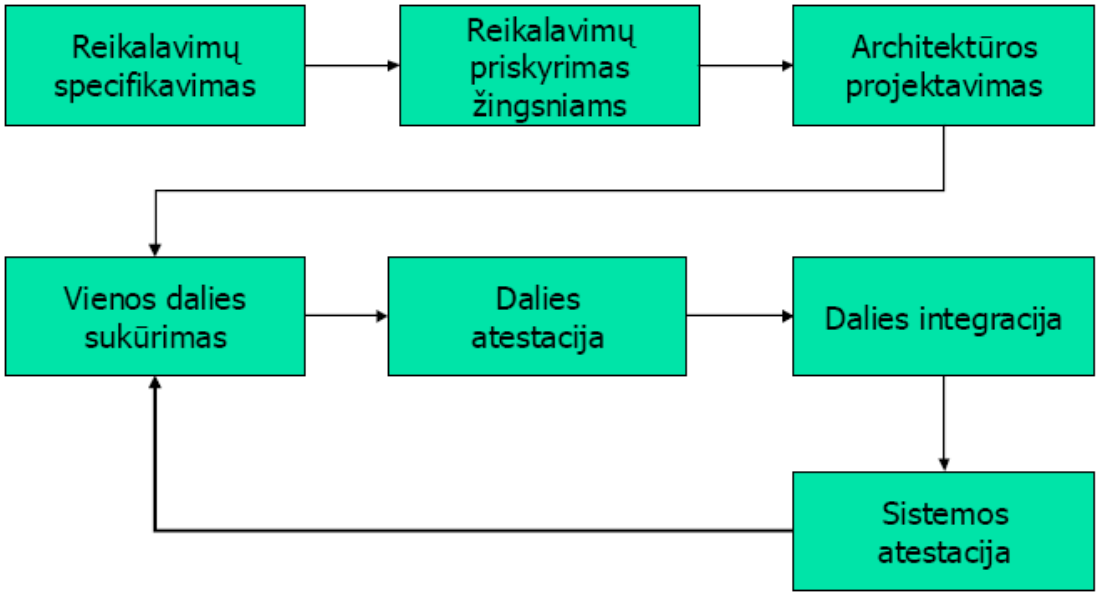
Šaltinis: sudaryta autoriaus pagal Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

Sistema yra aprašoma matematiškai ir formalių metodų pagalba generuojamas galutinis jos variantas (vykdoma programa). Tokiu būdu galima sugeneruoti tik vykdomą programą, tačiau lieka parengti dokumentaciją, "help" failus ir t.t.

1.3.5 Žingsninis kūrimas

Žingsninis sistemos kūrimas, tai kai sistema yra kuriama žingsniais ir palaipsniui. Vartotojų reikalavimams yra suteikiami prioritetai ir pradžioje stengiamasi realizuoti

svarbiausius panaudojimo atvejus. Tai yra kiekvieną kartą yra realizuojama tik dalis sistemos. Procesui prasidėjus, reikalavimų stengiamasi nekeisti, nors tai ne visada pavyksta. (A. Targamadzė, 2005)



Pav. 6. Žingsninis kūrimas

Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

5 lentelė

Žingsninio kūrimo analizė

Trūkumai	Privalumai
1. Problema – atskirų dalių integravimas 2. Gali tekti keisti architektūrą, kai bus realizuota didesnė dalis sistemos. 3. Sistema gali gautis neoptimali.	1. Sistema gali būti pradėta naudoti dar nerealizavus pilno funkcionalumo. 2. Žingsniniame realizavime įgaunamas patyrimas, kuris gali būti panaudotas kituose žingsniuose. 3. Procesą galima koreguoti, mažėja nesėkmės rizika. 4. Uždavinio dekompozicija lengvina realizacija.

Šaltinis: sudaryta autoriaus pagal Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

1.3.6 Spiralinis kūrimas

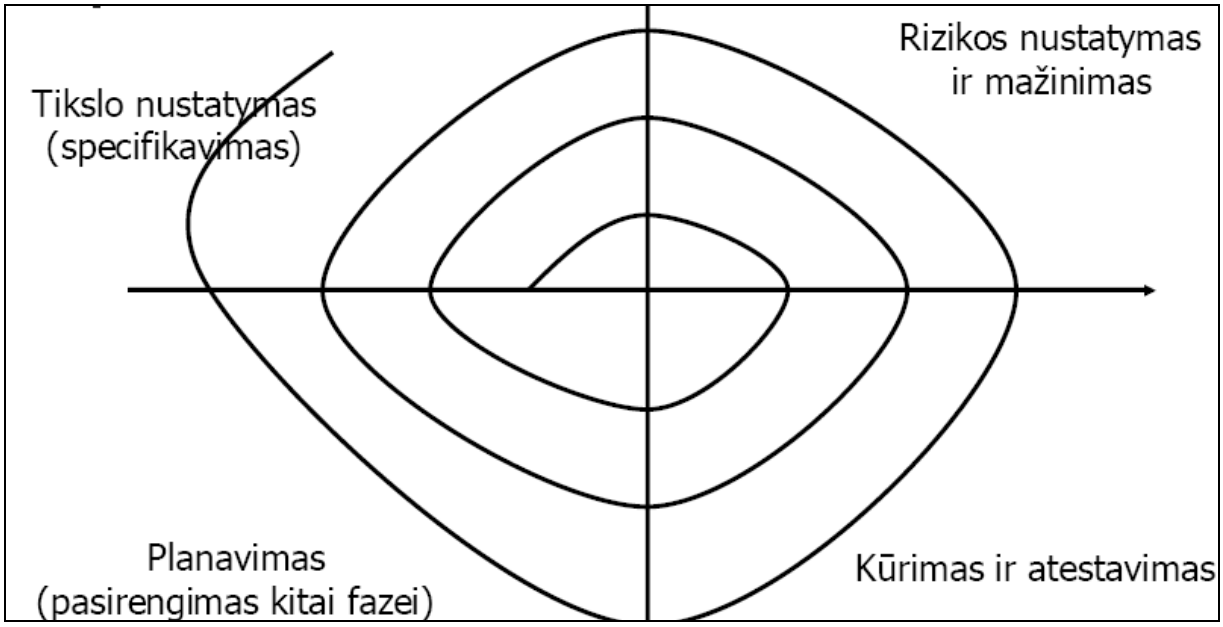
Spiralinis gyvavimo ciklo modelis – tai klasikinio (“krioklio”) modelio modifikacija pildant jį rizikos analize ir prototipų kūrimu (A. Targamadzė, 2005)

Procesas vystosi kaip spiralė, o ne kaip veiksmų seka su grįžimais. Kas buvo padaryta, nėra perdaroma, tik patobulinama. Nėra fiksuotų fazių, spiralėje tikslai pasirenkami pagal

poreikius. Specifikavimas vykdomas ne visuomet. Rizika didelė, ji specialiai vertinama ir stengiamasi ją mažinti.

Spiralinis kūrimas susideda iš tokių etapų:

1. Tikslo, apribojimų nustatymas (specifikavimas);
2. Rizikos nustatymas ir mažinimas;
3. Kūrimas ir atestavimas ;
4. Kitos stadijos planavimas.



Pav. 7. Spiralinis kūrimas

Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

6 lentelė

Spiralinio kūrimo analizė

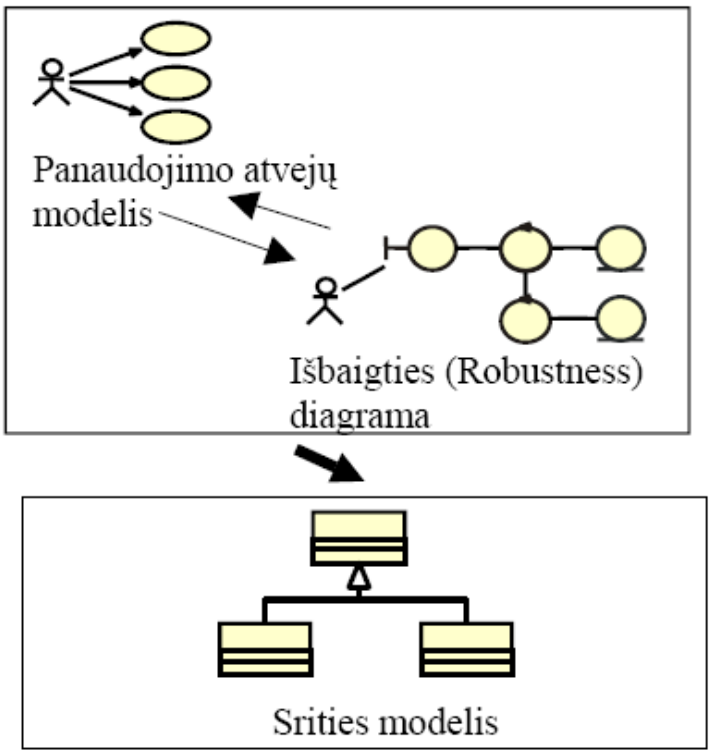
Trūkumai	Privalumai
1. Reikalauja didelio rizikos įvertinimo. Rizika specialiai vertinama ir stengiamasi ją sumažinti. 2. Padarytas funkcionalumas nėra perdaromas, o tik patobulinamas. 3. Jei rizikos faktorių nerandama, tai tikimybė atsirasti problemos yra didelė.	1. Galimybė užsakovui numatyti būtinus pakeitimus, panaudoti naujas technologijas. 2. Galimybė užsakovui ir kūrėjui kiekvienoje kūrimo stadijoje nustatyti ir įvertinti riziką. 3. Rizikos įvertinimas ir aptarimas kiekvienoje sistemos kūrimo stadijoje sumažina problemų skaičių;

Šaltinis: sudaryta autoriaus pagal Šaltinis: Targamadzė A. (2005) http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

Spiralinis gyvavimo ciklo modelis tinka dideliems ir sudėtingiems projektams. Kadangi daug dėmesio skiriama rizikos įvertinimui, šis modelis netinka mažiems ir nesudėtingiems projektams.

1.4 ICONIX metodas

ICONIX programinės įrangos kūrimo metodas, pagrįstas minimaliu UML diagramų kiekiu ir efektyvia metodika, kurios dėka kūrimo procesas „nuo panaudojimo atvejų iki kodo" yra greitas ir efektyvus. ICONIX labai daug dėmesio skiria reikalavimų apibrėžimui. Šio proceso etapai: probleminės srities aprašymas, panaudojimo atvejų aprašymas, išbaigtumo (angl. Robustness) analizė ir sekos diagramų sudarymas.



Pav. 8 ICONIX reikalavimų specifikacijos struktūra

Šaltinis. ICONIX Software Engineering (2008) http://www.iconixsw.com/

Pagrindinis *ICONIX* proceso rezultatas – sekos diagramos, kurios kartu su statine klasių diagrama toliau panaudojamos kodo rašymui. Siekiant ICONIX procesą padaryti pilnesniu, o projektavimą išsamesniu, šis procesas papildytas būsenų diagramomis. Be to, yra galimybė automatizuotam būsenų diagramos generavimui arba būsenų diagramų suderinimui su sekų diagramomis. Pirmajame etape kiekviena sekos diagrama (scenarijus) aprašoma įvykių seka, o antrajame generuojama būsenų diagrama.

ICONIX labai daug dėmesio skiria reikalavimų apibrėžimui, tačiau suderinamumas tarp panaudojimo atvejų, kuriuose užregistruoti reikalavimai ir kitų modelių, ypač klasių diagramų, nėra užtikrintas.

ICONIX programinės įrangos kūrimo metodui būdingos sekų ir išbaigties diagramos kiekvienam panaudojimo atvejui padeda užtikrinti reikalavimų specifikacijos pilnumą, išbaigtumą. Tačiau toks reikalavimų apibrėžimas nėra griežtai atskirtas nuo projekto modelio (reikalavimų specifikacijoje įvedami tik vėliau projekto modelyje reikalingi valdikliai), o tai nesuderinama su modeliu valdomos architektūros idėjomis. (Software Engineering, 2008)

7 lentelė

ICONIX metodo analizė

Trūkumai	Privalumai
1. Nėra užtikrintas suderinamumas tarp panaudojimo atvejų ir kitų modelių, ypač klasių diagramų.	1. Sekų diagramos ir išbaigties diagramos sudaromos kiekvienam panaudojimo atvejui, bet atsisakoma valdiklių (t.y., priešlaikinių projektinių sprendimų). 2. Galima automatiškai generuoti būsenų diagramą

Šaltinis. Sudaryta autoriaus pagal ICONIX Software Engineering (2008) http://www.iconixsw.com/

1.5 Agile metodai

Agile metodai akcentuoja tiesioginę komunikaciją. Reikalavimų nustatymas yra pagrįstas prototipų kūrimų. Atsiradus naujiems pakeitimams, turi būti greitai į juos reaguojama.

Agile metodai taikomi, kai reikalavimai kinta ir programų kūrimas yra sunkiai valdomas .

8 lentelė

Agile metodai

Trūkumai	Privalumai
1. Sudėtinga įtraukti vartotojus į kūrimo procesą ir aktyviai jame dalyvauti. 2. Sunku prognozuoti produkto kainą ir galutinio produkto pristatymo terminus. 3. Pripratus prie tradicinių metodų, sunku persiorientuoti. 4. Testuotojai reikalingi viso projekto metu, o	1. Orientacija į principus, ne į procesą. 2. Tiesioginė komunikacija tarp kūrėjų ir užsakovų. 3. Operatyvus reagavimas į reikalavimų pasikeitimus. 4. Pakeitimai galimi net paskutiniu momentu. 5. Mažos komandos.

tai padidina kaštus. 5. informacija gali būti įvairiai interpretuojama ir gali kisti ją perduodant.	6. Lankstumas.

Šaltinis: sudaryta autoriaus pagal Butkienė R. ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

Kadangi užsakovai dažnai tiksliai nežino, ko jiems reikia, kokias funkcijas naujoji sistema turės atlikti, todėl dažnai keičia reikalavimus. Tam reikia, kad būtų greitas reagavimas į reikalavimų pasikeitimus, t.y. lankstumas kūrimo procese. Todėl Agile metodai būtų tinkamiausi nedaug funkcijų turinčių sistemų kūrimui.

Nors Agile siūlomi metodai gali būti labai efektyvūs, tačiau jie reikalauja visiškai kitokio programinės įrangos kūrimo proceso, jo planavimo, valdymo ir finansavimo. Visų pirma Agile metodai pagrįsti tuo, kad programinės įrangos kūrimas yra apmokamas už trumpo periodo iteracijas pagal dirbtų valandų skaičių. Daugelyje projektų toks požiūris nėra priimtinas, nes iš anksto nustatomas viso projekto biudžetas, pagal numatytus reikalavimus skelbiami konkursai programinei įrangai kurti. (R. Butkienė, 2007)

Taip pat sunku įtraukti vartotoją, kai jam pateikiamos dažnai atnaujinamos programinės įrangos versijos. Tai ypač keblu, kai kalbama apie kritines arba svarbias verslo valdymo sistemas (bankinės, gamyklų valdymo, apskaitos sistemos), kurių galutiniai vartotojai neturėtų dirbti su tarpinėmis neišbaigtomis programinės įrangos versijomis. Be to, praktikoje dažniausiai yra labai sunku, o kartais ir neįmanoma, įtraukti vartotojus ir verslo atstovus į kasdienius programinės įrangos kūrimo procesus.

Bendraujant gyvai galima išsiaiškinti daug detalių, tačiau nedokumentuota informacija paprastai yra įvairiai interpretuojama ir kinta ją perduodant. Greitai reaguoti į pakeitimus ir atsisakyti formalaus reikalavimų pakeitimų valdymo galima tik tuomet, kai pakeitimai yra nežymūs. Tačiau stambesni pakeitimai turi būti apsvarstomi, įvertinami, jiems turi būti suteikiami prioritetai. Dideli pakeitimai lemia ne tik vykdomo projekto apimtį, darbų tvarkaraščius, bet ir produkto išleidimo į rinką laiką.

Dėl minėtų priežasčių Agile metodai labiau tinkami naujiems produktams kurti bei vykdyti vidinius projektus, kur galutinius vartotojus gali pakeisti kompanijos darbuotojai. Šiuo atveju prioritetų suteikimas ir pakeitimų valdymas tampa mažiau formalus, nes yra lengviau valdyti projekto apimtį, tvarkaraščius ir biudžetą.

Projektų, kuriuose naudojami *Agile* metodai sėkmės faktoriai – vartotojo įtraukimas, efektyvus reikalavimų valdymas, komunikacija. Pagrindinės projektų nesėkmių priežastys – reikalavimų valdymo problemos. Todėl reikia pakeisti Agile metodus prijungiant ICONIX metodą, kuris turi keturias UML diagramas ir efektyvią metodiką, kurios dėka kūrimo procesas „nuo panaudojimo atvejų iki kodo" yra greitas ir efektyvus.

1.6 Taikomųjų programų kūrimas panaudojant Agile

Viena svarbiausių pasaulyje pramonės šakų šiuo metu yra programinė įranga, ji apima visas gyvenimo sritis: nuo vaikų žaislų iki kompiuterių, nuo virtuvės kombainų iki automobilių ir kitų plačiai naudojamų galutinio vartojimo prekių. Taip pat programinė įranga plačiai naudojama medicinoje bei kuriant jos techniką, ryšių ir transporto sistemas, atomines elektrines, lėktuvus, raketas, ir klausos aparatus. Programinių produktų gamybos apimtys auga eksponentiškai. Galima teigti, kad tai svarbiausia pramonės šaka pasaulio ekonomikai, bet šioje pramonėje taip pat kyla problemų. Tokios problemos kaip įrangos patikimumo, savikainos, našumo ir kitos problemos kyla, nes vis dar yra plačiai naudojami amatiniai darbo metodai.

Šiais laikais visi reikalauja, kad programinė įranga būtų ne tik kuo greičiau sukurta ir pristatyta, bet ir turi būti didesnio funkcionalumo bei aukštos kokybės. Dar yra reikalaujama, kad kūrimo metu būtų greitai įvykdomi pakeitimai. Kad procesas būtų vikrus (Agile), jis turi būti pakankamai lankstus, kad sklandžiai ir laiku prisitaikytų prie reikalavimų pasikeitimo.

Dabartinių informacinių sistemų kūrimą tampa vis sunkiau ir sunkiau suvaldyti. Daugumos informacinių sistemų įrangų reikalavimai keičiasi. Yra apimamas planavimas, architektūra, sistemos dizainas, kokybę užtikrinantys pasikeitimai, nuolatinis atnaujinimas, sistemos priežiūra kai reikalavimai kinta.

Per paskutinius trisdešimt metų buvo pristatyta daug programinės įrangos procesų ir metodologijų, kurie padeda susitvarkyti su problemomis, susijusiomis su programinės įrangos inžineriniais projektais. (A. McDonald, 2005). Informacinių sistemų kūrimo praktiniai niuansai:

- Greitai daryti iteracijas. Tai padeda įgyvendinimo procesui.
- Naudoti, kol dar kuriate. Visi komandos nariai kūrimo metu turi naudoti programinę įrangą. Jie yra geriausi prototipo testuotojai.

- Daryti greitas iteracijas. Leisti vartotojams naudoti programinę įrangą, nors jinai dar vis yra kuriama. Taip galima pamatyti realius elgesio pavyzdžius ir pataisyti kritines vietas (A. McDonald, 2005)
- Yra daug metodologijų, skirtų kurti sistemoms, bet informacinėms sistemoms labiausiai tinka *Agile* metodai, nes *Agile* procesai yra paremtas:
- pakeitimų galimybe;
- žmonėmis (orientuoti į žmones);
- mažomis kūrėjų komandomis;
- trumpais kūrimo ciklais

Toliau esanti citata iš Manifesto programinės įrangos kūrimui pateikia tikslią santrauką:

„Mes atskleidžiame geresnius programinės įrangos kūrimo būdus tai darydami ir padėdami kitiems tai padaryti. Dirbdami priėjome prie išvadų:

- Individualumas ir sąveika per procesus ir įrankius.
- Veikianti programinė įranga per išsamią dokumentaciją.
- Užsakovų bendradarbiavimas pagal kontrakto sąlygas.
- Atsakas į pasikeitimus vykdant planą. (A. Ginige, 2004)

Apskritai inžineriniuose projektuose pagrindinis faktorius, nulemiantis sėkmę ar nesėkmę informacinių sistemų kūrime yra ir kūrėjai, ir organizacijos. Todėl *Agile* pasirinktas kelias su orientavimusi į žmones bei individualumu į sąveika per procesus ir įrankius stipriai priartina metodą prie informacinių sistemų kūrimo.

1.7 Agile metodologija

Agile metodologija yra metodų visuma, skirta minimizuotų pasikeitimų kainą, ypatingai ten, kur svarbūs faktai paaiškėja tik baigiantis projektui, arba kai reikia prisitaikyti prie svarbių nekontroliuojamų veiksnių (J. Bach, 2006).

2001 m. paskelbtas „Agile Software Development“ manifestas. Jo metu buvo suformuota dvylika *Agile* programinės įrangos kūrimo principų:

- Didžiausias prioritetas yra patenkinti užsakovo reikalavimus per ankstųjį ir vėlyvesnius programinės įrangos pristatymus.

- Priimtini besikeičiantys reikalavimai netgi vėlyvajam kūrime. *Agile* procesas kompetentingiems užsakovo norams suteikia pirmenybę.
- Dažnai pristatyti veikiančią programinę įrangą, nuo dviejų savaičių iki dviejų mėnesių, pirmenybė teikiant trumpesniems laiko intervalams.
- Verslo žmonės ir kūrėjai turi dirbti kartu vykstant projektui ištisai kiekvieną dieną.
- Projektas kuriamas motyvuojant individualiai. Reikia suteikti aplinką ir palaikymą, kurio jiems reikia, ir pasitikėti, kad darbas bus įvykdytas.
- Veiksmingiausias ir efektyviausias informacijos perteikimo metodas kūrimo komandai ir tarp komandos narių yra pokalbis akis į akį.
- Veikianti programinė įranga yra pagrindinis progreso matas.
- *Agile* metodai skatina nepertraukiamą kūrimą. Finansuotojai, kūrėjai ir vartotojai privalo kaskart stebėti nuolatinius žingsnius.
- Nepertraukiamas dėmesys techniniam meistriškumui ir geram dizainui pagerina judrumą.
- Paprastumas – menas maksimaliai išnaudoti nepadarytą darbą – yra svarbus.
- Geriausia architektūra, reikalavimai ir dizainas atsiranda iš autonominių komandų.
- Reguliariuose intervaluose komandos apgalvoja, kaip padaryti dar efektyviau, tada darnumas ir prisitaikymas funkcionuoja atitinkamai. (Agile Alliance, 2001)

  Agile metodai:

  - Extreme Programming (XP)
  - Scrum
  - Agile Modeling
  - Adaptive Software Development (ASD)
  - Crystal Clear and Other Crystal Methodologies
  - Dynamic Systems Development Method (DSDM)
  - Feature Driven Development (FDD)
  - Lean Software Development
  - Agile Unified Process (AUP)
  - Agile Database Techniques (AD)
  - Test-Driven Design (TDD)
  - XBreed

Agile metodai vertinami kaip adaptyvūs, o ne kaip prognozuojantys, orientuoti į žmones, o ne į procesą (S. W. Ambler, 2006)

Metodas yra *Agile*, jei sistemos kūrimas yra :

- laipsniškas (nedideli programinės įrangos leidiniai, trumpi kūrimo ciklai);
- kooperatyvus (užsakovas ir kūrėjas pastoviai dirba kartu ir glaudžiai bendrauja);
- nesudėtingas (metodą lengva išmokti, keisti; gerai dokumentuotas);
- adaptyvus (pakeitimai galimi net paskutiniu momentu) (S. W. Ambler, 2006)

1.8 Agile metodų analizė

Šitame skyriuje bus analizuojami Agile metodai.

1.8.1 Extreme Programming (XP) metodas

Ekstremalus programavimas – tai (gerai apgalvotas) požiūris į programinės įrangos kūrimą.. XP pradininkas – Kent Back. XP yra metodika, kuri yra priskiriama prie judriųjų (agile) metodikų.

XP metodika yra sėkminga, nes akcentuoja:

- kliento poreikius ir norus – tiekime PĮ klientui, tokią kokios jam reikia ir kada reikia.
- kolektyvinį darbą – vadybininkai, klientai, programuotojai – visi yra dalis kolektyvo, kuris tiekia kokybišką programinę įrangą.

XP pagerina programinės įrangos kūrimo projektą keturiais esminiais aspektais:

- Komunikavimas - XP programuotojai bendrauja su savo klientais ir kitais programuotojais.
- Paprastumas – sistemos dizainas turi būti kiek įmanoma paprastas.
- Atsiliepimai – pastoviai gaunami atsiliepimai apie kuriamą programinę įrangą. Sistema pristatoma klientui, kiek įmanoma anksčiau, ji ir toliau kuriama (vystoma) bei atliekami siūlomi pakeitimai.
- Drąsa - turint tokį pagrindą, XP programuotojai gali drąsiai reaguoti į besikeičiančius reikalavimus bei technologijas.

1.8.2 Kada naudojamas XP

XP programavimas atsirado kaip atsakas į problemiškas sritis, kuriose reikalavimai dažnai kinta. Klientai gali pradžioje gerai nežinoti, ką jų sistema turi daryti. Galima situacija,

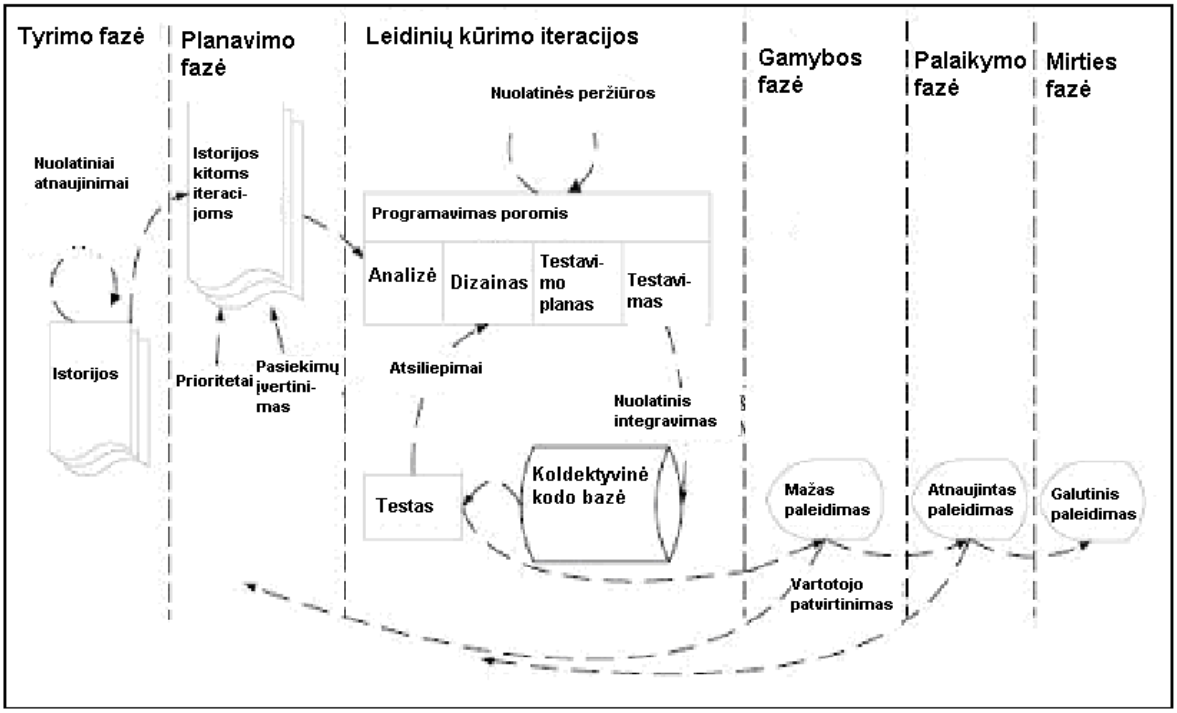
kai reikia sukurti sistemą, kurios funkcionalumas kinta kas kelis mėnesius. T.y. kai kuriose srityse dinamiškai besikeičiantys reikalavimai yra vienintelis pastovus dalykas. Tada XP yra sėkmingas, o kitos metodikos - ne.

Jei klientui reikia naujos sistemos tam tikrai datai, tai rizika yra didelė. Jei ši sistema yra naujas dalykas (anksčiau nieko panašaus nedarė) programinės įrangos kūrimo grupei, tai rizika yra dar didesnė. Jei ši sistema yra naujas dalykas visai programinės įrangos industrijai, tai rizika dar labiau padidėja. XP sumažina riziką ir padidina sėkmės tikimybę.

XP skirtas mažoms 2 - 10 programuotojų grupėms . XP yra sudėtinga pritaikyti kur dirba daug darbuotojų. Pastebėta, kad projektuose, kuriuose reikalavimai dinaminiai ar yra didelė rizika, mažam XP programuotojų kolektyvui sekasi labiau nei dideliam kolektyvui. Darbui su XP užtenka ir paprastų programuotojų – nereikia labai aukštos kvalifikacijos specialistų.

Klausinėjimas, derėjimasis dėl apimties ir tvarkaraščių, funkcinių testų kūrimas reikalauja daugiau nei tik programuotojų dalyvavimo kuriant programinę įrangą. Taip pat XP kolektyvo darbas yra pagrįstas darbu „alkūnė alkūnėn“, todėl kolektyvo sudėtį turėtų sudaryti įvairių žmonių kolektyvas. Tai yra programuotojai, vadybininkai, klientai.

XP projektų programuotojų produktyvumas geresnis palyginus su kitais projektais(ne XP), vykstančiais toje pačioje aplinkoje. Bet tai nėra XP metodikos tikslas. Tikrasis tikslas yra pristatyti programinę įrangą, kurios reikia ir tada, kada jos reikia. [15]



Pav. 9. Extreme Programming (XP) metodo schema

Šaltinis: Butkienė R. (2007) ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

1.8.3 XP taisyklės ir praktikos

- ✓ Planavimas:

Užrašomi vartotojų pasakojimai. Jie naudojami darbų/laidų planavimui, taip pat vietoj reikalavimų specifikacijos. Vartotojų pasakojimus užrašo užsakovas - tai yra tai, ką turi daryti sistema (pvz., galimybė pasižiūrėti metų ataskaitą.). Apie 80 (+-20) vartotojų pasakojimų yra idealus skaičius sukurti laidos planui. Vartotojo pasakojimas nusako sistemos išorines funkcijas, o ne vidinę struktūrą.

Sukuriamas laidos planas. T.y. bendras projekto planas. Laidos planas specifikuoja, kokie konkrečiai vartotojo pasakojimai bus įgyvendinti kiekvienoje sistemos laidoje, ir kokios tų laidų datos. Taip leidžiama užsakovui pasirinkti, kokius vartotojo pasakojimus būtina įgyvendinti pirmiausia, kurie jų yra svarbiausi/reikalingiausi. Laidos planas dar vadinamas įsipareigojimų tvarkaraščiu). Šis planas toliau naudojamas sukurti iteracijų planams.

Daryti daug mažų laidų. Kolektyvas turėtų dažnai išleisti sistemos versijas klientui(po tam tikrų iteracijų). Jau po pirmos iteracijos turėtume turėti veikiančią sistemą (kuri galbūt nieko nedaro, bet ją jau galima paleisti ir pačiupinėti).

Matuojamas projekto greitumas. Tai metrika, kuria matuojama kaip greitai atliekamas darbas šiame projekte. Laidos planavimo susirinkimo metu užbaigtų iteracijų greitumas gali būti naudojamas prognozuojant, kiek pasakojimų bus realizuota per sekančią iteraciją.

Projektas dalinamas į iteracijas. Iteracinis kūrimas teikia judrumo kūrimo procesui. Reikia padalinti visą kūrimo tvarkaraštį į krūvą iteracijų, 1 – 3 savaičių ilgio. Nereikia planuoti programavimo užduočių. Viena iš pagrindinių XP idėjų -- ateities atspėti neįmanoma. Reikalavimai gali pasikeisti. Kam daryti dabar tai, ko gal ateityje neprireiks.

Kiekviena iteracija pradedama nuo iteracijos planavimo. Kiekvienos iteracijos pradžioje padaromas iteracijos planavimo susirinkimas, kurio metu suplanuojama ši iteracija, t.y., kas jos metu bus realizuota. Kiekviena iteracija trunka nuo 1 iki 3 savaičių. Kokie pasakojimai bus įgyvendinti šioje iteracijoje, pasirenka užsakovas iš laidos plano.

Stumdyk žmones. XP nėra taip, kad pasodinam vieną programuotoją daryti grafinę sąsają, kitą daryti SQL sąsają ir pan. Visi programuotojai dirba su viskuo, tad jie visi susipažįsta su visomis projekto pusėmis. Žmonių „stumdymas“ bei programavimas poromis leidžia suteikia kryžminį apmokymą. Esant reikalui, galima perkelti programuotojus prie "karščiausios" projekto dalies ir jie visi bus produktyvūs.

Kiekviena diena pradedama nuo trumpo susirinkimo. Šio susirinkimo tikslas – kolektyvo bendravimas. Susirinkimas daromas stovint, kad diskusijos neužtruktų per ilgai. 5 minučių pakanka. Susirinkime identifikuojamos problemos, bet nebandoma jų spręsti.

XP pritaikymas. T.y. kiekviena organizacija turi savų ypatumų ir XP procesą reikės jiems pritaikyti. Nesakoma "jeigu", nes žinoma, kad reikės atlikti kai kuriuos pakeitimus konkrečiam projektui. Reikia naudotis XP taisyklėmis, tam, kad pradėtumėme, bet nereikia delsti, jei kažkas neveikia. Tai nereiškia, kad kolektyvas gali daryti, ką nori. Taisyklių turi būti laikomasi, kol kolektyvas jų nepakeičia. Kiekvienas programuotojas turi aiškiai žinoti, ko tikėtis iš kitų.

- ✓ Projektavimas:

Paprastumas. Sistemos dizainas turi būti kiek įmanoma paprastesnis. Paprasto dizaino sistemą daug lengviau palaikyti nei sudėtingą. Jei pamatome, kad kažkas yra labai sudėtinga, reikia pakeisti tai kažkuo paprastu.

Parinkti sistemos metaforą. Reikia parinkti sistemos metaforą, kad klases ir metodus žmonės vadintų vienodai. Tiek programuotojai, tiek vartotojai turi naudoti tuos pačius terminus, kad jiems bendraujant nekiltų nesusipratimų.

Naudoti CRC korteles projektavimo sesijoms. CRC tai Class, Responsibilities ir Collaboration. Jos naudojamos kuriant sistemos dizainą. Didžiausia CRC kortelių vertė yra ta, kad jos leidžia žmonėms atitrūkti nuo procedūrinio mąstymo ir geriau naudoti objektinę technologiją. CRC kortelės leidžia visam projekto kolektyvui dalyvauti kuriant sistemos dizainą. (Čia yra tokia sąvoka kaip kolektyvinė kodo nuosavybė, t.y. visi žino (beveik) viską apie visą kodą ir projektą, kiekvienas gali keisti bet kokią kodo eilutę ir suteikti naują funkcionalumą, taisyti klaidas ar pertvarkyti kodą/dizainą.

Rizikai mažinti kuriami spontaniški sprendimai. Idėja ta, jog jei kas nors labai neaišku (nežinai, kaip kažką padaryti, arba nežinai, kiek tai laiko užtruks), tai sėdi ir pradedi programuoti, eksperimentuoti be ypatingos disciplinos, be „Unit" testų. Kai išsiaiškini, kas buvo neaišku, išmeti šitą priprogramuotą kodą ir darai viską pagal taisykles ir kokybiškai. Tikslas – sumažinti riziką.

Joks funkcionalumas nepridedamas anksčiau laiko. Nereikia apkrauti sistemos papildomu funkcionalumu, kurio šiuo metu nereikia ir kuris, galbūt(o gal ir ne) bus naudojamas ateityje. Bet tik labai mažas procentas dažniausiai bus panaudojamas ateityje, ir toks darbas, būtų projekto stabdymas bei resursų rijimas.

Pertvarkoma visada ir visur, kur tik įmanom. Programuotojai yra linkę laikytis senų dizainų, nors jie galbūt jau pasidarė visai nepatogūs. Jie yra linkę naudoti kodą, kurio jau

nebeįmanoma suprasti ar palaikyti, bet kuris vis dar veikia, ir mes bijome jį modifikuoti. Bet tai neapsimoka. Kai mes išmetame pasikartojimus, nenaudojama funkcionalumą bei atnaujiname pasenusius dizainus, mes pertvarkome. Pertvarkymas viso projekto gyvavimo laiku sutaupo laiko ir pagerina kokybę. Pertvarkyti reikia negailestingai, kad dizainas liktų paprastas. Kodas turi būti švarus ir trumpas, kad jį būtų lengva suprasti, modifikuoti ir išplėsti.

- ✓ Kodavimas:

Užsakovas visada "po ranka". Vienas iš nedaugelio XP reikalavimų yra tai, kad sistemos užsakovas turi būti visada pasiekiamas/prieinamas. Jis yra dalis kolektyvo. Visos XP projekto fazės reikalauja bendravimo su užsakovu ir geriau akis į akį. Geriausia yra, kai užsakovas paskiria vieną ar kelis žmones(būtinai ekspertus) dirbti kartu su kūrėjų kolektyvu. Atrodytų, kad sueikvojama daug užsakovo laiko, tačiau daug jo ir sutaupoma, nereikalaujant detalios reikalavimų specifikacijos ir vėliau nepristatant jam "blogos" sistemos.

Kodas rašomas laikantis standartų. Turi būti sutarta pagal kokius standartus bus rašomas kodas. Kodas bus vienodai parašytas, lengvai skaitomas kitiems ir pertvarkomas.

Pirma parašomi modulių testai. Pirma parašius testus, paskui bus lengviau parašyti kodą. Modulių testų kūrimas padeda programuotojams suprasti, ką iš tikrųjų reikia padaryti. Reikalavimai pririšti prie testų. Paprastai ne visada yra aišku, kada programuotojas užbaigė darbą/įgyvendino visą būtiną funkcionalumą. O parašius testus tai tampa aišku – kada visi testai praeina. Kitas pliusas – testai, parašyti prieš rašant kodą, padeda palaikyti paprastesnį sistemos dizainą.

Visas kodas rašomas poromis. Visas išleidžiamas kodas yra kuriamas dviejų žmonių prie vieno kompiuterio. Programavimas poromis padidina programinės įrangos kokybę. Esant didesnei kokybei, vėliau projekte sutaupoma daugiau pinigų.

Vienu metu tik viena pora programuotojų integruoja kodą. Programuotojai ištestuoja savo kodą ir integruoja jį, tikėdami, kad viskas yra gerai. Bet dėl lygiagretaus integravimo galima kombinacija išeities tekstų, kurie buvo testuojami atskirai, bet kartu ne. Kyla integravimo problemos. Norisi, kad programuotojai galėtų dirbti lygiagrečiai, drąsiai darydami pakeitimus bet kuriai sistemos daliai, kada reikia, bet taip pat norisi, kad visa tai vyktų be klaidų. Taigi integravimas turi būti nuoseklus – kodas yra integruojamas, testuojamas. Tokiu būdu paskutinė versija yra visada apibrėžta. Esmė yra, kad kūrimas daromas prie vieno kompiuterio, o paskui programuotojai eina prie specialiai integravimui pastatytos mašinos ir ten daro integraciją.

Kodą reikia integruoti dažnai. Integruoti ir išleisti kodą reikia kuo dažniau ir negalima palikti pakeitimų ilgiau nei dienai. Visi turi dirbti su naujausia versija. Beveik nuolatinė

integracija anksti aptinka suderinamumo problemas. Jei integruosime po truputį visu projekto kūrimo metu, neteks pačioje pabaigoje kankintis savaites bandant integruoti sistemą.

Kolektyvinė kodo nuosavybė. Tai reiškia, jog visi programuotojai gali keisti visas projekto dalis. Nėra kokių nors posistemių prižiūrėtojų, pro kuriuos reiktų prastūminėti pakeitimus. Praktika rodo, jog kolektyvinė kodo nuosavybė yra patikimesnė, nei atsakomybės už atskiras klases paskirstymas atskiriems žmonėms. Ypač atsižvelgiant į tai, jog tie žmonės gali bet kada palikti projektą.

Optimizavimas atliekamas pačioje pabaigoje. Niekada nereikia spėlioti iš anksto, kur bus sistemos silpnoji vieta. Reikia tai tiesiog išmatuoti – t.y. iš pradžių padaryk, kad veiktų, po to padaryk, kad veiktų teisingai, o tada padaryk, kad veiktų greitai.

Jokių viršvalandžių. Darbas viršvalandžiais gadina kolektyvo nuotaiką ir "varomąją jėgą". Vietoj to geriau perplanuoti, pakeisti projekto tvarkaraščius ar apimtį.

- ✓ Testavimas:

Testas kuriamas kiekvienam moduliui. Modulių testai yra vienas iš kertinių XP dalykų. Pirma yra sukuriamas arba parsisiunčiamas modulių testų karkasas/infrastruktūra (framework'as), kad galima būtų kurti automatizuotus modulių testų rinkinius. Iš pradžių turi būti kuriami testai, o paskui kodas. Kodas yra išleidžiamas kartu su testais.

Parašytas kodas turi praeiti visus modulių testus, prieš išleidžiant naują versiją.

Jei randama klaida, kuriami testai. Testai kuriami, norint apsisaugoti nuo tos klaidos, kad ji nepasikartotų. Klaida produkcijoje reikalauja priėmimo testų sukūrimo.

Priėmimo testai praleidžiami dažnai, fiksuojami ir peržiūrimi jų rezultatai. Priėmimo testai kuriami remiantis vartotojų pasakojimais. Iteracijos eigoje realizuojami vartotojų pasakojimai verčiami į priėmimo testus. Anksčiau priėmimo testai buvo vadinami funkciniais testais. Pavadinimas pakeistas, nes jis geriau atspindi jų prasmę – užsakovo reikalavimai yra atlikti ir sistema yra priimtina. [15]

9 lentelė

XP analizė

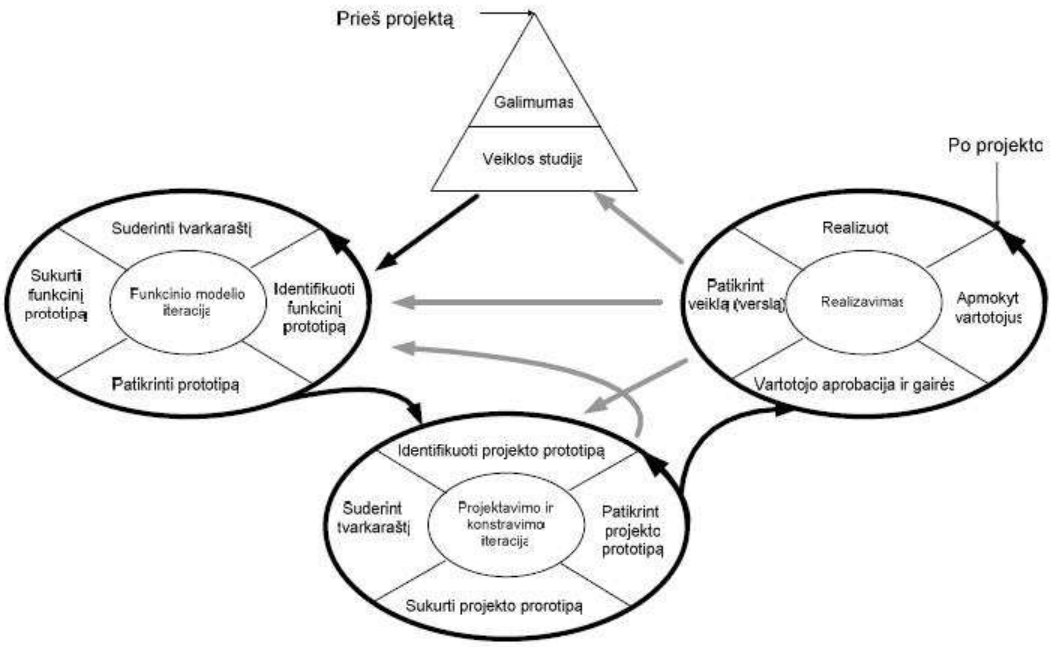
XP savybė	Tinkamumas programų sistemų kūrimui
Labiausiai tinka mažoms komandoms.	Ši savybė tinka, nes pagal statistika projektai dažniausiai pavyksta, kai būna nedidelės komandos.
Reikalingas betarpiškas bendradarbiavimas tarp komandos narių.	Ši savybė tinkama, nes bendradarbiavimas reikalingas, kad sukurtumėme tinkama produktą. Jei vienas programuotojas darys tik savo neatsižvelgdamas į kitą programuotoją, tai gali

	nepavykti projektas.
Naudojantis XP metodu, labai svarbus yra testų rašymas. Pirmiausia realizuojamas testas, o po to kodas. Testuojama yra nuolatos.	Puikiai tinka savybė, nes galima iš anksto aptikti klaidas ir jas iškart pašalinti.
XP vienas iš praktikuojamų dalykų yra programavimas poromis, kai du žmonės vienu metu rašo kodą prie vieno kompiuterio.	Tinkama savybė, nes programuojant dviese greičiau bus pastebėtos klaidos, lengviau rašomas kodas, nes bus pasitarimas tarp abiejų programuotojų. Be to, jei vienas iš programuotojų susirgtų ar išvyktų, kitas galėtų programuoti toliau, nes būtų su viskuo susipažinęs ir žinotų, ką reikia daryt.
XP propaguoja tai, kad visi komandos nariai galėtų daryti pakeitimus, t.y. bet kuris komandos narys bet kada gali pakeisti bet kurią kodo dalį.	Nelabai tinkama savybė, nes kiekvienas narys yra atsakingas už savo kodo dalį ir nederėtų lįsti ten, kur neišmanai.
Sistemos funkcionalumas apibrėžiamas iš pasakojimų, kaip sistema turi veikti. Vėliau tai naudojama užsakovo ir programuotojo.	Dalinai tinkama savybė.

Šaltinis: sudaryta autoriaus pagal Butkienė R.
ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

1.8.4 Dynamic System Development Method (DSDM)

Paskutinius 15 metų programavimo specialistai intensyviai ieškojo naujų, efektyvesnių programų kūrimo technologijų, kurios geriau tenkintų verslo poreikius. Viena naujausių technologijų – tai Dinaminis sistemų kūrimo metodas (Dinamic Systems Development Method) DSDM metodas buvo atrastas Didžiojoje Britanijoje 1990 m. Šis metodas praktiškai šioje šalyje tapo greito programinių sistemų kūrimo (RAD – rapid application development) technologiniu standartu. DSDM devyni principai apima vartotojo įtraukimą, komandos sprendimus, integruotą projekto gyvavimo ciklo testavimą, grįžtamuosius pokyčius vystyme. DSDM esmė trumpai apibūdinama taip: nuoseklus vis naujų programų prototipų kūrimas, darbo eigoje su vartotojų pagalba išsiaiškinant reikalavimus programai. Laiko limitas ir kiti turimi resursai apsprendžia, kokios funkcijos bus kuriamoje programoje. Tokios nuostatos visiškai skiriasi nuo tų programų kūrimo technologijų nuostatų, kur funkciniai reikalavimai programai lemdavo darbui reikalingų resursų apimtis, tame tarpe ir laiko sąnaudas, reikalingas programai sukurti. Dažnai programa morališkai pasendavo, dar jos nebaigus sukurti. [26]



Pav. 10 DSDM proceso schema

Šaltinis: sudaryta autoriaus pagal Butkienė R.
ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

10 lentelė

DSDM analizė

DSDM savybė	Tinkamumas informacinių sistemų kūrimui
Pagrindinė DSDM idėja yra ta, jog vietoje to, kad pirmiau fiksuoti produkto funkcionalumo apimtį ir po to skaičiuoti kiek tam funkcionalumui realizuoti reikės laiko ir kitų išteklių, verčiau pirmiau fiksuoti laiką bei kitus išteklius ir pagal juos priderinti sistemos funkcionalumo apimtį. [8]	Visiškai netinkama informacinių sistemų kūrimui, nes kuriant sistemą svarbiausia yra, kad ji atitiktų reikiama funkcionalumą. Ir pagal tai turėtų būti skaičiuojami laiko ir kiti ištekliai.
Vartotojų apmokymas	Labai svarbi savybė, nes sukūrus programinę įranga, būtina apmokyti vartotojus kaip su ja dirbti.
Tvarkaraščių suderinimas	Tinkama savybė, nes patiems darbuotojams bei užsakovams labai svarbios yra datos.

Šaltinis: sudaryta autoriaus pagal Butkienė R.
ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

1.8.5 Savybių vairuojamas kūrimo procesas (angl. Feature Driven Development)

Tai yra pakankamai naujas metodas, sukurtas apie 2000 m. Jo autorius yra Jeff De Luca.

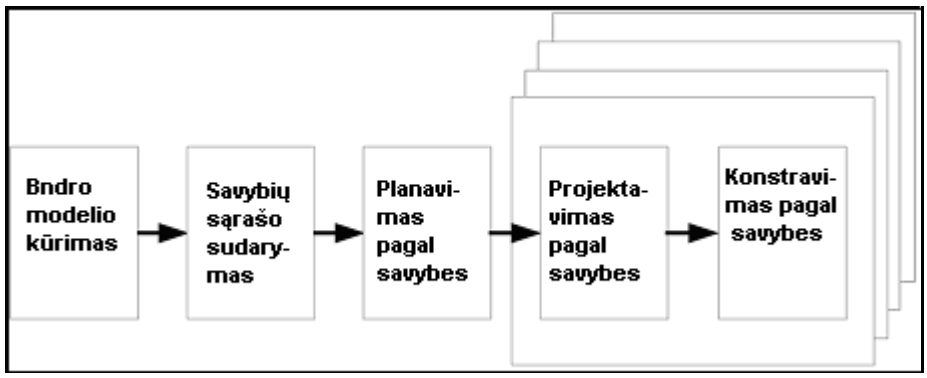
FDD proceso žingsniai: bendro modelio kūrimas, savybių sąrašo sudarymas, planavimas pagal savybes, projektavimas ir konstravimas pagal savybes.

FDD labiausiai tinkamas veiklai kritinėms programų sistemoms kurti (jei reikia jas sukurti laiku). [7]

Metodas iš esmės apima tik sistemos projektavimo ir konstravimo etapus.

FDD proceso etapai: bendro modelio kūrimas, savybių sąrašo sudarymas, planavimas pagal savybes, projektavimas ir konstravimas pagal savybes.

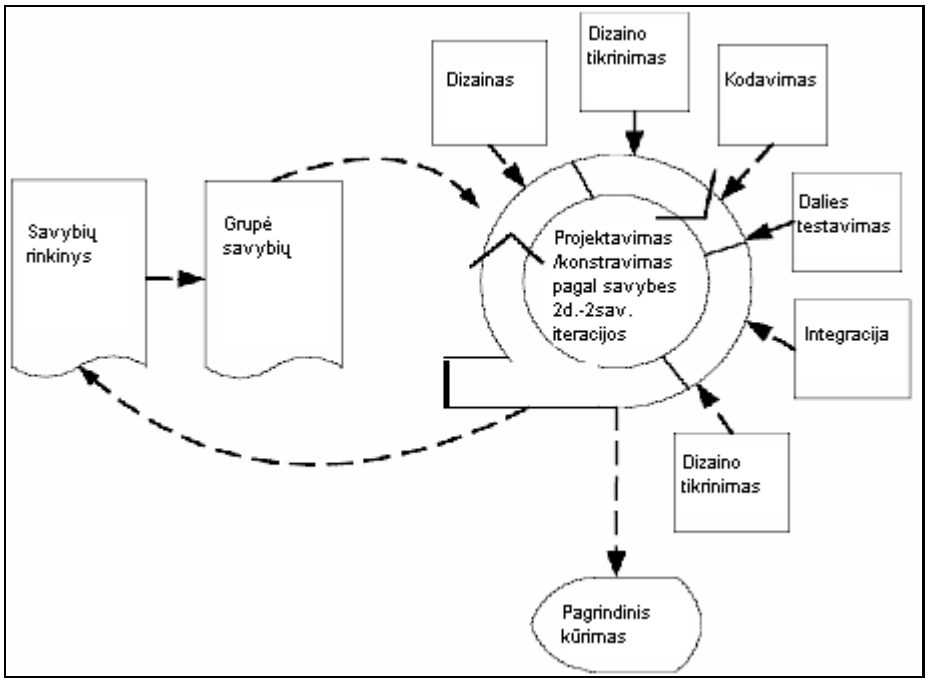
FDD susideda iš penkių nuoseklių procesų, per kuriuos atliekamas projektavimas ir sistemos sudarymas. Pasikartojanti FDD procesų dalis (Projektavimas ir Konstravimas) palaiko judrų išsivystymą su greita adaptacija į vėlyvius reikalavimų ir verslo poreikio pakeitimus. Tipiškai, ypatybės iteracija apima vienos iki trijų savaitės periodą darbo komandai (R. Butkienė. 2006)



Pav. 11. FDD proceso schema

Šaltinis: Butkienė R. (2007) ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

Kai bendro modelio kūrimas prasideda, srities specialistai jau žino sritį, kontekstą ir būsimos sistemos reikalavimus. Reikalavimai tokie kaip naudojimo atvejai ar funkcinė specifikacija egzistuoja šioje stadijoje. Tačiau FDD nekreipia dėmesio į reikalavimų rinkimą bei vadovavimą. Yra pristatoma taip vadinama „walkthrough“, kuriame komandos nariai ir vyriausias architektas yra informuoti apie aukšto lygio sistemos apibūdinimą. Po kiekvieno „walkthrough“ plėtojimo komanda dirba mažose grupelėse, kad pagamintų kiekvienai sferai po vieną objekto modelį. Tada plėtojimo komanda aptaria ir apsisprendžia dėl tinkamų objekto modelių kiekvienoje sferoje. Tuo metu yra konstruojama pavyzdinė bendro modelio sistema.



Pav. 12. FDD proceso ciklas

Šaltinis: Butkienė R. (2007) ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

11 lentelė

FDD analizė

FDD savybė	Tinkamumas Informacinių sistemų kūrimui
Iš pradžių kuriamas visos sistemos modelis, kad srities ekspertas suprastų sistemos apimtį, kontekstą, reikalavimus. Modelis yra suskirstomas į sritis ir komanda grupelėmis kuria objektų modelius.	Tinkama savybė
Sudaromas visapusiškas savybių sąrašas, kuris yra vis tikslinamas.	Kartais nebūtina iš karto sudaryti savybių sąrašo. Geriau tai daryti palaipsniui.
Už klasę atsakingas tik vienas asmuo.	Dažniausiai tinkama savybė
Tinkamas dideliems projektams.	Šita savybė netinkama, nes yra kuriamas metodas nedideliems projektams
FDD neapima reikalavimų surinkimo, vartotojo sąsajos dizaino, testavimo ir išdėstymo.	Netinkama savybė, nes kuriamai sistemai reikia reikalavimų surinkimo, vartotojo sąsajos dizaino bei testavimo.

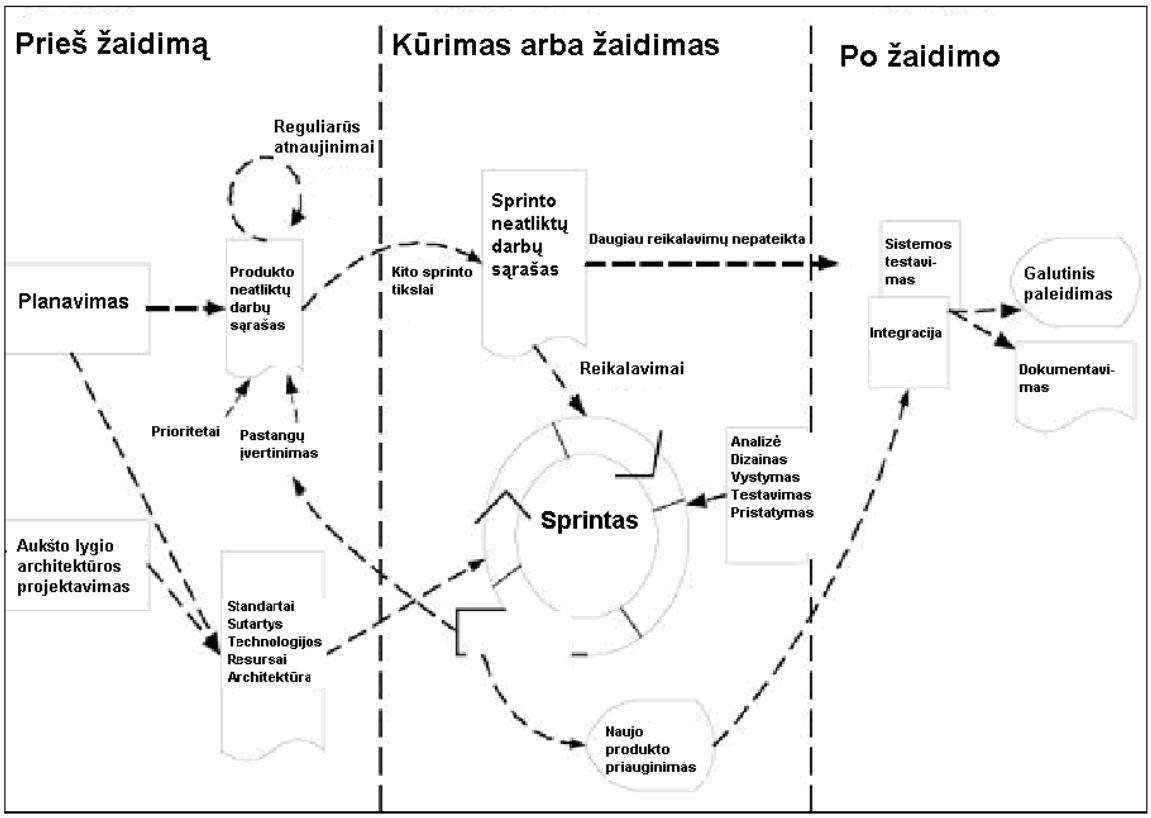
Šaltinis: sudaryta autoriaus pagal Butkienė R.
ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

1.8.6 Scrum metodas

Scrum metodo idėja paskelbta 1987 m., pats metodas aprašytas 2002 m.

Sistemos realizavimo priemonėms nekeliami apribojimai. Pagrindinis uždavinys – veiklos organizavimas, kad besikeičiančių reikalavimų aplinkoje būtų lanksčiai kuriama sistema.

Scrum pagrindinė idėja – įtraukiant naujas greito valdymo veiklas, kurių paskirtis – pastoviai identifikuoti trūkumus ir kliūtis kūrimo procese ir naudojamose priemonėse. Kūrimo metu gali kisti reikalavimai, todėl tuo pat metu gali pasikeisti ir programos kūrimo laikas, ištekliai, naudojamos technologijos [7].



Pav. 13. Scrum proceso schema

Šaltinis: Butkienė R. (2007) ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

Schemoje pateikti 3 *Scrum* proceso fazės: prieš žaidimą, kūrimas arba žaidimas, po žaidimo.

Prieš žaidimą fazė susideda iš: planavimo ir aukšto lygio architektūros projektavimo.

Planavimas apima išvystytos sistemos apibrėžimą. Neįvykdytų produktų sąrašas yra sudaromas pagal visus reikalavimus, kurie šiuo metu jau yra žinomi. Reikalavimai gali atsirasti iš kliento, pardavimų ir rinkodaros skyriaus, programinės įrangos supirkėjų. Reikalavimai yra išdėstyti pagal svarbą, todėl visos pastangos juos įgyvendinti yra įvertintos. Neįvykdytas produkto sąrašas yra pastoviai atnaujinamas su naujais ir išsamesniais dalykais, taip pat su tikslesniu įvertinimu bei naujais prioritetais. Planavimas apima projektinės

komandos apibrėžimą, įrankius ir kitus išteklius. Kiekvienoje iteracijoje Scrum komanda apžvelgia atnaujinto produkto neatliktus darbus (Backlog), kad įsipareigotų kitam kartojimui..

Architektūros fazėje, aukšto lygio sistemos projektas, apimdamas ir architektūra, yra pagrįsta einamaisiais dalykais neatliktų darbų sąraše. Perkėlimo į egzistuojančią sistemą atveju, pakeitimai būtini, kad neįvykdytų darbų sąrašas yra identifikuojamas su problemomis, kurias jie gali sukelti. Projekto peržiūros susirinkimas reikalingas tam, kad peržiūrėti ištesėjimo pasiūlymus ir kad priimamas sprendimai paremti šio susirinkimo pagrindu.

Išsivystymo fazė (taip pat pavadinta žaidimo faze) yra judri Scrum siūlymo dalis. Ši fazė laikoma kaip „juoda dėžė“, kur yra laukiami nenuspėjami dalykai. Skirtingi aplinkos ir techniniai Scrum kintamieji (tokie kaip kokybė, reikalavimai, ištekliai, technologijos ir įrankiai ir net išsivystymo metodai), kurie gali pasikeisti proceso metu, yra stebimi ir valdomi per įvairias Scrum praktikas. Scrum siekia kontroliuoti juos, kad galėtų lanksčiai prisitaikyti prie pasikeitimų.

Išsivystymo fazėje sistema yra išvystyta „Sprinte“. „Sprintas“ yra pasikartojantys periodai, kur funkcionalumas yra išvystytas ar išplėstas, kad pagamintų naują prieaugį. Kiekvienas „Sprintas“ apima tradicines programinio išsivystymo fazes: reikalavimas, analizė, kompozicija, vystymasis ir pristatymas. Architektūra ir sistemos kompozicija plėtojasi per „Sprinto“ išsivystymą. Yra suplanuota, kad vienas „Sprintas“ truktų nuo vienos savaitės iki vieno mėnesio. Viename sistemos išsivystymo procese gali būti nuo trijų iki aštuonių „Sprint‘ų“, kol sistema bus paruošta dalijimui. Taip pat gali būti daugiau kaip viena komanda, sudaranti prieaugį.

Po žaidimo fazė turi savyje paleidimo užbaigimą. Ši fazė būna įvesta, kai būna sudaryta sutartis, kad aplinkos kintamieji tokie kaip reikalavimai yra užbaigti. Šiuo atveju, negali būti išrasta naujų elementų ir problemų. Sistema yra paruošta paleidimui.

12 lentelė

Scrum analizė

Scrum savybė	Tinkamumas informacinių sistemų kūrimui
Vykdant *Scrum* metodą, yra akcentuojami neatliktų darbų sąrašai, pastangų įvertinimai, kasdieniai susirinkimai.	Tinkama savybė, nes aptariami neatlikti darbai ir tada kiekvienas žino, ką reikia daryti toliau.
Lankstumo, adaptyvumo ir produktyvumo idėja.	Tinkama savybė
Kintantys reikalavimai yra priimtini.	Tinkama savybė, nes užsakovo reikalavimai

	gali kisti labai dažnai.
Nuolatinis trūkumų ir kliūčių kūrimo procese identifikavimas.	Reikalinga, nes galima būtų tai atlikti kiekvienos iteracijos metu pasitariant su užsakovu.
Kasdieniniai susirinkimai	Reikalingi, kad aptarti neatliktų darbų sąrašus.
Neatliktų darbų sąrašai	Tinka.

Šaltinis: sudaryta autoriaus pagal Butkienė R. ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

1.8.7 Crystal metodas

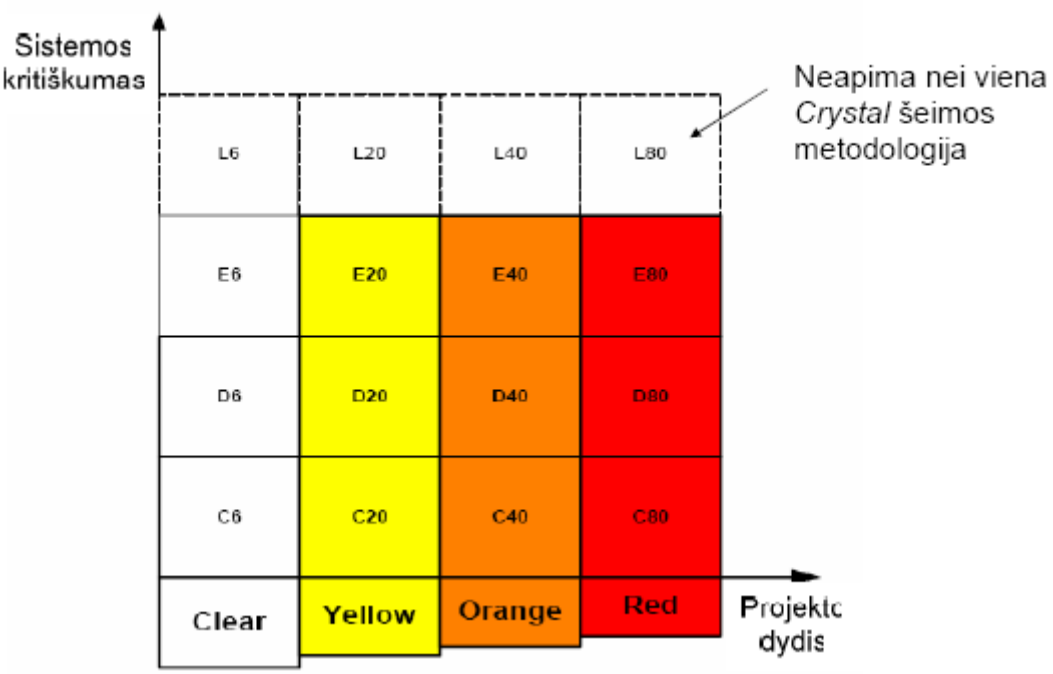
Alistair Cockburn yra autorius “Crystal” šeimos metodų. Crystal metodas remiasi žmonių aspektais - bendradarbiavimas, pilietybė, parama. Alistair naudoja projekto dydį, tikslus kaip tinkamai apmokyti kiekvieną vartotoją Crystal šeimos metodologijos.

Crystal siūlo keletą skirtingų metodologijų, iš kurių galima pasirinkti tinkamiausią konkrečiam projektui. Taip pat siūlo metodologijos pritaikymo principus atsižvelgiant į permainingus skirtingų projektų aplinkybes. Tai yra į žmogų orientuota metodologijos. Nesvarbu koks projekto dydis ir kokie jo prioritetai, ši metodologija leidžia iki praktiškumo sumažinti popierinį darbą, pridėtines išlaidas, biurokratizmą.

Crystal šeimos metodologijos remiasi tokiais samprotavimais:

- Kiekvienam projektui reikia šiek tiek kitokių strategijų, susitarimų arba metodologijos;
- Projekto darbų kokybė priklauso nuo juos atliekančių žmonių savybių; ji pagerėja, jei pagerėja žmonių savybės; jei individas dirba geriau, tai ir visos komandos darbas gerėja;
- Geresnis ir dažnesnis bendravimas sumažina tarpinių darbu produktų kiekį;
- Kiekvienas Crystal šeimos narys pažymėtas spalva – kuo spalva tamsesnė, tuo metodologija sunkesnė;
- Siūloma rinktis metodologiją atsižvelgiant į projekto dydį ir kritiškumą:
    - Kuo didesnis ar kritiškesnis projektas, tuo sunkesnės metodologijos reikalauja;
- Projekto dydis matuojamas asmenų įtrauktų į sistemos kūrimą skaičiumi;
- Projekto kritiškumas gali būti įvertintas taip:

- Comfort (C) - tai reiškia, kad sugedus sistemai vartotojas praras tik komfortą;
- Discretionary Money (D);
- Essential Money (E);
- Life (L)



Pav. 14 Crystal metodologijų rūšiavimo schema.

Šaltinis: Butkienė R. (2007) ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

Yra trys pagrindinės metodologijos:

- Crystal Clear – labai mažiems projektams (D6, su praplėtimais tinka ir E8/D10 projektams);
- Crystal Orange – vidutinio dydžio projektams (D40);
- Crystal Orange Web.

Bendros Crystal šeimos metodų savybės:

- Priauginantys kūrimo ciklai, kurių įprastinė trukmė 1-3 mėnesiai (bet beviršija 4 mėnesių);
- Išryškinamas bendravimas ir bendradarbiavimas tarp žmonių;
- Neriboja kūrimo priemonių, įrankių, įpročių;
- Sumažina tarpinių darbo produktų;

- Leidžia pritaikyti metodologiją individualiam projektui ir ją plėtoti projekto metu;
- Visos Crystal šeimos metodologijos pateikia gaires politikos standartams, darbo produktams, „vietiniams reikalams“, įrankiams, standartams ir vaidmenims, kurių reikia laikytis kūrimo procese.
- Politikos standartai:
    - Priauginantis kūrimas įprastinėje bazėje;
    - Progreso matavimas sukurta programine įranga ir priimtais svarbiausias sprendimais, o ne parašytais dokumentais;
    - Tiesioginis vartotojo įtraukimas;
    - Funkcionalumo testavimas automatizuotos regresijos būdu;
    - Produktą peržiūri 2 vartotojai;
    - Seminarai produkto ir metodologijos suderinimui kiekvieno ciklo pradžioje ir vidury.

Crystal darbo produktai:

- Bedri:
    - Produktų išleidimo seka;
    - Bendrų objektų modeliai;
    - Vartotojo vadovas;
    - Testuojami atvejai;
    - Migravimo kodas.

- Crystal Clear:
    - Panaudojimo atvejų arba savybių aprašai;
    - Tvarkaraštis tik vartotojų peržiūrai, produktų sukūrimui ir palyginimui;
    - Ekraninių formų eskizai, projekto pamatai, pastabos.

- Crystal Orange:
    - Reikalavimų dokumentas
    - Tvarkaraščių daugiau nei Crystal Clear;
    - Vartotojo interfeiso projektinę dokumentacija, specifikacijos komandoms bendradarbiauti;
    - Būsenos ataskaita.

Crystal Clear vaidmenys

Viena komanda, kurios nariai atlieka tokius vaidmenis:

- Rėmėjas;
- Vyresnysis projektuotojas-programuotojas;
- Projektuotojas-programuotojas;
    - Biznio klasės projektuotojas;
    - Programuotojas;
    - Programinės įrangos dokumentuotojas;
    - Dalies testuotojas.
- Vartotojas;
- Visi kiti dar gali atlikti tokius vaidmenis:
    - Koordinatorius;
    - Veiklos ekspertas;
    - Reikalavimų surinkėjas.

Crystal Orange vaidmenys

Keletas komandų, pagal kurias sugrupuojami tokie vaidmenys:

- Crystal Clear vaidmenys;
- Vartotojo interfeiso projektuotojas;
- DB projektuotojas;
- Naudojimo ekspertas;
- Technikas;
- Veiklos analitikas/projektuotojas;
- Architektas;
- Projekto vadovas;
- Atsakingasis už pakartotinį panaudojimą;
- Rašytojas;
- Testuotojas.

(R. Butkienė, 2006)

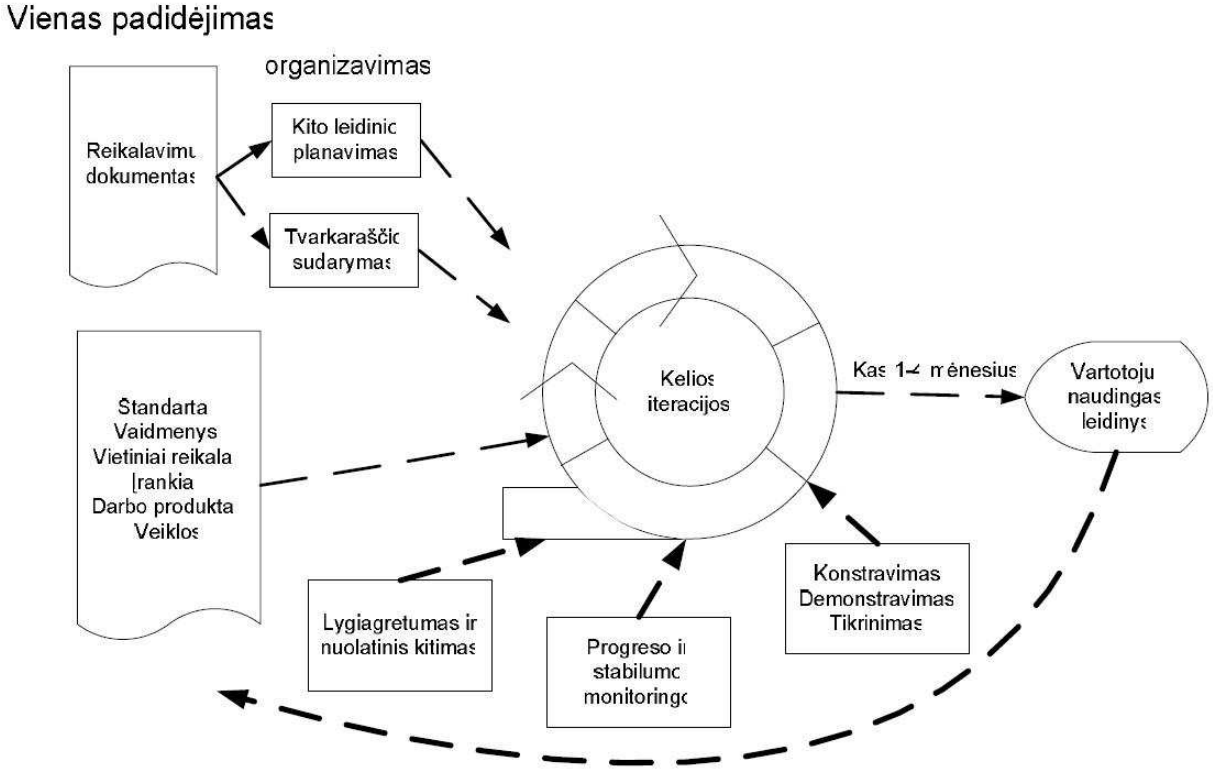
13 lentelė

Crystal analizė

Crystal metodo savybės	Tinkamumas informacinių sistemų kūrimui
Tvarkaraščių sudarymas	Tinkama savybė, nes visi nori žinoti datas
Monitoringas	Ganėtinai tinkama savybė

Kito leidinio planavimas	Netinkama

Šaltinis: sudaryta autoriaus pagal Butkienė R. ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf



Pav. 15. Crystal proceso schema

Šaltinis: Butkienė R. (2007) ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

Agile metoduose praktikuojamų dalykų apibendrinimas pateiktas 14 lentelėje

14 lentelė

Agile metoduose praktikuojamų dalykų apibendrinimas.

	XP	Scrum	Crystal	FDD	DSDM	ASD
1	Žaidimo planavimas	Neatliktų darbų sąrašas	Režisavimas	Srities objektų modeliavimas	Aktyvus vartotojo įtraukimas	Iteratyvus kūrimas
2	Maži/trumpalaikiai leidiniai	Pastangų įvertinimas	Revizija ir peržiūra	Kūrimas pagal savybes	Komandos įgalintos priimti sprendimus	Savybėmis (komponentais) grindžiamas planavimas
3	Metafora	Sprintas	Monitoringas	Individualios klasės (kodo) nuosavybė	Greitas produktų pateikimas	Peržiūros užsakovo akimis
4	Paprastas projektavimas	Susirinkimas sprintui suplanuoti	Lygiagretumas ir nuolatinis kitimas	Savybių komanda	Atitikimas veiklos tikslams yra pagrindinis produkto priėmimo kriterijus	
5	Testavimas	Sprinto neatliktų darbų sąrašas	Suskaidymo pagal įvairumą strategija	Inspektavimas	Įtertyvus ir priauginantis kūrimo procesas	
6	Kodo perrašymas	Kasdieninis Scrum susirinkimas	Metodologijos derinimo metodika	Pastovus konstravimas	Visi pakeitimai yra atšaukiami	

7	Programavimas poromis	Sprinto peržiūros susirinkimas	Vartotojo peržiūros	Konfigūracijos valdymas	Fiksuojami tik aukščiausio lygio reikalavimai	
8	Kolektyvinė nuosavybė		Apmastymų seminarai	Ataskaitos apie progresą	Integruotas testavimas	
9	Pastovus integravimas				Bendravimas ir bendradarbiavimas visų projektų dalyvių yra esminis	
10	Darbas 40 val. Per savaitę.					
11	Užsakovas visada šalia					
12	Kodavimo standartai					
13	Atvira darbo vieta					
14	Tik taisyklės					

Šaltinis: Sudaryta autoriaus

Agile metodų procesų etapų lentelė pateikta 15 lentelėje

15 lentelė

Agile metodų procesai

Metodas	**Proceso etapų skaičius**	**Proceso etapai**	**Komentarai**
XP	6	Tyrimas, planavimas, leidinių kūrimo iteracijos, gamyba, palaikymas, mirtis.	Prieinamas, tačiau detalus metodo aprašas prieinamas tik konsorciumo nariams.
DSDM (Dynamic System Development Method)	7	Prieš-projektinis etapas, galimybių studija, verslo (veiklos) studija, funkcinio modelio iteracija, projektavimo ir konstravimo iteracija, realizavimas, po-projektinė stadija.	Praktikuojami dalykai tinka daugelyje situacijų, tačiau bendram vaizdui ir valdymui skirta mažai dėmesio.
ASD (Adaptive Software Development)	3	Spėlioti, bendradarbiauti, mokytis.	Kalba daugiau apie sąvokas ir kultūrą, nei apie programinės įrangos kūrimo praktiką
Crystal	5	Tvarkaraščio sudarymas, kito leidinio planavimas, Režisavimas, vartotojo peržiūros, monitoringas.	Remiasi daugiau žmonių aspektais: bendradarbiavimas, pilietybė, parama
FDD (Feature Driven Development)	5	Bendro modelio kūrimas, savybių sąrašo sudarymas, planavimas pagal savybes, projektavimas ir konstravimas pagal savybes.	Metodas yra paprastas, sistema projektuojama ir realizuojama pagal savybes. Skirtas tik

			projektavimui ir realizavimui. Reikalingi kiti palaikantys metodai (požiūriai).
Scrum	3	Prieš žaidimą, kūrimas arba žaidimas, po žaidimo.	Detalizuoja kaip valdyti 30-dienų Srum ciklą, bet nedetalizuoja integravimo ir priėmimo testų.

Šaltinis: sudaryta autoriaus pagal Butkienė R.
ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

16. Lentelė

Bendros Agile programinės įrangos išsivystymo metodų ypatybės.

Metodas	Pagrindiniai punktai	Ypatybės	Identifikuoti trūkumai
ASD	Adaptyvumas: bendradarbiavimas; misijos vedamas, komponentinis, iteratyvus kūrimas.	Organizacijos matomos kaip adaptyvios sistemos	ASD yra labiau apie sąvokas ir kultūrą bei apie programinės įrangos praktiką.
Crystal	Metodų šeima. Kiekvienas turi tas pačias pagrindines vertes ir principus, bet skiriasi priemonės, vaidmenys, įrankiai bei standartai.	Metodo projektavimo principai. Gebėjimas išrinkti labiausiai tinkamą metodą, pagrįsta projektiniu dydžiu ir svarba.	Per anksti dar vertinti: Tik du iš keturių siūlytų metodų egzistuoja.
DSDM	Kontrolės pritaikymas RAD, laiko įrėminimas, įgaliotos DSDM komandos, aktyvus metodo plėtrą prižiūrintis DSDM konsorciumas.	Pirmas iš tikrųjų judrus programinės įrangos išsivystymo metodas, kelių vartotojų vaidmenų: "ambasadorius", 'Aiškiaregys" ir "patarėjas".	Metodas yra pasiekiamas, bet tik konsorciumo nariai turi prieigą prie detalaus metodo aprašo. tikras metodo naudojimas.
FDD	Penkių žingsnių procesas, objektiniais komponentais grįstas kūrimas. Labai trumpos iteracijos“ nuo 1 valandos iki 2 savaičių.	Metodo paprastumas, sistemos projektavimas ir realizavimas pagal savybes.	FDD kreipė dėmesį tik į projektavimą ir realizavimą. Reikia kitų palaikymo metodų.
XP	Užsakovo “vairuojamas” kūrimas, mažos komandos, kasdieniai produktai	Faktoringas iš naujo - tebevykstantis sistemos perprojektavimas, kad pagerint jos atlikimą ir jautrumą pasikeitimams.	Tuo metu, kai atskiros praktikos yra tinkamos daugeliui situacijų, bet bendro vaizdo ir vadybos praktikoms skiria mažiau dėmesio.
Scrum	Nepriklausomos, mažos, saviorganizuojančios išsivystymo komandos. 30 dienų produkto versijos sukūrimo ciklas	Leidžia peršokti nuo "apibrėžtas ir pakartojamas” prie “Naujas Scrum požiūris produktui sukurti”	Nors detalizuoja, kaip valdyti 30-dienų Scrum ciklą, bet nedetalizuoja integravimo ir priėmimo testų.

Šaltinis: sudaryta autoriaus

17 Lentelė

Agile metodų apibendrinimas

Metodas	Savybės	Tinkamumas
XP	Testų rašymas prieš kodo kūrimą	+
	Programavimas poromis.	+
	Komunikavimas su klientais	+

	Komunikavimas su kitais programuotojais	+
	Paprastas dizainas	+
	Kolektyvinė kodo nuosavybė.	+
	Kasdieniai produktai	–
Crystal	Apmastymų seminarai, kurie vyksta prieš ir po kiekvieno ciklo	–
	Kito ciklo planavimas	–
	Progreso matavimas praeitais etapais ir etapo stabilumu	–
	Dviejų vartotojų peržiūra po kiekvieno ciklo	+
ASD	Kadangi aplinka dažniausiai nėra gerai suprasta ar apibrėžta, o kintanti, tai kūrimo pastangos turi būti sutelktos perdaryti produktą, o ne padaryti jį iš karto gerai.	–
	Skatina projekto dalyvius anksčiau priimti neišvengiamus, sunkiai suderinamus sprendimus	–
	Rizikingiausių elementų kūrimas turi būti pradėtas kuo anksčiau.	–
	Prisitaikyti prie pasikeitimų yra svarbiau, nei juos kontroliuoti	–
	Visos kiekvieno ciklo veiklos turi būti pagrįstos projekto misija.	–
	Visos kiekvieno ciklo veiklos turi būti pagrįstos viso projekto misija.	–
DSDM (Dynamic System Development Method)	Tvarkaraščio sudarymas	+
	Vartotojo apmokymas	+
	Pirmiausia fiksuojamas laikas bei kiti ištekliai, ir pagal juos priderinama sistemos funkcionalumo apimtis.	–
	Prototipų kūrimas	+
FDD (Feature Driven Development)	Pirmiausia sukuriamas visos sistemos modelis	+
	Sudaromas visapusiškas savybių sąrašas	–
	FDD neapima reikalavimų surinkimo, vartotojo sąsajos dizaino, testavimo ir išdėstymo.	–
	Labai trumpos iteracijos (nuo 1 val. iki 2 sav.)	+
Scrum	Pastoviai identifikuoja trūkumus ir kliūtis kūrimo procese ir naudojamose priemonėse.	+
	Kasdieniai susirinkimai	–
	Neatliktų darbų sąrašai	+
	Susirinkimas sprintui suplanuoti	+
	Iteracijos trukmė yra 2-4 sav.	–
ICONIX	Automatizuotas būsenų diagramos generavimas	+
	Sekos diagramos, kurios kartu su statine klasių diagrama toliau panaudojamos kodo rašymui	+

Šaltinis: sudaryta autoriaus

Kadangi Agile metodų yra nemažai, ir jie visi yra pakankamai geri, bet kiekvienas iš jų turi tokių savybių, kurios nėra būtinos ar net netinka programų sistemų kūrimui. Todėl pabandysime sudaryti naują metodą, remdamiesi jau esamų metodų savybėmis.

Yra susidariusi nuomonė, kad Agile metodologijoje nėra dokumentacijos, modeliavimo, planavimo, nenuspėjama sistema, nenustatoma projekto suma. Bet iš tikrųjų realybė yra ta, kad yra ir dokumentacija, ir modeliavimas, ir planavimas, ir t.t. Bet Agile metoduose nėra nurodyta, kaip viskas vyksta, todėl prie Agile metodų prijungsim ICONIX metodą. ICONIX metodas pasižymi minimaliu UML diagramų kiekiu ir efektyvia metodika, kurios dėka kūrimo procesas yra greitas ir efektyvus. ICONIX reikalavimų apibrėžimo proceso etapai: probleminės srities aprašymas, panaudojimo atvejų aprašymas, išbaigtumo analizė ir sekos diagramos sudarymas.

Todėl, išanalizavus Agile metodų savybes, pastebime, kad metodo, kuris paimtų visas reikiamas savybes – nėra. Todėl paimant iš įvairių metodų mums reikalingas savybes, sudarysim naują metodą, kuris bus pritaikomas programų sistemoms kurti.

Programų sistemų kūrimui geriausia tiktų XP, FDD ir Scrum bei ICONIX metodai. Tačiau kad sistemų kūrimas būtų efektyvus reikėtu vadovautis kitokiais prioritetais. Šie prioritetai apimtų visos sistemos modelio bei tvarkaraščių sudarymus, taip pat greitas kūrimas, stengiantis skirti kuo mažiau dėmesio kasdieniniams susirinkimams, nenutrūkstantis komunikavimas su užsakovais, bei komandos narių, nuolatinis trūkumų ir kliūčių kūrimo procese identifikavimas. Programų sistemų kūrime svarbus vartotojų apmokymas.

XP metodas skirtas mažoms 2 - 10 programuotojų grupėms . XP yra sudėtinga pritaikyti kur dirba daug darbuotojų. Pastebėta, kad projektuose, kuriuose reikalavimai dinaminiai ar yra didelė rizika, mažam XP programuotojų kolektyvui sekasi labiau nei dideliam kolektyvui. Darbui su XP užtenka ir paprastų programuotojų – nereikia labai aukštos kvalifikacijos specialistų.

FDD metodas neapima reikalavimų surinkimo, testavimo. Metodas apima tik projektavimo ir realizavimo etapus. Norint taikyti šį metodą, reikalinga remtis ir kitais požiūriais. Šis metodas labiausiai tinkamas kritinėms sistemoms kurti, kai reikalinga jas sukurti laiku. Tam turi būti išsamiai aprašomos visos sistemos savybės (R. Butkienė, 2006).

Scrum metodo savybė rengti kasdienius susirinkimus nėra tinkama informacinių sistemų kūrimui, kadangi viskas turi vykti greitai. Todėl kasdieniniai susirinkimai nebūtini,

pakaktų susirinkimo po kiekvienos iteracijos nesklandumams išsiaiškinti. Be to, Scrum metode iteracijos trukmė nuo 2 iki 4 savaičių. Tai gal ilgas laikotarpis, ypač, jeigu kuriama nedaug funkcijų turinti informacinė sistema. Pirmai iteracijai reikėtų ilgesnio laiko, kažkur vienos savaitės, o kitos iteracijos turėtų būti iki 5 dienų.

Tad apibendrinus viską, galima daryti išvada, jog reikalingas naujas metodas informacinėms sistemoms kurti. Sudarysime naują Agile metodologiją, naudodamiesi kai kuriomis XP, Scrum, FDD bei ICONIX metodų savybėmis.

Naujasis metodas pasižymės tokiomis savybėmis:

XP metodo savybės:

- Testų rašymas prieš kodo kūrimą
- Programavimas poromis.
- Komunikavimas su klientais
- Komunikavimas su kitais programuotojais
- Kolektyvinė kodo nuosavybė
- Paprastas dizainas

FDD metodo savybės:

- Pirmiausia sukuriamas visos sistemos modelis
- Labai trumpos iteracijos (nuo 1 val. iki 2 sav.)

Scrum metodo savybės:

- Pastoviai identifikuoja trūkumus ir kliūtis kūrimo procese ir naudojamose priemonėse.
- Neatliktų darbų sąrašai
- Susirinkimas sprintui suplanuoti

ICONIX metodo savybės:

- .Automatinis būsenų diagramų generavimas
- Minimalus UML diagramų kiekis

Metodas, skirtas informacinėms sistemoms kurti, turi pasižymėti greitu kūrimu, nuolatiniu testavimu. Komandos nariai turi bendradarbiauti vienas su kitu, išsiaiškinti, ką kiekvienas daro ir sutarti dėl terminų ir pan. Pildyti neatliktų darbų sąrašus. Be to, turi būti įtrauktas užsakovas į visą procesą, nes gali kisti jo reikalavimai, tad komandos nariai galėtų

kuo greičiau sureaguoti į juos. Komandai iškilus klausimams ir neaiškumams, visada reikia kreiptis į užsakovą, kuris turėtų noriai bendradarbiauti su komandos nariais.

1.9 Analizės išvados

1. Agile požiūris labiau tinkamas projektams, kurie gali būti vykdomi ir finansuojami pagal iteracinį gyvavimo ciklą.
2. Išanalizavus sistemų kūrimo metodus, buvo pasirinktas Agile požiūris, nes Agile metodologija teigia, kad kuriant programinę įrangą yra efektyviau išsiaiškinami reikalavimai turint dalį veikiančios programos nei išsamiai dokumentuojant juos. Bendraujant gyvai su užsakovu yra geriausia komunikacijos priemonė, nes užsakovo reikalavimai gali kisti ir juos reikia įvertinti kuriant programinę įrangą trumpomis iteracijomis.
3. Išanalizavus Agile metodus, nustatyta, kad tinkamiausi metodai informacinių sistemų kūrime yra Scrum, XP bei FDD. Bet jie nėra visiškai tinkami naudoti, tokie, kokie jie yra, nes XP metodas skirtas nedidelėms programuotojų grupėms, FDD metodas neapima reikalavimų surinkimo bei testavimo; Scrum metodas siūlo rengti kasdieninius susirinkimus, kurie tik gaišintų laiką, nes pakaktų susirinkimo po kiekvienos iteracijos.
4. Norint pagerinti informacinių sistemų kūrimo procesą, reikia sukurti naują metodą paremtą XP, Scrum, FDD bei ICONIX metodų savybėmis:

    - Greitas kūrimas;
    - Visos sistemos modelio sukūrimas
    - Efektyvus reagavimas į reikalavimų pasikeitimus;
    - Trūkumų bei kliūčių identifikavimas
    - Nuolatinis bendradarbiavimas su užsakovu;
    - Programavimas poromis
    - Užtikrina bendravimą tarp komandos narių;
    - Naudojant UML diagramas:
        - Panaudojimo atvejų
        - Sekos diagrama.
    - Dėmesys funkcionalumui;
    - Kuo paprastesnis dizainas
    - Neatliktų darbų sąrašų tikrinimas

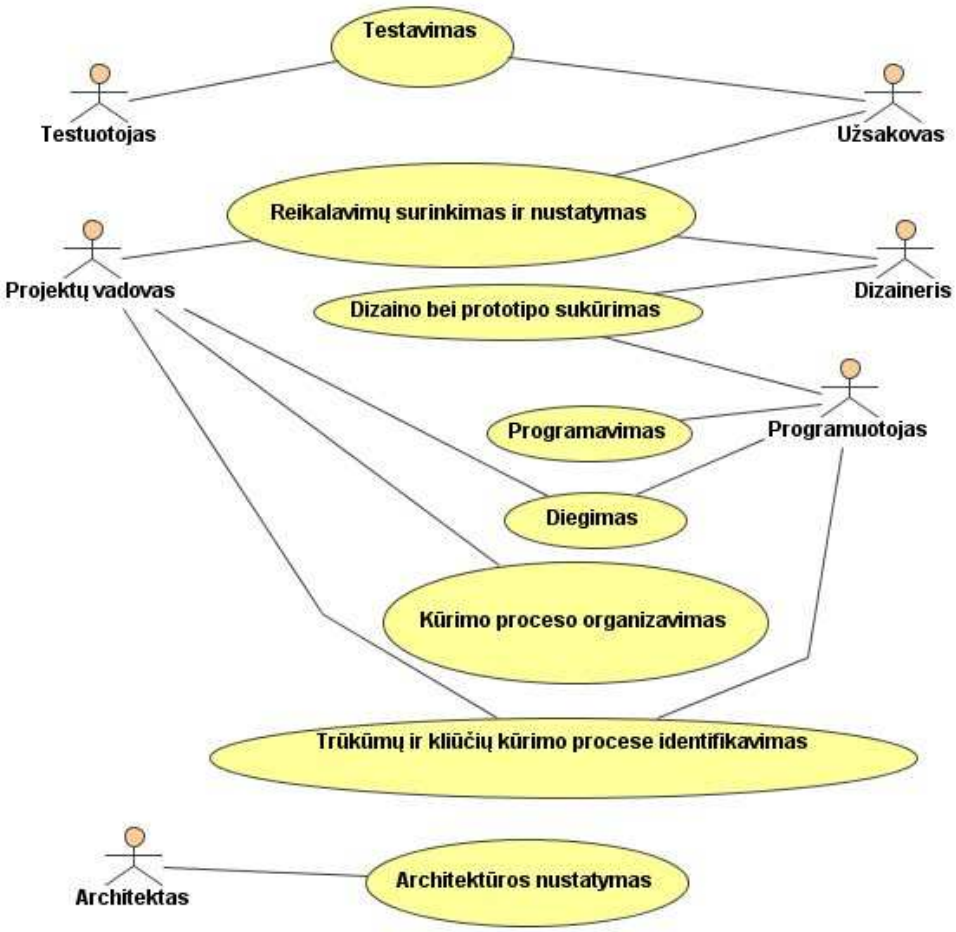
- Trumpos iteracijos;

2 METODAS, SKIRTAS INFORMACINĖMS SISTEMOS KURTI

Pagrindinės ir ne retai iškylančios problemos, kuriant programų sistemas yra tai, kad buvo pasirinkta netinkama kūrimo metodika. Tai susiję su tuo, kad patys užsakovai dažnai tiksliai nežino, kokios sistemos jiems reikės, todėl gali dažnai keisti reikalavimus kūrimo procese. Taigi mano manymu tinkamiausias tokiu atveju būtų Agile požiūris. Šis metodas greitai bei efektyviai leidžia reaguoti į pasikeitimus, bet tam reikalingas nenutrūkstamas komunikavimas su užsakovu.

Kuriant bet kokią programinę įranga yra aptariamos tokios temos:

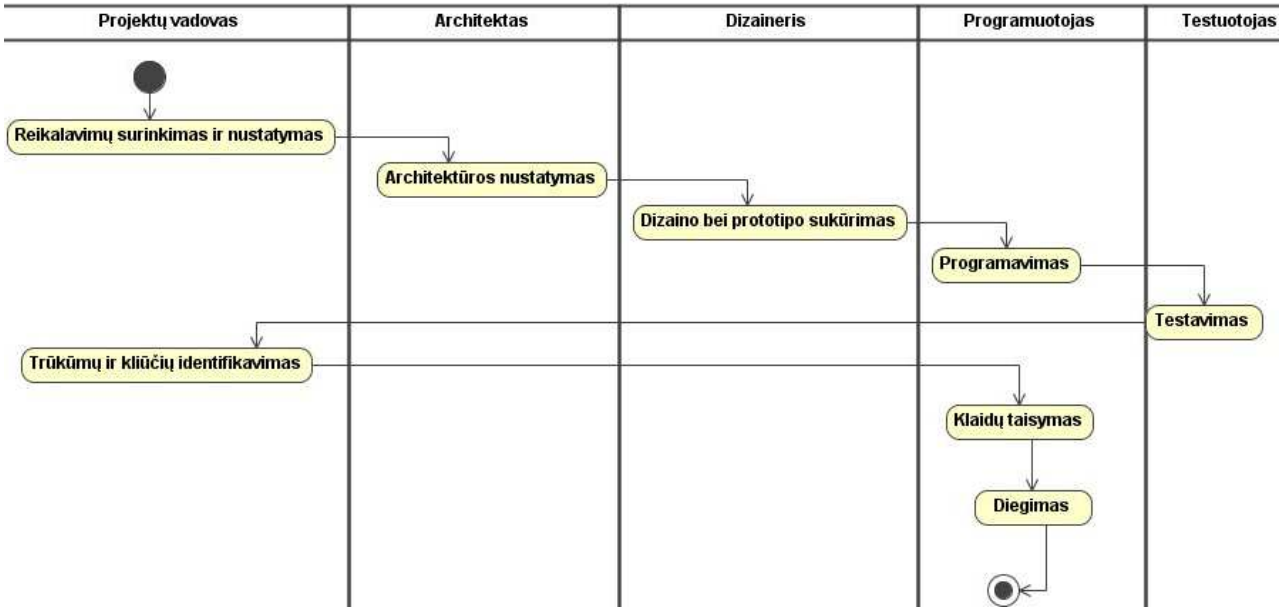
- Funkciniai ir nefunkciniai reikalavimai
- Vartotojo reikalavimai
- Sistemos reikalavimai
- Sistemos modeliavimas
- Programinės įrangos reikalavimų dokumentas



Pav. 16 Funkciniai metodo reikalavimai.

Šaltinis: sudaryta autoriaus.

Šis Use Case modelis yra apibendrintas, srautai nėra įvardinti, veiklos uždavinys neaprašytas. Šis modelis apibendrintai atskleidžia kaip yra kuriamos informacinės sistemos.



Pav. 17. veiklos diagrama.

Šaltinis: sudaryta autoriaus

Veiklos diagrama – procesų modeliavimo priemonė. Veiklos diagramoje labai aiškiai atskleidžiama, kokie procesai vyksta, kokia eiga, kas juos vykdo ir kokie dalyviai dalyvauja. Dalyviai yra šie:

- Projektų vadovas
- Architektas
- Dizaineris
- Programuotojas
- Testuotojas

Visų pirma užsakovas su projekto vadovu, nusistato reikalavimus kuriamai sistemai bei pateikia pirminį užsakymą. Programuotojas programuodamas identifikuoja trūkumus ir kliūtis ir jas derina su projektų vadovu. Atlikus programavimo darbus, reikia ištestuoti suskurtą programą. Šią funkciją atlieka testuotojas. Po testavimo atrastas klaidas ir trūkumus reikia ištaisyti. Programuotoju su projektų vadovu ištaiso klaidas. Galiausiai lieka išbaigtos ir ištestuotos programos įdiegimas. Tai atlieka projektų vadovas su programuotoju.

Nefunkciniai reikalavimai:

Norint naudoti šitą metodą, reikia, kad užsakovas visa laiką būtų greta, t.y. įtrauktas į kūrimo procesą. Geriausia yra, kai užsakovas paskiria vieną ar kelis žmones(būtinai ekspertus) dirbti kartu su kūrėjų kolektyvu. Vartotojų pasakojimus užrašo užsakovas su programuotojų pagalba. Užsakovas užtikrina, kad vartotojų pasakojimai padengia didžiąją dalį sistemos funkcionalumo. Užsakovas taip pat dalyvauja laidų planavimo susitikimuose ir išsako savo nuomonę, derasi dėl to, kokie vartotojų pasakojimai turi būti realizuoti kiekvienoje suplanuotoje laidoje. Taip pat aptariama, kada ir kokios laidos bus pristatomos klientui – svarbu tai padaryti kuo anksčiau, kad klientas kuo greičiau pradėtų dirbti su sistema ir laiku (kuo anksčiau) kolektyvas gautų atsiliepimus. Kuo didesnis projektas, tuo daugiau užsakovo laiko reikės. Užsakovas taip pat turės talkinti atliekant funkcinius testus, jų kūrime – reikės sukurti testinius duomenis, apskaičiuoti ir verifikuoti testų rezultatus. Gali atsitikti taip, kad sistema nepraeis visų funkcinių testų, tada užsakovas peržiūri testų rezultatus ir nusprendžia, ar išleisti sistemą ar ne. Atrodytų, kad sueikvojama daug užsakovo laiko, tačiau daug jo ir sutaupoma, nereikalaujant detalios reikalavimų specifikacijos ir vėliau nepristatant jam “blogos” sistemos.

Kitiems dalyviams ypatingų reikalavimų nėra keliama. Kiekvienas asmuo yra atsakingas už tam tikrus procesus. Pilnam programų kūrimo procesui užtikrinti, komandoje pilnai užtenka penkių pareigybių

2.1 Metodo, skirto informacinėms sistemoms kurti, projektas

Projekto tikslas – metodikos parengimas ir demonstravimas, kuriant informacines sistemas panaudojant Agile metodologiją bei ICONIX metodą..

Galima taikyti Agile metodologiją, nes užsakovas gali keisti savo reikalavimus.

2.1.1 Metodo reikalavimai

Metodas skirtas informacinių sistemų kūrimui, kai yra problemiška sritis, kuriose reikalavimai dažnai kinta. Klientai gali pradžioje gerai nežinoti, ką jų sistema turi daryti. Galima situacija, kai reikia sukurti sistemą, kurios funkcionalumas kinta kas kelias savaites. T.y. kai kuriose srityse dinamiškai besikeičiantys reikalavimai yra vienintelis pastovus dalykas.

Pirmiausia yra vykdomas tyrimas, t.y. išklausomi vartotojo pasakojimai apie pageidautinos kuriamos sistemos savybes. Jos užrašomos kortelėse, vartotojo pasakomai gali būti bet kada papildyti bei patikslinti. Tada sudaromas sistemos funkcionalumo apimtis. Bet kadangi reikalavimai gali pasikeisti, todėl sunku nustatyti išteklius ir laiko sąnaudas. Todėl kūrimo procesas tampa sudėtingas ir neorganizuotas, reikalaujantis lankstumo, kad būtų tinkamai sureaguota į pasikeitimas, kad sukurti tinkamą sistemą. Tam yra naudojami kliūčių ir trūkumų identifikavimo veiklos kūrimo procese eigoje.

Metodo principai:

- Būtinas aktyvus vartotojo įtraukimas;
- Komandos vieningumas
- Komanda turi būti įgalinta priimti sprendimus;
- Programavimas porose
- Pirmiausia sukuriamas pradinis produktas, o vėliau daromi pataisymai, pridedamos naujos savybės;
- Mažesnis dėmesys dizainui;
- Pastovus integravimas
- Rėmėjai, kūrėjai ir vartotojai turi sugebėti išlaikyti pastovu tempą.
- Iteratyvus ir priauginantis kūrimo procesas yra būtinas siekiant tikslaus sprendimo;
- Testavimas integruotas kiekvienoje iteracijoje;
- Darbas ne daugiau 40 valandų per savaitę;
- Bendravimas ir bendradarbiavimas visų projekto dalyvių yra esminis.

Metodo savybės:

- Testų rašymas prieš kodo kūrimą
- Programavimas poromis.
- Komunikavimas su klientais
- Komunikavimas su kitais programuotojais
- Paprastas dizainas
- Pirmiausia sukuriamas visos sistemos modelis
- Labai trumpos iteracijos (nuo 1 val. iki 2 sav.)
- Pastoviai identifikuoja trūkumus ir kliūtis kūrimo procese ir naudojamose priemonėse.
- Neatliktų darbų sąrašai

- Susirinkimas sprintui suplanuoti
- Automatinis būsenų diagramų generavimas
- Minimalus UML diagramų kiekis

Komandos dydis priklauso nuo projekto dydžio. Gali būti nuo kelių narių iki keliasdešimt. Komandą sudaro programuotojas, dizaineris, architektas, testuotojas. Praktika rodo, jog kolektyvinė kodo nuosavybė yra patikimesnė, nei atsakomybės už atskiras klases paskirstymas atskiriems žmonėms. Ypač atsižvelgiant į tai, jog tie žmonės gali bet kada palikti projektą.

Kadangi testai yra automatizuoti, komandos nariams ir užsakovui nereikia atskirai atlikinėti testavimo. Aišku testai rašomi su užsakovo pagalba, tad yra įtraukiamas ir užsakovas į komandos sudėtį.

Komandoje dalyvauja tokie:

- Užsakovas
  - užsakovas nusprendžia, kokius vartotojo pasakojimus reikia realizuoti pirmiausia
  - užtikrina, kad vartotojų pasakojimai padengia didžiąją dalį sistemos
  - padeda atliekant funkcinius testus, jų kūrime – sukuria testinius duomenis, apskaičiuoja ir verifikuoja testų rezultatus
- Komanda
  - atsakinga už sprendimus dėl būtinų veiksmų ir organizavimosi siekiant tikslų. Testuoja, kuria kuo paprastesnį bei aiškesnį programos kodą, programos dizainą. Komandos nariai turi bendrauti tarpusavyje.
  - Komandą sudaro:
    - Dizaineris;
    - Architektas;
    - Programuotojas;
- Vadovas
  - priima sprendimus, atsakingas už projektą, valdymą, neatliktų darbų sąrašo kontrolę ir pateikimą. Turi kurti ir palaikyti viziją, uždegti komandą, skatinti komandinį darbą, šalinti progresui trukdančias kliūtis. Bendrauja su projekto komanda, dalyvauja nustatant tikslus ir reikalavimus;

Kuriant nedaug funkcijų turinčią sistemą, nebūtinai turi būti žmogus kiekvienam vaidmeniui, vienas žmogus gali turėti keletą vaidmenų.

Komandos nariai tarpusavyje turi nuolatos bendrauti (jei komandą sudaro daugiau nei vienas narys). Bendravimas gali vykti internetu, telefonu, tačiau bent kartą per savaitę turi būti suorganizuotas bendras susitikimas.

18 lentelė

Vaidmenų atsakomybės

Vaidmuo	Atsakomybė
Užsakovas	Pateikia sistemos reikalavimus, dalyvauja užduotyse susijusiose su produkto kūrimo darbais: testuoja sistemą, praneša apie testavimo rezultatus vadovui, nusprendžia kada reikalavimai yra patenkinti, suteikia reikalavimams prioritetus.
Dizaineris	Atsakingas už sistemos dizainą. Dizainas turi būti paprastas. Jei pamatome, kad kažkas yra labai sudėtingo, tada reikia pakeisti į kuo paprastesnį būdą.
Architektas	Tai nėra vieno žmogaus pareigybė. Už architektūra yra atsakinga visa komanda.
Programuotojas	kiekvienas programuotojas rašo modulių testus savo kuriamam kodui. Visas išleidžiamas kodas turi modulių testus. Visas kodas padengtas automatizuotais testais. Šis testų rinkinys užtikrina, kad kitų programuotojų pakeitimai nepastebimai nesugriaus sistemos.
Vadovas	Priima sprendimus, atsakingas už projektą, valdymą, neatliktų darbų sąrašo kontrolę ir pateikimą. Turi kurti ir palaikyti viziją, uždegti komandą, skatinti komandinį darbą, šalinti progresui trukdančias kliūtis. Bendrauja su projekto komanda, dalyvauja nustatant tikslus ir reikalavimus;

Šaltinis: sudaryta autoriaus

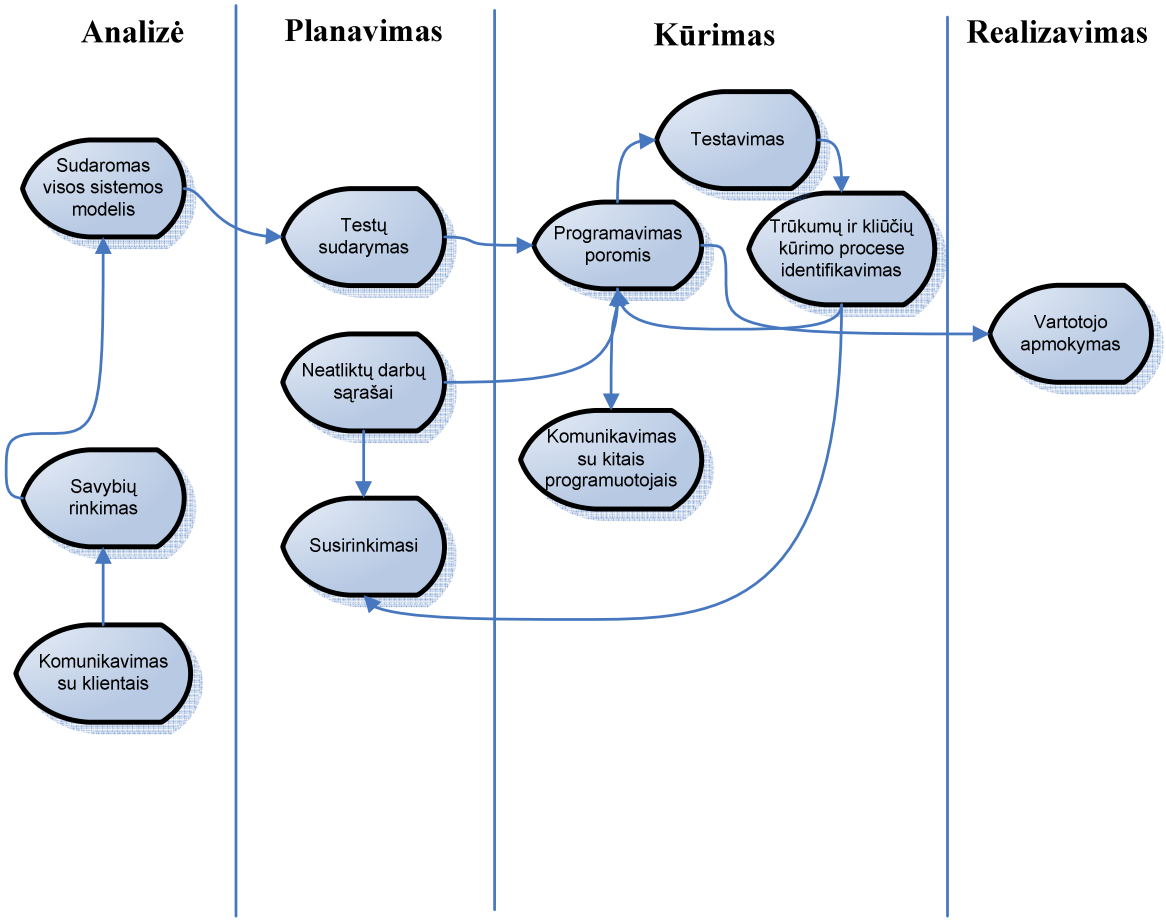
2.2 Kūrimo procesas

Kūrimo procesas susideda iš šių fazių:

- Analizės
- Planavimo
- Kūrimo
- Realizavimo

Kūrimo procesas prasideda nuo savybių surinkimo, komunikuojant su klientais bei sudarant visos sistemos medelis. Tokiu būdu identifikuojami reikalavimai kuriamai sistemai. Reikalavimai nėra užrašomi formaliai. Svarbiausia yra suprantamai apibrėžti reikalavimus.

Planavimo fazėje yra sudaromi testai.



Pav. 18. Projekto vykdymo schema

Šaltinis: sudaryta autoriaus

2.2.1.1 Analizė

Analizės metu vykstantys procesai pateikti 18 pav.

Pav. 19. Analizės metu vykstantys procesai

Šaltinis: sudaryta autoriaus

Komunikuojant su klientais užsakovas užrašo tai, ką turi daryti sistema. Kliento pasakojimų formatas – maždaug trys sakiniai, užrašyti dalykinės srities terminologija be techninio žargono. Šie kliento pasakojimai paskui transformuojami į savybių rinkinius. Vartotojo pasakojimai taip pat naudojami kuriant priėmimo testus, tam, kad įsitikintume, jog vartotojo pasakojimas teisingai realizuotas. Didžiausias skirtumas tarp kliento pasakojimų ir reikalavimų specifikacijų – detalumo lygis. Klientas pasakojimas turi kuo detaliau nupasakoti, ką turi daryti sistema, kuo tikslesni savybių rinkiniai bus suformuoti, tuo mažiau laiko užtruks

visas programos kūrimo procesas. Kitas skirtumas tarp reikalavimų specifikacijos ir kliento pasakojimų yra dėmesys kliento poreikiams.

Analizės metu išsiaiškinami šie dalykai:

- Informacinės sistemos funkcijos
    - Duomenų bazė
    - Vartotojai ir sistemos funkcijos
- Vartotojo sąsaja
    - Informacinės sistemos struktūra
- Nefunkciniai reikalavimai

2.2.1.2 Informacinės sistemos funkcijos

Surinkus iš kliento reikalavimus nubraižomos šios diagramos: panaudojimo atvejų diagrama, klasių diagrama.

2.2.1.3 Vartotojai

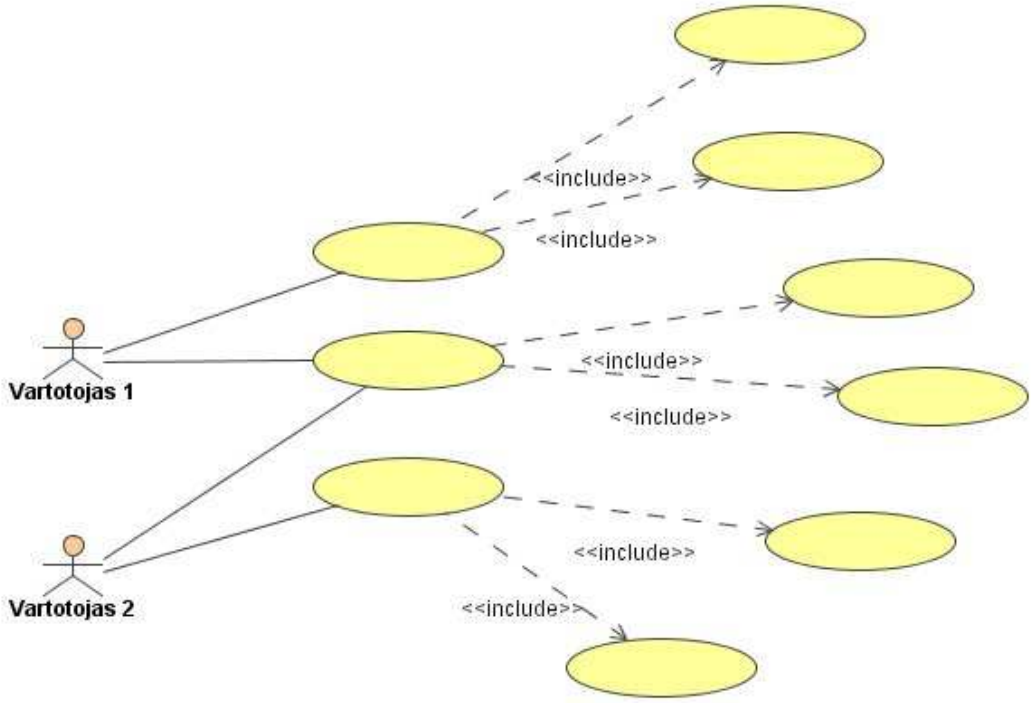
Klientams užduodami šie klausimai, kad išsiaiškintume reikalavimus kuriamai informacinei sistemai. Rezultate nubraižoma panaudojimo atvejų diagrama.

- Kokie bus sistemos vartotojai?
- Kokias funkcijas atliks vartotojai?
- Prie kokių duomenų galės prieiti kiekviena vartotojų grupė?
- Kokios papildomos sistemos funkcijos?

Vartotojui gali būti pasiūlytos informacinei sistemai būdingos tipinės sistemos funkcijos:

- Vartotojų prisijungimas;
- Slaptažodžio priminimas;
- Paieškos funkcijos pateikimas;
- Kontaktų pateikimas;
- Kt.

Gauta informacija užrašoma tekstu.



Pav. 20. Panaudojimo atvejų modelis

Šaltinis: sudaryta autoriaus

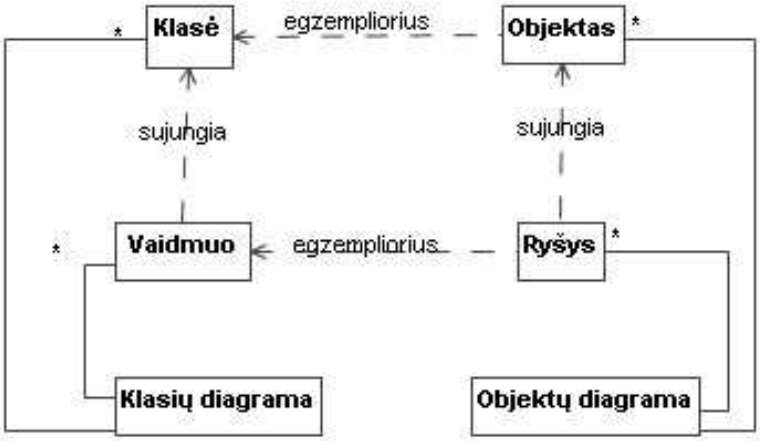
Ši diagrama parodo elgseną iš vartotojų pozicijų: aprašomos funkcijos ir kas už jas atsakingi. Aprašanti ką projektuojama sistema gali atlikti, kartu aprašydama ir išorinius sistemos veikėjus (aktorius). Pagrindinis šios diagramos elementas – panaudos atvejis, aprašantis aibę panašių sąveikos scenarijų. Kiekvienas panaudos atvejis paprastai turi vieną pagrindinį ir keletą šalutinių scenarijų, Panaudojimo atvejų diagramos lengvai suprantamos ir “nespecialistams”, todėl vartotojai gali būti įtraukiami į programos kūrimą nuo pat pradžių. Lieka mažiau perdarymų pabaigoje. Leidžia kurti ir įdiegti naujas IT sistemas greičiau ir pigiau nei konkurentai. (UAB "Baltijos programinė įranga", 2008)

Jeigu sistema atlieka nedaug funkcijų ir panaudos atvejis yra aiškus, tai galima ir nebraižant panaudos atvejų diagramos, užtenka ir trumpai aprašyti.

2.2.1.4 Duomenų bazė

Renkant reikalavimus iš vartotojų, kurie papasakoja, kokias funkcijas turi sistema atlikti ir kokius laukus jie nori matyti programos lange, programuotojai pagal tai braižo klasių diagrama. Klasės diagrama yra duomenų bazių pagrindas, kuri gali pasikeisti kintant reikalavimams. Klasių diagrama specifikuoja statinę objektų struktūrą, kuri apima atributus, operacijas ir ryšius tarp jų. Ši diagrama yra naudojama pačių programuotojų, todėl vartotojams ji gali būti ir nerodoma. Iš klasių diagramos automatiškai yra generuojama duomenų bazė. Tai palengvina programuotojų darbą, nes nereikia antrą kartą sudaryti

duomenų bazės schemos. Jeigu duomenų bazės schema braižoma ant popieriaus, tai po to vis tiek reikia viską suvedinėt programuojant.



Pav. 21. Klasių diagramos schema

Šaltinis: Vikipedija (2007) http://lt.wikipedia.org/wiki/Vaizdas:KlasiuDiagramosMetamodelis.jpg

Klasė yra vaizduojama stačiakampiu, padalintu į tris dalis. Pirmoje skiltyje yra rašomas klasės vardas, antrojoje atributai – savybės, apibūdinančios klasę ir trečiojoje dalyje pateikiamos operacijos – veiksmai, kurios galima taikyti klasės objektui.(Vikipedija, 2007)

2.2.1.5 Vartotojo sąsaja

Vartotojo sąsaja – tai nefunkcinis reikalavimas. Tai įrankių valdymo paletė, kuri atskiriems vartotojams atrodo nevienodai. Galingose sistemose yra galimybė kiekvienam vartotojui turėti savo unikalias valdymo paletes. Šiuo atveju vartotojo sąsajos modelis yra projektuojamas pagal vartotojo reikalavimus

2.2.1.6 Informacinės sistemos struktūra

Informacinės sistemos struktūra pateikia programos meniu, t.y. kaip vartotojas gali pasiekti reikalingas programos formas. Informacinės sistemos struktūrą gali pateikti įvairiais būdais: formalia UML diagrama arba neformaliai nubraižyta schema.

2.2.2 Nefunkciniai reikalavimai ir apribojimai

Nefunkciniai reikalavimai apima:

- Reikalavimai sąsajai:
  - ✓ vartotojo sąsaja,
  - ✓ grafinė vartotojo sąsaja,

- ✓ diagnostika,
- ✓ programinės įrangos sąsaja
- ✓ taikomųjų programų sąsaja.

- Sistemos darbo reikalavimai
- Projekto apribojimai:
  - ✓ standartų, kurių reikia laikytis
  - ✓ Apribojimai techninei įrangai
  - ✓ Grafiniai duomenys
- Kiti nefunkciniai sistemos apribojimai:
  - ✓ Saugumas (ugniasienė, slaptažodžiai)
  - ✓ IS išplėtimo reikalavimai – galimybė prijungti naujus įrenginius
  - ✓ Taikomųjų programų suderinamumas - pvz., suderinamumas su MS programine įranga
  - ✓ Reikalavimai servisui - per tam tikrą laiko tarpą turi būti atliekamas sistemos saugumo ir stabilumo patikrinimas

Nefunkciniai reikalavimai užrašomi tekstu.

2.3 Planavimas

Pav. 22. Planavimo metu vykstantys procesai

Šaltinis: sudaryta autoriaus

2.3.1 Testų sudarymas

Testas kuriamas kiekvienam moduliui. Pirma yra sukuriamas arba parsisiunčiamas modulių testų karkasas/infrastruktūra, kad galima būtų kurti automatizuotus modulių testų rinkinius. Turi būti ištestuotos visos klasės sistemoje. Iš pradžių turi būti kuriami testai, o paskui kodas. Kodas yra išleidžiamas kartu su testais. Be to, testai turi būti rašomi iš karto, o ne projekto pabaigoje. Unit testai apsaugo kodą nuo netyčinio sugadinimo. Norint išleisti kodą, reikia, kad testavimas būtų teigiamas, t.y. jokių klaidų. Po kiekvieno pakeitimo, modulių testai užtikrina, kad pakeitimas struktūroje nepakeitė funkcionalumo. Integruojant paskutinius pakeitimus, praleidžiamas testų rinkinys. Be to, yra labai retas dalykas, kad klaida bus ir kode, ir teste. Kartais atsitinka, kad kodas geras, o testas blogas.[15]

Jei randama klaida, kuriami testai. Testai kuriami, norint apsisaugoti nuo tos klaidos, kad ji nepasikartotų. Klaida produkcijoje reikalauja priėmimo testų sukūrimo. Funkcinio testo sukūrimas prieš sprendžiant problemą padeda užsakovui trumpai ją apibrėžti ir pateikti programuotojams. Jei priėmimo testas nepraėjo, programuotojai gali sukurti modulių testus, kad pamatytų defektą iš savo, t.y. „source code“ pusės. Kai unit testai praeina 100%, tada galima vėl paleisti priėmimo testus.

Priėmimo testai praleidžiami dažnai, fiksuojami ir peržiūrimi jų rezultatai. Priėmimo testai kuriami remiantis vartotojų pasakojimais. Iteracijos eigoje realizuojami vartotojų pasakojimai verčiami į priėmimo testus. Užsakovas specifikuoja testavimo scenarijų, kai vartotojų pasakojimai yra korektiškai įgyvendinti. Vienas pasakojimas gali turėti vieną ar kelis priėmimo testus. Priėmimo testai tai sistemos juodosios dėžės testai. Kiekvienas toks testas atitinka kažkokį laukiamą sistemos darbo rezultatą. Užsakovas yra atsakingas už priėmimo testų korektiškumo verifikavimą ir testų rezultatų peržiūrą, bei nepraėjusių testų taisymo prioritetus. Vartotojo pasakojimas laikomas neišbaigtu, kol jis nepraėjo savo priėmimo testų. Priėmimo testai turi būti automatizuoti, kad juos galima būtų dažnai leisti. Jų rezultatai yra renkami ir komunikuojami kolektyvui. Kūrėjai turi išskirti laiko kiekvienoje iteracijoje, kad pataisytų nepraėjusių testų klaidas. Anksčiau priėmimo testai buvo vadinami funkciniais testais. Pavadinimas pakeistas, nes jis geriau atspindi jų prasmę – užsakovo reikalavimai yra atlikti ir sistema yra priimtina. [15]

2.3.2 Neatliktų darbų sąrašas

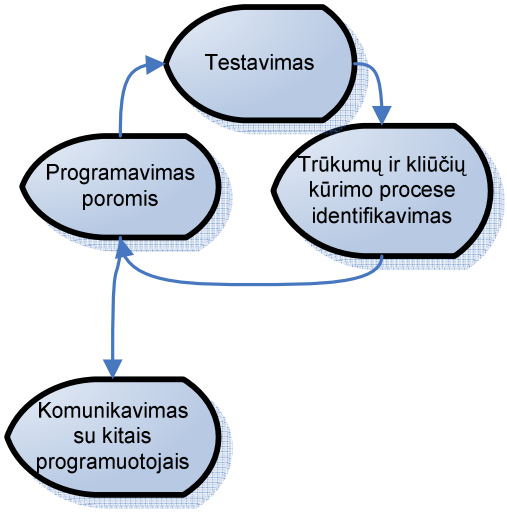
. Neatliktų darbų sąrašas yra pastoviai atnaujinamas su naujais ir išsamesniais dalykais, taip pat su tikslesniu įvertinimu bei naujais prioritetais. Planavimas apima projektinės komandos apibrėžimą, įrankius ir kitus išteklius. Kiekvienoje iteracijoje komanda apžvelgia atnaujinto produkto neatliktus darbus, kad įsipareigotų kitam kartojimui..

Sistemos projektas, apimdamas ir architektūra, yra pagrįsta einamaisiais dalykais neatliktų darbų sąraše. Perkėlimo į egzistuojančią sistemą atveju, pakeitimai būtini, kad neįvykdytų darbų sąrašas yra identifikuojamas su problemomis, kurias jie gali sukelti. Projekto peržiūros susirinkimas reikalingas tam, kad peržiūrėti ištesėjimo pasiūlymus ir kad priimamas sprendimai paremti šio susirinkimo pagrindu.

2.3.3 Susirinkimas

Susirinkimo metu yra aptariami neatliktų darbų sąrašai. Susirinkimai yra organizuojami priklausomai nuo projekto dydžio, klientų poreikio ir pan. Juo metu gali būti išsiaiškinami pakitę reikalavimai, perskirstomi darbai. Susirinkimai gali vykti realioje patalpoje, arba gali vykti nuotoliniu būdu, naudojant e-konferencijas, jei klientas, ar kuris nors kitas komandos narys yra išvykęs, ar negali atvykti. Susirinkime turėtų dalyvauti visi nariai, t.y. vadovas, visa komanda ir klientas.

2.4 Kūrimas



Pav. 23. Kūrimo metu vykstantys procesai

Šaltinis: sudaryta autoriaus

2.4.1 Programavimas poromis

Visas išleidžiamas kodas yra kuriamas dviejų žmonių prie vieno kompiuterio. Programavimas poromis padidina programinės įrangos kokybę, neatsiliepdamas pristatymo laikui. Esant didesnei kokybei, vėliau projekte sutaupoma daugiau pinigų. Geriausias būdas rašyti kodą poromis yra tiesiog sėdėti abiem prieš monitorių, tai vienam, tai kitam rašyti kodą. Vienas žmogus renka ir mąsto taktiškai apie kuriamą metodą, o kitas mąsto strategiškai kaip tas metodas tinka šiai klasei.

Vienu metu tik viena pora programuotojų integruoja kodą. Programuotojai ištestuoja savo kodą ir integruoja jį, tikėdami, kad viskas yra gerai. Bet dėl lygiagretaus integravimo galima kombinacija išeities tekstų, kurie buvo testuojami atskirai, bet kartu ne. Kyla integravimo problemos. Dar didesnės problemos kyla, kai neįmanoma nustatyti paskutinės versijos. Tas liečia ne tik kodą, bet ir testų rinkinį. Yra daug šios problemos sprendimų, bet

nei vienas neišsprendžia iki galo. Norisi, kad programuotojai galėtų dirbti lygiagrečiai, drąsiai darydami pakeitimus bet kuriai sistemos daliai, kada reikia, bet taip pat norisi, kad visa tai vyktų be klaidų. Taigi integravimas turi būti nuoseklus – kodas yra integruojamas, testuojamas. Tokiu būdu paskutinė versija yra visada apibrėžta. Čia reikia tam tikro užrakto mechanizmo. Paprasčiausias dalykas – turėti tam tikrą fizinį daiktą ir perdavinėti, pvz., kompiuterį. Kitais žodžiais tariant, kūrimas daromas prie vieno kompiuterio, o paskui programuotojai eina prie specialiai integravimui pastatytos mašinos ir ten daro integraciją. [15]

2.4.2 Trūkumų ir kliūčių kūrimo procese identifikavimas

Šiame etape yra identifikuojami trūkumai ir jie aptariami susirinkimo metu. Stengiamasi kuo greičiau pašalinti kliūtis.

2.4.3 Komunikavimas su kitais programuotojais

Kadangi programos kodas yra visiems prieinamas, todėl yra labai svarbus komunikavimas tarp programuotojų. Kiekvienas programuojas žino, ką daro jo kolega ir bet kada gali pakeisti jo sudarytą kodą. O, kad nepridarytų klaidų visada yra testuojama ir ieškoma netikslumų. Visas išleidžiamas kodas turi modulių testus. Visas kodas padengtas automatizuotais testais. Šis testų rinkinys užtikrina, kad kitų programuotojų pakeitimai nepastebimai nesugriaus sistemos. Praktika rodo, jog kolektyvinė kodo nuosavybė yra patikimesnė, nei atsakomybės už atskiras klases paskirstymas atskiriems žmonėms. Ypač atsižvelgiant į tai, jog tie žmonės gali bet kada palikti projektą. [15]

2.4.4 Iteracinis kūrimas

Agile veiklos modeliuotojai turėtų dirbti iteraciškai. Tai reiškia, jog sukuriamas paprastas modelis ir jis naudojamas, kol adekvačiai aprašo modeliuojamą procesą, o susidarius naujai situacijai, modelis modifikuojamas, ir taip atliekamas tiek kartų kiek reikia t.y. kol pasiekiamas optimalus ir klientą tenkinantis rezultatas. Yra įprasta sukurti keletą tos pačios srities modelių. Tai leidžia suvokti problemą skirtingais pjūviais ir atsiradus pakeitimams, juos įvykdyti tinkamoje vietoje.

Darbas atliekamas mažais žingsneliais, modeliuojama tiek kiek reikia artimiausiai ateičiai, negalvojant, kokius reikalavimus ar pakeitimus užsakovas pateiks po mėnesio t.y. sprendžiamos šios dienos problemos. Toks požiūris suteikia labai didelį lankstumą ir reagavimo į pokyčius greitį. Savo ruožtu tai gali sukelti kontroliavimo ir koordinavimo

problemų, ypač jeigu probleminė sritis yra plati (sudėtinga organizacijos verslo procesų struktūra). Be to, toks požiūris atrodo trumparegiškas ir visada egzistuoja pavojus nepastebėti kažko svarbaus - „jeigu važiuodamas dviračiu žiūrėsi tik po ratais, nepastebėsi posūkio už 10 metrų“.

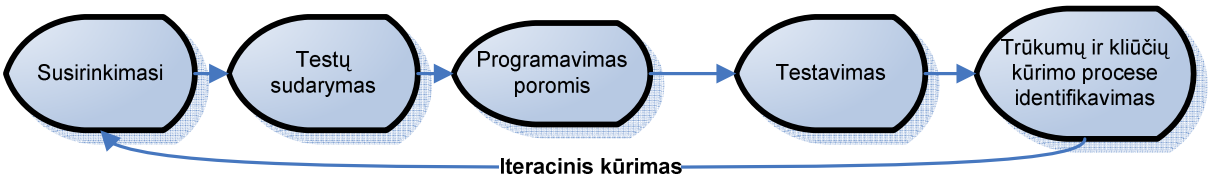
Pirmasis etapas yra vizijos ir preliminarių modelių sukūrimas. Būtent preliminarių, kadangi modeliai turi būti pakankamai geri ir leistų eiti pirmyn, tačiau jie neturi būti idealūs.

Iteracinio kūrimo metu:

- Vykdomi susirinkimai, kurių metu aptariami būsimi darbai, neatliktų darbų sąrašai.
- Kuriama/keičiama informacinės sistemos struktūra;
- Rašomas programos kodas;
- Duomenų bazės kūrimas.

Iteracijos trukmė priklauso nuo sistemos funkcijų skaičiaus. Atsižvelgiant į tą skaičių, vadovas nustato iteracijos trukmę. Iteracijų trukmė gali kisti. Tai priklauso nuo atliktų darbų, bei, kiek liko padaryti. Iteracijos trukmė gali svyruoti nuo keleto valandų iki 1-2 savaičių

Kūrimo proceso iteracija pavaizduota 23 paveiksle



Pav. 24. Kūrimo proceso iteracija

Šaltinis: sudaryta autoriaus

Kiekvienos iteracijos pradžioje organizuojami susirinkimai. Jų metu yra aptariami neatliktų darbų sąrašai, suplanuojama kita iteracija. Toliau yra sudaromi nauji testai, kurie testuoja naujai kuriamą kodą. Pirma parašius testus, paskui bus lengviau parašyti kodą. Modulių testų kūrimas padeda programuotojams suprasti, ką iš tikrųjų reikia padaryti. Paprastai ne visada yra aišku, kada programuotojas užbaigė darbą, įgyvendino visą būtiną funkcionalumą. O parašius testus tai tampa aišku – kada visi testai praeina. Kitas pliusas – testai, parašyti prieš rašant kodą, padeda palaikyti paprastesnį sistemos dizainą. Paprastai sudėtinga pratestuoti (unit test) kai kurias sistemas – tokios sistemos dažniausiai užprogramuojamos, o paskui stengiamasi jas testuoti ir testuoja visai kiti žmonės.

2.4.5 Užsakovo įtraukimas

Viso kūrimo metu užsakovas turi bendradarbiaut su komandos nariais. Jei iškyla kažkokių neaiškumų, rengiamas susirinkimas, kuriame dalyvauja ir užsakovas. Jeigu užsakovas negali dalyvauti susirinkime, su juo galima susisiekti telefonu, el. Paštu, ar kitais būdais. Bet jis turi būti pasiekiamas bent jau vienu iš išvardintų būdų.

2.5 Realizavimas

Pav. 25. Realizavimo metu vykstantys procesai

Šaltinis: sudaryta autoriaus

Kai sistema yra pilnai ištestuota ir neaptikta jokių problemų bei užsakovas nepateikia naujų reikalavimų, tada atiduodamas galutinis produktas užsakovui. Prieš atiduodant programą, yra atliekamas vartotojo apmokymas, vartotojui yra parodoma, kaip dirbti su programa, kokia funkcijas ji atlieka. Užsakovui pageidavus, galima sudaryti programos aprašą.

3 METODO, SKIRTO INFORMACINĖMS SISTEMOMS KURTI EKSPERIMENTINIS TYRIMAS

Šiame skyriuje bus atliekamas bandomasis tyrimas kuriamam metodui. Šio tyrimo metu bus projektuojama informacinė sistema įmonei „Kemira-Lifosa“. Tyrimo metu bus tikrinama, ar metodas yra efektyvus informacinių sistemų kūrime.

3.1 Eksperimentinio tyrimo projektas

Šiame skyriuje pateikiama, kaip yra surenkami reikalavimai kuriamai sistemai, pritaikant naująjį metodą.

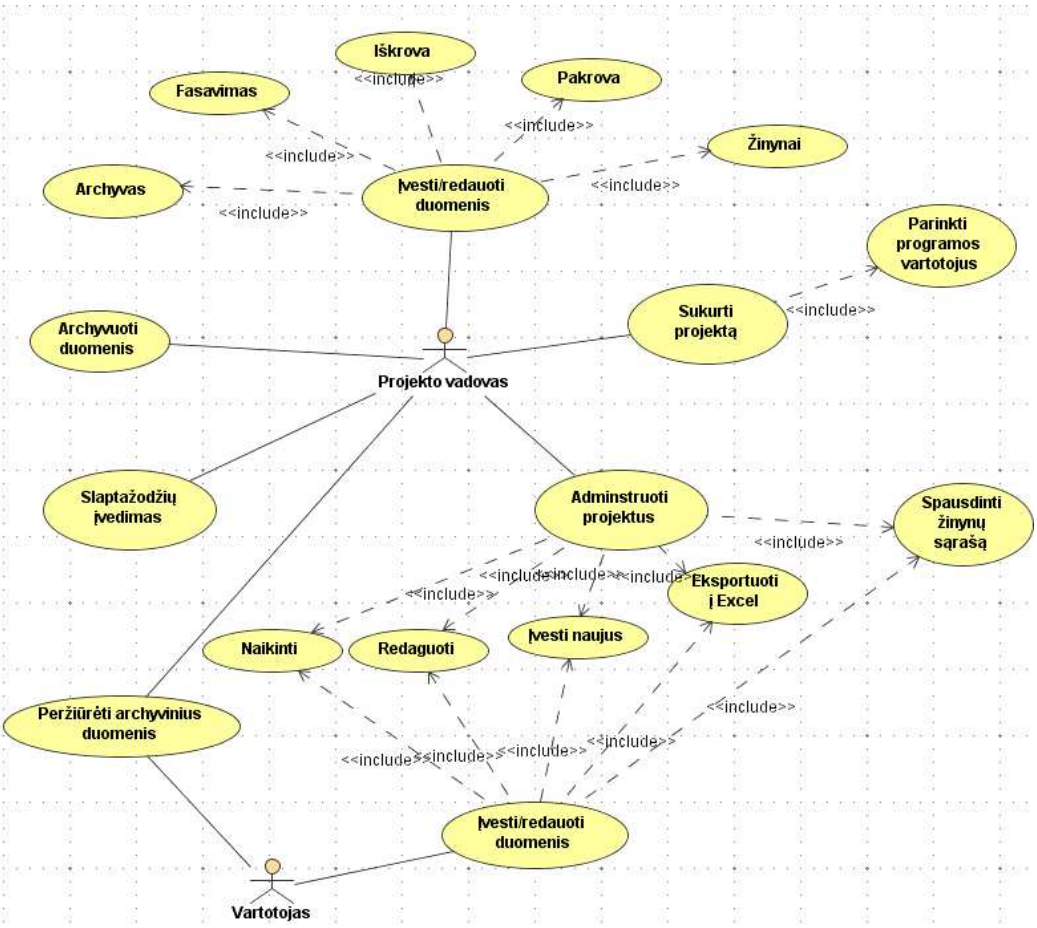
3.1.1 Vartotojai ir sistemos funkcijos

Sistemoje išskiriamos dvi vartotojų kategorijos:

- vartotojas;
- vadovas/administratorius;

Kiekviena vartotojų grupė turi skirtingas teises ir apribojimus.

Sistemos panaudos atvejų modelis pavaizduotas 26 paveiksle.



Pav. 26. Panaudos atvejų diagrama

Šaltinis: sudaryta autoriaus

Vartotojų panaudojimo atvejų modelyje vaizduojamos operacijos, kurias atliks sistemos vartotojai. Šis modelis vaizduoja sistemą vartotojų atžvilgiu.

Projekto vartotojai – projektų vadovas: asmuo, užimantis aukštesnes pareigas, kuriam suteikiama administratoriaus teisės, t.y. gali daryti viską; Vartotojas – įmonės darbuotojai, turintys teisę prieiti prie tam tikrų sistemos duomenų.

Projekto vadovo galimybės:

- Slaptažodžių įvedimas
- Archyvinių duomenų peržiūrėjimas
- Naujo projekto sukūrimas
- Įvesti/redaguoti duomenis
- Parinkti programos vartotojus
- Administruoti projektus
- Archyvuoti duomenis

Vartotojo galimybės

- Peržiūrėti archyvinius duomenis
- Įvesti/redaguoti duomenis

Vartotojų registracija

Naujus vartotojus gali registruoti projekto vadovas/administratorius. Registruojant naują vartotoją yra įvedamas jo vardas, pavardė, pareigos, telefono nr., ir jeigu reikia yra galimybė pridėti vartotojo nuotrauką. Naujam vartotojui suteikiamas prisijungimo vardas, t.y. trys pirmos vardo raides ir trys pavardes raides. Slaptažodis sukuriamas administratoriaus ir be administratoriaus niekas jo negales pakeisti.

Norint užregistruoti naują vartotoją, būtina užpildyti visus laukus, nes kitaip negalima bus išsaugoti.

Projektų vadovas/administratorius norėdamas užregistruoti naują vartotoją turi užpildyti formą, pateiktą 19 lentelėje.

19 lentelė

Vartotojų registracijos pavyzdys

Vardas	*Vardenis*
Pavardė	*Pavardenis*
Prisijungimo vardas	*varpar*
Slaptažodis	*******
Pareigos	*krovikas*
Telefono nr.	*538468*
Nuotrauka	....

Šaltinis: sudaryta autoriaus

Žemiau parodyta kaip vyksta duomenų archyvavimas. Kaip turėtų reaguoti sistema, jei pateikiami netikslūs duomenys, kas gali archyvuoti duomenis.

20 lentelė

Specifikacijos panaudojimo atvejis archyvuoti duomenis

Panaudojimo atvejis	**Archyvuoti duomenis**
Aktorius	Projekto vadovas
Sistema	Vagonų ir trąšų fasavimo IS
Prieš sąlyga	Darbuotojas turi būti registruotas duomenų bazėje
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Iš meniu pasirenkami duomenys, kuriuos norime archyvuoti 2. Įvedama laikotarpis, kurio duomenis norime archyvuoti. 3. Paspaudžiamas pasirinkimo patvirtinimą užtikrinantis mygtukas.	
Po sąlyga	
Alternatyvos (nesėkmės atvejai)	Archyvavimas gali būti nevykdomas, jei bus nurodyta neteisingas laikotarpis.
Vykdymo variantai	Duomenis, kuriuos nori archyvuoti, vartotojas pasirenka iš IS “Duomenų tvarkymas” meniu punkto.
Veiklos taisyklės	Vartotojas turi įvesti laikotarpį pagal kurį bus archyvuojami duomenys.

Šaltinis: sudaryta autoriaus

Žemiau pateikta, kaip vyksta naujų duomenų įvedimas bei redagavimas. Kaip turėtų reaguoti sistema, jei nauji duomenys yra neteisingi, kas ir kokius duomenis gali įvesti bei redaguoti

21 lentelė

Specifikacijos panaudojimo atvejis įvesti/redaguoti duomenis

Panaudojimo atvejis	**Įvesti / Redaguoti duomenis**
Aktorius	Visi
Sistema	Vagonų ir trąšų fasavimo IS
Prieš sąlyga	Darbuotojas turi būti registruotas duomenų bazėje
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Iš meniu pasirenkamas norimos įvesti/koreguoti informacijos punktas. 2. Užpildoma forma 3. Spaudžiamas patvirtinimo mygtukas	3.1. Sistema tikrina ar įvesti/koreguoti duomenys yra teisingi
Po sąlyga	Nauji duomenys įrašomi į DB ir gaunamas pranešimas apie sėkmingą operaciją
Alternatyvos (nesėkmės atvejai)	1. Įvesti duomenys gali būti nekorektiški 2. Bandomas dubliuoti unikalus duomenų tipas
Vykdymo variantai	Duomenys, kuriuos norime įvesti arba redaguoti, vartotojas pasirenka iš IS meniu punkto.
Veiklos taisyklės	

Šaltinis: sudaryta autoriaus

Šioje lentelėje yra pavaizduota kaip turėtų vykti projektų administravimas.

22 lentelė

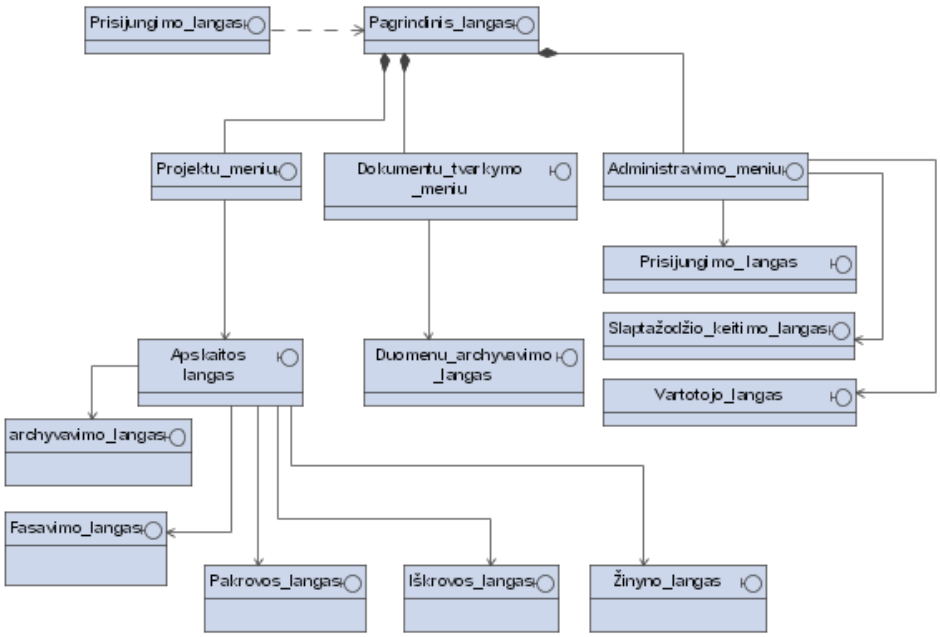
Specifikacijos panaudojimo atvejis administruoti projektus

Panaudojimo atvejis	**Administruoti projektus**
Aktorius	Projekto vadovas
Sistema	Vagonų ir trąšų fasavimo IS
Prieš sąlyga	Darbuotojas turi būti registruotas duomenų bazėje
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Iš meniu atidaroma nauja projekto redagavimo forma 2. Įvedami/trinami projekto dokumentai (failai), redaguojama data 3. Įvedami/trinami su projektu susiję vartotojai 4. Išvedama galutinė patvirtinimo forma ir spaudžiamas patvirtinimo klavišas	2.1 Sistema tikrina ar dokumentai yra aktyvūs ir ar korektiškai įvesta data 3.1 Sistema tikrina ar vartotojai yra aktyvūs 4.1 Sistema tikrina ar teisingai užpildyta forma
Po sąlyga	Nauji duomenys įrašomi į DB ir gaunamas pranešimas apie sėkmingą operaciją
Alternatyvos (nesėkmės atvejai)	2. Parinktas dokumentas yra aktyvus ir naudojamas 3. Parinktas vartotojas yra aktyvus 4. Galutinėje patvirtinimo formoje pakeitimai atliekami nekorektiškai
Vykdymo variantai	Redagavimas parenkamas iš IS “Projektas->Redaguoti” meniu punkto.
Veiklos taisyklės	Užpildyti visi reikiami formos laukai

Šaltinis: sudaryta autoriaus

3.1.2 Vartotojo sąsaja

Vartotojo sąsaja – tai nefunkcinis reikalavimas. Tai įrankių valdymo paletė, kuri atskiriems vartotojams atrodo nevienodai. Galingose sistemose yra galimybė kiekvienam vartotojui turėti savo unikalias valdymo paletes. Šiuo atveju vartotojo sąsajos modelis yra projektuojamas pagal vartotojo reikalavimus. Kuriamos sistemos vartotojo sąsajos modelis pavaizduotas 26 paveiksle.

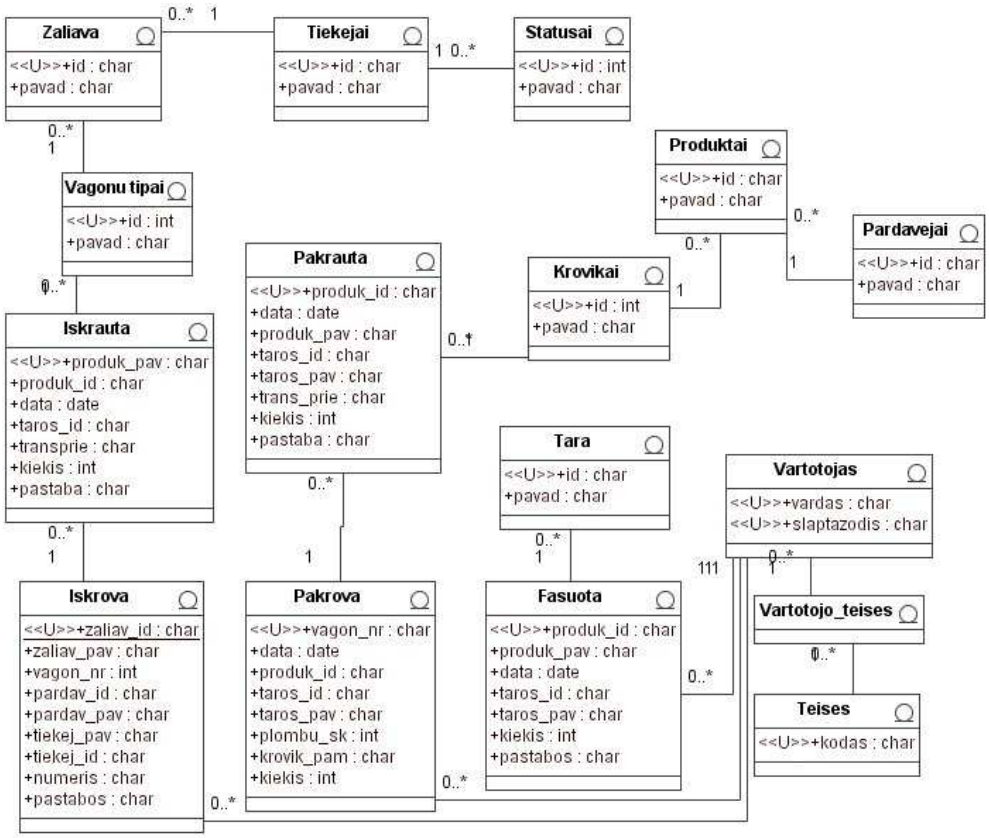


Pav 27. vartotojo sąsajos modelis.

Šaltinis: sudaryta autoriaus

3.1.3 Duomenų bazė

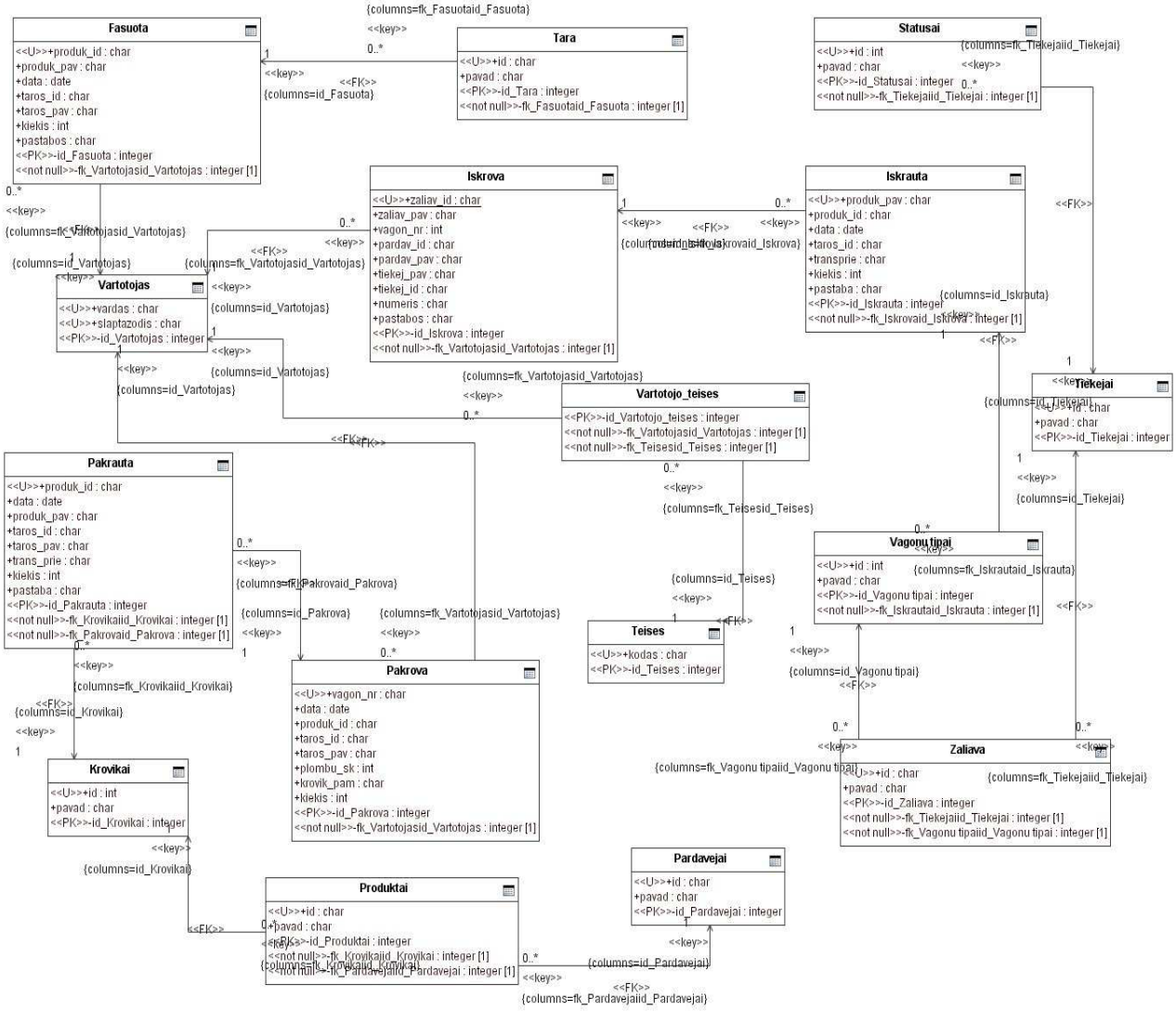
Surinkus reikalavimus iš vartotojų, kurie papasakojo, kokias funkcijas turi sistema atlikti ir kokius laukus jie nori matyti programos lange, programuotojai pagal tai nubraižė klasių diagramą. Klasės diagrama yra duomenų bazių pagrindas, kuri gali pasikeisti kintant reikalavimams. Klasių diagrama specifikuoja statinę objektų struktūrą, kuri apima atributus, operacijas ir ryšius tarp jų. Ši diagrama yra naudojama pačių programuotojų, todėl vartotojams ji gali būti ir nerodoma.



Pav. 28. Klasių diagrama

Šaltinis: sudaryta autoriaus

Iš klasių diagramos automatiškai yra generuojama duomenų bazė. Tai palengvina programuotojų darbą, nes nereikia antrą kartą sudaryti duomenų bazės schemos. Jeigu duomenų bazės schema braižoma ant popieriaus, tai po to vis tiek reikia viską suvedinėt programuojant.



Pav. 29. Duomenų bazės schema.

Šaltinis: sudaryta autoriaus

Duomenys atitinka realaus pasaulio tam tikros - skirtos automatizuoti - dalies (taikymo srities) modelį. Duomenys saugomi ir apdorojami ne bet kaip, o laikantis tam tikrų susitarimų, taisyklių. Kitaip tariant, duomenys tam tikru būdu sutvarkyti. Reiktų skirti vadinamąjį fizinį ir loginį duomenų organizavimą. Pirmasis nurodo duomenų fizinio išdėstymo būdus kompiuterio atmintyje, o antrasis - duomenų struktūros vaizdavimą - modelį, reikalingą vartotojams.

Duomenims, saugomiems DB, būdingos šios savybės:

- integruotumas;
- nepertekliškumas;
- nepriklausomumas.

Duomenų integruotumas reiškia, kad visi duomenys kaupiami ir saugomi kartu nustačius jų tarpusavio ryšius. Taip saugomus duomenis dažniausiai naudoja ne vienas, o keli vartotojai.

Duomenys saugomi vengiant jų dubliavimo. Kai yra pertekliškumas, t.y. kelios duomenų kopijos, joms veltui eikvojama atmintis, o modifikuojant duomenis tenka kelis kartus naudoti tas pačias atnaujinimo operacijas. Be to, kai duomenų kopijos atitinka skirtingas atnaujinimo stadijas, tai gali iššaukti prieštaringos informacijos pateikimą.

Duomenų nepriklausomumas reiškia, kad duomenų apdorojimo taikomosios programos nesikeičia modifikuojant duomenis.

3.2 Nefunkciniai reikalavimai ir apribojimai

3.2.1 Sistemos saugumo priemonės

Saugumo priemonė yra slaptažodis ir duomenų kontrolė. Norint dirbti su programa reikia prisijungti. O prisijungti gali tik registruoti vartotojai. Registruojant naujus vartotojus yra tikrinama, ar nėra jau tokio vartotojo, ar varde ir pavardėje bei prisijungimo varde nėra skaitmenų. Tikrinama ar telefone nėra raidžių. Nuotrauka gali būti tik „.jpg“ formato ir užimti iki 500 kB. Kitu atveju metama klaida.

3.2.2 Techninė specifikacija

Programinė įranga galima kurti naudojant MS FoxPro arba MS Access. Duomenų bazė sukuriama naudojant SQL Server 7.0 ar naujesnė versija.

3.3 Kūrimo procesas

Kūrimo procese yra vykdoma:

- Analizė
- Savybių rinkimas
- Planavimas
    - Susirinkimas
    - Testų sudarymas
    - Neatliktų darbų sąrašas
- Kūrimas
    - Programavimas poromis
    - Trūkumų ir kliūčių kūrimo procese identifikavimas
    - Komunikavimas su kitais programuotojais.

3.3.1 Analizė

Savybių rinkimas

Analizės metu buvo išsiaiškinta, kokia yra įmonės veikla. Trąšų fasavimas - viena pagrindinių įmonės veiklų. Paruoštos naudojimui, biriosios ar skystosios trąšos fasuojamos įvairiais kiekiais, pradedant nuo mažų plastikiniu maišelių ir baigiant vagonais iš kurių tiesiai perpilama i gigantiškus jūrinius keltus, kurie plukdo užsakytą produkciją i nurodytas šalis.

Vėliau buvo išsiaiškinta, ko vartotojas nori iš būsimos programos. Vartotojui reikėjo programinės įrangos, kurios pagalba būtų kontroliuojama trąšų gabenimo ir fasavimo procesai. T.y., kad nereikėtų pildyti visokių blankų, kurie dažnai pasimeta, susipurvina ar suplyšta.

Be to, buvo aptarta, kokios bus vartotojų grupės. Kokia vartotojų grupė galies prieiti prie fasavimo duomenų, iškrovos, archyvavimo. Buvo sudarytas veiklos modelis su užsakovo pagalba.

Aptarta duomenų bazės schema. Ji buvo vis papildoma laikui bėgant, nes keitėsi užsakovo reikalavimai.

3.3.2 Planavimas

Planavimo metu vysta:

- Susirinkimas
- Testų sudarymas
- Neatliktų darbų sąrašų peržiūra

3.3.2.1 Susirinkimas

Dažni susirinkimai nebuvo daromi, nes komandą sudarė tik vienas asmuo. Todėl susirinkimas buvo daromas tik su užsakovu, kai iškildavo neaiškumų, bei buvo norima parodyti, kas po kiekvienos iteracijos buvo padaryta.

3.3.2.2 Testų sudarymas

Testai buvo sudaromi naudojant Unit testų karkasą. Unit testai yra automatizuoti programų testai, tikrinantys kodo vienetus atskirai vieną nuo kito. Kiekvienas kodo vienetas (t.y. funkcija, klasė, metodas) turi savo testą ar kelis testus. Testai buvo sudaromi prieš kiekvieną iteraciją.

3.3.2.3 Neatliktų darbų sąrašas

Neatliktų darbų sąrašas buvo stebimas kiekvieną dieną, prieš pradedant darbą ir po darbo. Prieš darbo pradžią neatliktų darbų sąrašai tikrinami tam, kad žinotumėme, ką dar reikia padaryti. Po darbo tiesiog yra pažymima, kokie darbai buvo atlikti.

3.3.3 Kūrimas

Kūrimo metu buvo atliekami šie procesai:

- Programavimas poromis
- Testavimas
- Trūkumų ir kliūčių identifikavimas

3.3.3.1 Programavimas poromis

Kadangi šio darbo metu buvo atliekamas tik sistemos projektavimas, todėl nebuvo rašomas programos kodas ir po to ji realizuota. Bet jeigu būtų vykęs programavimas, tai būtų programuojama su Ms FoxPro bei kuriama duomenų bazė su Ms SQL Server.

Programuotojai būtų dirbę po du prie vieno kompiuterio. Vienas rašytų kodą, o kitas patarinėtų ir žiūrėtų, ar metodas tinka šiai klasei.

3.3.3.2 Trūkumų ir kliūčių kūrimo procese identifikavimas

Šiame etape yra identifikuojami trūkumai ir po to aptariami susirinkimo metu bei stengiamasi kuo greičiau juos pašalinti

3.3.3.3 Komunikavimas su kitais programuotojais

Žmonių „stumdymas“ bei programavimas poromis leidžia suteikia kryžminį apmokymą, kuris kitose organizacijose dažnai daromas specialiai. Jei nėra "specialistų", nebus taip, jog dalis programuotojų bus perkrauti darbu, o kiti neturės ką veikti. Galima perkelti programuotojus prie "karščiausios" projekto dalies ir jie visi bus produktyvūs. Tai yra projekto vadovo svajonių išsipildymas..

3.3.4 Iteracinis kūrimas

Kaip jau įprasta, užsakovas iš pradžių nežinojo, ką sistema turėtų daryti. Todėl jo reikalavimai kito laikui bėgant. Po kiekvienos iteracijos buvo rengiamas susirinkimas, kurio metu buvo surenkami nauji užsakovo reikalavimai.

Kaip pavyzdys, pateikiama, kas būtų atliekama kiekvienos iteracijos metu

Atlikti darbai per pirmąją iteracija:

- Sudarytas testas
- Sukurta pradinė duomenų bazė
- Sukurta vartotojo sąsaja
- Testuojama

Atlikti darbai per antrąją iteraciją:

- Papildoma duomenų bazė
- Sukuriama iškrovos forma
- Sukuriama vartotojo registracija
- Testuojama

Atlikti darbai per trečiąją iteraciją:

- Sukuriama pakrovos forma
- Redaguojama duomenų bazė
- Testuojama

3.4 Palyginamosios analizės rezultatai

23 lentelė

Naudojamų praktikų palyginimas Agile su RUP

Praktikos	RUP	XP	Scrum
Iteracinis ir priauginantis	✓	✓	✓
Koncentracija į architektūrą	✓	?	
Kliento bendradarbiavimas		✓	✓
Vadovo bendradarbiavimas	✓		✓
Rizikos valdymas	✓	✓	✓

Kokybiškas kodas		✓	✓
Didelės komandos	✓		✓
Mažos komandos		✓	✓
Sudėtingi projektai	✓	✓	✓

Šaltinis: sudaryta autoriaus pagal D. Rawsthorne, http://www.netobjectives.com/files/events/download/rup_xp_scrum_pc_030326_ppt.pdf

24 lentelė

Agile metodų palyginimas

Koncepcija	**XP**	**Scrum**	**Crystal**	**DSDM**
Komandų skaičius	1 komanda vienam projektui	1-4	Įvairiai. Iki 40 – monių. 1-10	1-6
Komandos dydis	3-16	5-9	4-8	2-6
Komandos nariai/pareigos	Klientas, programuotojas, testuotojas, seklys, treneris.	Scrum meistras, Scrum komanda, užsakovas, valdytojas	Vyresnysis projektuotojas-programuotojas, vartotojas	Vartotojo patarėjas, aiškiaregys
Projektiniai vaidmenys	Vadovas (Big Boss)	produkto savininkas, valdytojas	Rėmėjas, architektas,	Vykdantysis rėmėjas

Šaltinis: sudaryta autoriaus

Lyginant naująjį metodą, skirta informacinėms sistemoms kurti, su esamais Agile metodais, galima teigti, kad jis yra geresnis. O jis geresnis tuo:

1. FDD metodas neapima reikalavimų surinkimo ir testavimo, bet užtai yra trumpos iteracijos (nuo 1 al iki 2 sav.) Šią savybę taikom naujam metodui. Be to, FDD didelį dėmesį skiria dizainui. Kadangi mums bereikalinga ši savybė, tad jos atsisakome.
2. Scrum propaguoja kasdieninius susirinkimus, o tai tik lėtintų kūrimo procesą. Bei Scrum iteracijos trukmė 2-4 savaitės. Tai per ilgas laikotarpis vienai iteracijai. Šių dviejų savybių atsisakome.
3. Kadangi FDD neapima testavimo, todėl paimam savybę „testų rašymas prieš kodo kūrimą“ iš XP metodo. Tai yra labai gera savybė, nes visą laiką yra testuojamas kodas ir žiūrima, ar nėra klaidų. Manau, kad yra sutaupoma gan nemažai laiko, nes nereikia parašius kodą testuoti ir ieškoti klaidų, kaip kituose metoduose, kur testuojama tik iteracijos pabaigoje.

3.5 Metodo, skirto informacinėms sistemoms kurti, palyginimas su tradicine RUP metodika

RUP nėra pavienis procesas, tai apdatuotas procesų karkasas, tinkantis programinės įrangos kūrėjų organizacijoms ar komandoms, kurios pagal poreikį pasirenka tam tikrus elementus iš procesų. Produktas apima tarpusavyje susijusias žinias, grįstas artefaktais ir daugelio skirtingų veiklų detalizuotą aprašymą. RUP įtrauktas į IBM Rational Method Composer produktą, kuris leidžia derinti procesus. Suvienytas procesas buvo sukurtas, kad apimti viešą dalykinės srities procesą (žinomą kaip jungtinis procesas) ir detalesnę specifikaciją, žinomą kaip Rational Unified Process, kurią galima pateikti kaip komercinį produktą.(Vikipedija, 2007)

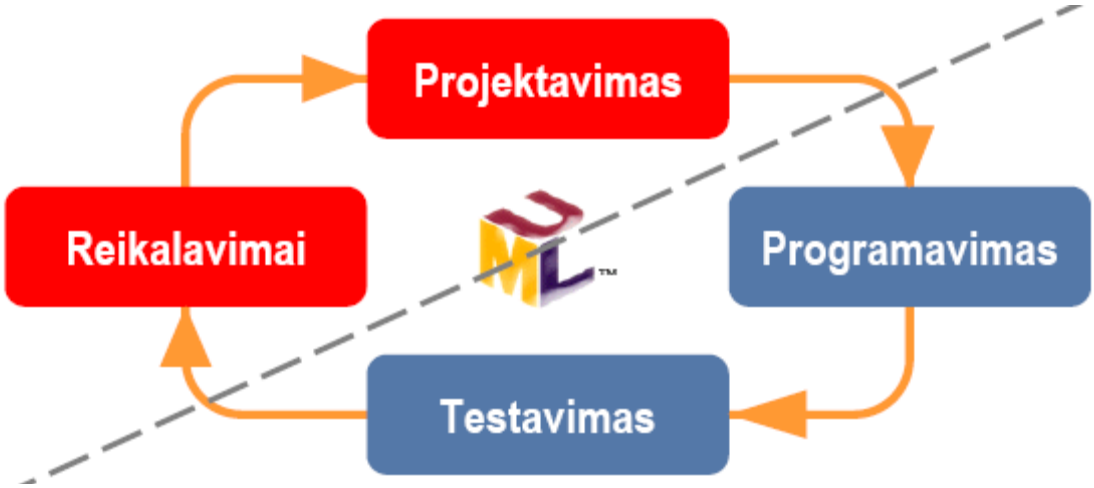
RUP gyvavimo ciklas susideda iš 4 etapų: Pradžios, parengimo, konstravimo, įdiegimo.

Pav. 30 RUP gyvavimo ciklas

Šaltinis: Z. Šleinius, 2004. http://vddb.library.lt/fedora/get/LT-eLABa-0001:E.02~2004~D_20040601_094942-17388/DS.005.0.01.ETD

Tradicinis požiūris, paremtas pasikeitimų kontrolės grupės sudarymu ir formaliu pasikeitimų vertinimu ir priėmimu arba atmetimu, remiantis apibrėžtomis procedūromis;

Naujas Agile požiūris, skatinantis nevengti pasikeitimų ir juos įvertinti kuriant programinę įrangą mažomis iteracijos, kurių metu įgyvendinamas funkcionalumas iškart aptariamas su vartotojais.



Pav. 31 RUP ir Agile procesų palyginimas

Šaltinis: „Baltijos programinė įranga“, 2007, http://vaidila.vdu.lt/~i5dasi/se2/paskaitos/02_procesai.pdf

Paveiksle pavaizduoti „Sunkiasvoriai“ procesai ir „Lengvasvoriai“ procesai. Sunkiasvoriai procesai priklauso RUP, ITIL ir kt. Sunkiasvoriai procesai akcentuoja reikalavimų surinkimą ir projektavimą, t.y. pradines veiklas. Lengvasvoriai procesai priklauso Agile. Juose akcentuojama programavimas ir testavimas, t.y. galinės veiklos.

25 lentelė

Programinės įrangos kūrimo problemos

Problema	**Sprendimo būdas**	
	Metodas, paremtas Agile metodų savybėmis	**RUP**
Neefektyvi komunikacija IT komandoje	Pagal metodo savybę, yra būtinas bendradarbiavimas tarp komandos narių, Be to yra programuojama poromis bei yra kolektyvinė kodo nuosavybė, t.y., kad kiekvienas komandos narys žino, ką daro jo kolega.	Pagal RUP principus, komunikavimas tarp komandos narių yra būtinas. Tad ši problema neturėtų iškilti kuriant programinę įrangą.
Nepilni ir neaiškūs reikalavimai	Į kūrimo procesą yra įtraukiamas užsakovas, todėl, jei kas nors yra neaišku, visada galima pasiklaust užsakovo.	Gali iškilti problema, nes reikalavimai yra surenkami kūrimo pradžioje. Užsakovas gali iškarto nepateikti pilnų reikalavimų, nes pats gerai nežino, ko jis nori iš sistemos.
Reikalavimų pasikeitimai	Reikalavimų pasikeitimai yra priimtini. Po kiekvienos iteracijos daromi susirinkimai, kurių metu užsakovas gali pakeisti reikalavimus. Kadangi iteracijos yra trumpos, todėl pasikeitus reikalavimams reikia mažiau perdaryti. O tokiu būdu yra sutaupomas laikas bei pinigai.	Kadangi viskas yra dokumentuojama, todėl labai sunku tinkamai sureaguoti į reikalavimų pasikeitimą
IT komandos narių pasikeitimai	Kadangi yra kolektyvinė kodo nuosavybė ir kiekvienas narys yra susipažinęs su kolegos darbu, bet kada jis gali tęsti jo pradėtą darbą, jam susirgus, ar išėjus iš darbo. Be to, yra programuojama poromis,	Išėjus vienam, ar keliems komandos nariams iš darbo, gali sužlugti visas projektas, nes niekas nežino, ką jie darė. Be to, naujam darbuotojui reiktų tyrinėti visą dokumentaciją, kurios yra tikrai

	todėl retas atvejis, kad abu darbuotojai pasišalina iš projekto.	nemažai. Tokiu būdu yra gaištamas laikas bei projekto kaina didėja.
Nedokumentuotos sistemos	Pagal metodo savybes, dokumentacijos pakanka.	Dokumentacijos tikrai pakanka.

Šaltinis: sudaryta autoriaus

Pagal lentelėje pateiktus duomenis, galima teikti, kad kuriant programinę įranga pagal RUP metodiką, tikimybė, kad iškils problemų kūrimo metu yra didesnė, nei kuriant patobulintu Agile metodu, skirtu informacinėms sistemoms kurti.

IŠVADOS

1. Agile požiūris labiau tinkamas projektams, kurie gali būti vykdomi ir finansuojami pagal iteracinį gyvavimo ciklą.
2. Išanalizavus sistemų kūrimo metodus, buvo pasirinktas Agile požiūris, nes Agile metodologija teigia, kad kuriant programinę įrangą yra efektyviau išsiaiškinami reikalavimai turint dalį veikiančios programos nei išsamiai dokumentuojant juos. Bendraujant gyvai su užsakovu yra geriausia komunikacijos priemonė, nes užsakovo reikalavimai gali kisti ir juos reikia įvertinti kuriant programinę įrangą trumpomis iteracijomis.
3. Išanalizavus Agile metodus, nustatyta, kad tinkamiausi metodai informacinių sistemų kūrime yra Scrum, XP bei FDD. Bet jie nėra visiškai tinkami naudoti, tokie, kokie jie yra, nes XP metodas skirtas nedidelėms programuotojų grupėms, FDD metodas neapima reikalavimų surinkimo bei testavimo; Scrum metodas siūlo rengti kasdieninius susirinkimus, kurie tik gaišintų laiką, nes pakaktų susirinkimo po kiekvienos iteracijos.
4. Norint pagerinti informacinių sistemų kūrimo procesą, reikia sukurti naują metodą paremtą XP, Scrum, FDD bei ICONIX metodų savybėmis:
    - Greitas kūrimas;
    - Visos sistemos modelio sukūrimas
    - Efektyvus reagavimas į reikalavimų pasikeitimus;
    - Trūkumų bei kliūčių identifikavimas
    - Nuolatinis bendradarbiavimas su užsakovu;
    - Programavimas poromis
    - Užtikrina bendravimą tarp komandos narių;
    - Naudojant UML diagramas:
        - Panaudojimo atvejų
        - Sekos diagrama.
    - Dėmesys funkcionalumui;
    - Kuo paprastesnis dizainas
    - Neatliktų darbų sąrašų tikrinimas
    - Trumpos iteracijos;

5. Atlikus palyginamąją analizę, buvo įrodyta, kad naujas metodas, skirtas informacinėms sistemoms kurti, yra geresnis už esamus Agile metodus bei RUP metodiką, nes dėl trumpų iteracijų kūrimo procesas vyksta greičiau, yra sutaupoma pinigų bei pristatomas kokybiškas produktas žymiai greičiau.
6. Kadangi šis metodas buvo tikrinamas tik su vienu projektu, todėl yra sunku pasakyti, ar jis tikrai yra tinkamas visokio dydžio projektams. Siūlyčiau atlikti dar kelis projektus naudojant šį metodą, kad įsitikinti jo tinkamumu.

LITERATŪRA

1. Šarkiūnaitė I., Krikščiūnienė D., Simutis R (2007). Magistro BD metodiniai nurodymai 2007 [interaktyvus]. [žiūrėta 2007 m. gegužės 21 d.]. ftp://ftp.vukhf.lt/aulay/_Informatikos_kat/_Studentams/Metodiniai/Magistro%20BD%20metodiniai%20nurodymai%202007.PDF

2. Frederick Brooks, (**1995**), The Mythical Man-Month

3. D. Šilingas. Programinės įrangos reikalavimų valdymo principai ir praktika. [interaktyvus] [žiūrėta 2006m spalio 20 d.]. Prieiga per Internetą: http://www.bpi.lt/text.php?lang=1&item=161&arg=119

4. A. Abran, J. W. Moore . Software Requirements. [interaktyvus] [žiūrėta 2007 spalio 25 d]. Prieiga per Internetą: http://www.swebok.org/ch2.html

5. S. Nuseibeh, Requirements Engineering: A Roadmap. [interaktyvus] [žiūrėta 2007 m. spalio 25 d.]. Prieiga per Internetą: http://www.doc.ic.ac.uk/~ban/pubs/sotar.re.pdf

6. Ambler S.W. (2006) Agile Enterprise Architecture. . [interaktyvus] [žiūrėta 2007 m. Spalio 25d.]. Prieiga per Internetą: http://www.agiledata.org/essays/enterpriseArchitecture.html

7. Butkienė. R. Informacijos sistemų inžinerijos metodai ir modeliai . [interaktyvus] [žiūrėta 2008 m. balandžio 25 d.]. Prieiga per Internetą: ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

8. Ambler S.W. (December 3, 2006) Agile Enterprise Architecture [interaktyvus] [žiūrėta 2007 m. sausio 08 d.]. Prieiga per internetą: http://www.agiledata.org/essays/enterpriseArchitecture.html

9. Ambler S.W. (2006) Imperfectly Agile: You Too Can Be Agile! [interaktyvus]. [žiūrėta 2007 m. sausio 08 d.]. Prieiga per internetą: http://www.ddj.com/dept/architect/192700252?cid=Ambysoft

10. Ambler S. W. (June 12, 2006) The Agile Unified Process (AUP) [interaktyvus]. [žiūrėta 2007 m. sausio 11 d.]. Prieiga per internetą: http://www.ambysoft.com/unifiedprocess/agileUP.html

11. J. Bach. Defining Agile Methodology. . [interaktyvus] [žiūrėta 2007 m. lapkričio 20 d.]. Prieiga per Internetą: http://www.satisfice.com/blog/archives/45

12. Bennekum A. (2001) Manifesto for Agile Software Development [interaktyvus]., [žiūrėta 2007 m. sausio 11 d.]. Prieiga per internetą: http://agilemanifesto.org/

13. Murugesan S., Ginige A.. Web Engineering: Introduction and Perspectives, 2005 . [interaktyvus] [žiūrėta 2007m lapkričio 06 d] Prieiga per Internetą: http://www.idea-group.com/downloads/excerpts/01%20Suh.pdf

14. Hayes S., Andrews M. (2005) An Introduction to Agile Methods [interaktyvus]., [žiūrėta 2007 m. Birželio 16 d.]. Prieiga per internetą: http://www.wrytradesman.com/articles/IntroToAgileMethods.pdf

15. XP. Ekstremalus programavimas. [interaktyvus] [žiūrėta 2007 m. birželio 16 d.] Prieiga per internetą: http://gedmin.as/study/inf98/pkpm/xp.doc

16. Butkienė R. Informacijos sistemų inžinerijos metodai ir modeliai. . [interaktyvus] [žiūrėta 2008 m sausio 11 d.]. Prieiga per Internetą: ftp://isd.ktu.lt/Isd/Butkiene/T120M051/3_1_Agile_Metodologijos.pdf

17. Bach J. Defining Agile Methodology. . [interaktyvus] [žiūrėta 2008 m. sausio 18 d.]. Prieiga per Internetą: http://www.satisfice.com/blog/archives/45

18. Agile Alliance. [žiūrėta 2007 m. gegužės 21 d.]. Prieiga per Internetą: http://agilemanifesto.org/principles.html

19. Agile Alliance. Manifesto for Agile Software Development. [žiūrėta 2007 m. birželio 16 d.]. Prieiga per Internetą: http://www.agilemovement.it/index.php?newlang=eng

20. Ambler S.W. Agile Modeling and eXtreme Programming (XP). Agile modeling. March 6, 2006 [žiūrėta 2007 m. birželio 16 d.]. Prieiga per Internetą: http://www.agilemodeling.com

21. Milevičienė E. , Šilingas D. , Programinės įrangos reikalavimų pasikeitimai ir jų valdymas. [interaktyvus] [žiūrėta 2007 m. birželio 20 d.] Prieiga per internetą: http://www.ktu.lt/lt/apie_renginius/konferencijos/2006/k6_02/IT2005/Sekc09.pdf

22. S. Thomas, 2008 An Agile comparison.. [interaktyvus] [žiūrėta 2007 m. birželio 23 d.] Prieiga per internetą: http://www.balagan.org.uk/work/agile_comparison.htm

23. Ambler S. W. (November 21, 2006) Agile Modeling [interaktyvus]. [žiūrėta 2007 m. sausio 08 d.]. Prieiga per internetą: http://www.agilemodeling.com/

24. Targamadzė A. Procesų modeliai. . [interaktyvus] [žiūrėta 2007 m. sausio 08 d.]. Prieiga per internetą http://oras.if.ktu.lt/moduliai/T120M026/Skaidres/PIP_IV_dalis_2005.pdf

25. Cunningham W. (2001) Manifesto for Agile Software Development [interaktyvus] [žiūrėta 2006 m. gruodžio 08 d.]. Prieiga per internetą http://agilemanifesto.org/

26. Delta. (2006) Dinaminis sistemų kūrimo metodas [interaktyvus] [žiūrėta 2008 m. balandžio 08 d.]. Prieiga per internetą http://www.is.lt/delta/index.php?turinys=4

27. A. McDonald, R. Welland. Agile Web Engineering (AWE) Process. [interaktyvus] [žiūrėta 2008 m. balandžio 08 d.]. Prieiga per internetą: http://www.dcs.gla.ac.uk/publications/PAPERS/7087/TR-2001-98%5B1%5D.pdf

28. Vikipedija (2007) [interaktyvus] [žiūrėta 2008 m. Gegužės 15 d.]. Prieiga per internetą http://lt.wikipedia.org/wiki/UML_klasi%C5%B3_metamodelis

29. UAB „Baltijos programinė įranga“ [interaktyvus] [žiūrėta 2008 m. Gegužės 19 d.]. Prieiga per internetą: http://www.bpi.lt/text.php?lang=1&item=162&arg=152

30. UAB „Baltijos programinė įranga“ [interaktyvus] [žiūrėta 2008 m. Gegužės 28 d.]. Prieiga per internetą: http://www.bpi.lt/text.php?arg=119&item=161&lang=1

31. T. Bilevičienė (2006) Kompiuterinės ir informacinės sistemos samprata. [interaktyvus] [žiūrėta 2008 m. Gegužės 19 d.]. Prieiga per internetą: http://www3.mruni.lt/padaliniai/FAKULTETAI/vvf/tik/dokumentai/Smulkmenos/Informatika/4%20dalis1.htm#turinys

32. Z. Šleinius. 2004. Automatizuotas kompiuterizuotos IS prototipo kūrimas informacijos srautų specifikacijos pagrindu. [interaktyvus] [žiūrėta 2008 m. Gegužės 19 d.]. Prieiga per internetą http://vddb.library.lt/fedora/get/LT-eLABa-0001:E.02~2004~D_20040601_094942-17388/DS.005.0.01.ETD