KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Šarūnas Packevičius

NUOTOLINIS MODULIŲ TESTAVIMAS MOBILIESIEMS ĮRENGINIAMS

Magistro darbas

Vadovas
dr. Eduardas Bareiša

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

TVIRTINU
Katedros vedėjas
dr. Eduardas Bareiša
2005-05

NUOTOLINIS MODULIŲ TESTAVIMAS MOBILIESIEMS ĮRENGINIAMS

Informatikos inžinerijos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių k. katedros lekt.
dr. J. Mikelionienė
2005-04-

Vadovas
dr. Eduardas Bareiša
2005-05-

Recenzentas
dr. Tomas Blažauskas
2005-05-

Atliko
IFM 9/2 gr. stud.
Šarūnas Packevičius
2005-05-

KAUNAS, 2005

Remote Unit Testing for Mobile Devices

SUMMARY

Software for mobile devices is becoming increasingly popular. The limited resources and limited user interface brings new issues in software development process for mobile devices. These limitations should be considered especially in design and testing phases of software development process. The software for mobile devices can not be thoroughly tested, because unit tests can not be stored in mobile device due to the lack of storage on it. Testing results can not be easily observed on small screens of mobile devices.

This work depictures the influence of mobile technologies to software development process. The influence is exemplified with a project “Mobile Professor”.

This work concentrates on problems encountered in software testing process for mobile devices especially in unit testing. The work presents other authors suggested solutions for earlier mentioned problems, and also suggest the new approach for performing unit testing for mobile devices, which does not suffer for these limitations of mobile devices. Existing and suggested methods are compared and evaluated experimentally.

Turinys

1. Įvadas 9
   1.1. Tikslas 9
   1.2. Pateikimas 10
2. Analitinė dalis 11
   2.1. Bandomasis projektas 15
   2.2. Nagrinėjamos problemos 15
   2.3. Programinės įrangos testavimas mobiliuosiuose įrenginiuose 16
   2.4. Programinės įrangos testavimo vieta programinės įrangos kūrime 17
   2.5. Testais paremtas programinės įrangos kūrimas 18
   2.6. Modulių testų vieta programinės įrangos testavime 19
   2.7. Modulių testavimo karkasas 20
   2.8. Resursų augimo tendencijos mobiliuosiuose įrenginiuose 22
   2.9. Mobiliųjų įrenginių programinės įrangos sudėtingumo, dydžio augimas 23
   2.10. Siūlomi sprendimo būdai 24
   2.10.1. Emuliatoriaus sprendimo būdas 24
   2.10.2. Perteklinių testų pašalinimas 27
   2.10.3. Mock objektai 29
   2.10.4. Nuotolinis modulių testavimas 31
   2.10.4.1. Universalaus valdiklio būdas 35
   2.10.4.2. Konkretaus valdiklio būdas 38
   2.10.4.3. Darbas aplinkoje be atspindžių 39
3. Projektinė dalis 40
   3.1. Sistemos paskirtis 40
   3.2. Panašūs projektai 40
   3.2.1. Kowloon Wan Yah koledžo sprendimas 40
   3.2.2. „Mobile teacher“ programinis sprendimas 41
   3.2.3. eScholar 42
   3.3. Esminiai reikalavimai 43
   3.4. Architektūra 44
   3.5. Mobilių sistemų architektūros 49
   3.6. Realizacija 50
   3.7. Vartotojo sąsaja 53

3.8. Duomenų bazė 54
3.9. Energijos taupymas 55
3.10. Sistemos testavimas 56
3.11. Vartotojo dokumentacija 57
3.12. Diegimas ir palaikymas 57
4. Tyrimo dalis 59
4.1. Palyginimo metrikos 59
4.2. Metodų palyginimas 62
5. Eksperimentinė dalis 64
5.1. Tikslas 64
5.2. Realizacija 64
5.3. Universalaus valdiklio realizacija 66
5.4. Konkretaus valdiklio realizacija 67
5.5. Palyginimo kriterijai 68
5.6. Procesas 69
5.7. Palyginimo rezultatai 70
6. Išvados 74
7. Literatūra 75
8. Terminų ir santrumpų žodynas 80
9. Priedai 84
Priedas A: Energijos suvartojimas 84
Priedas B: Eksperimentų rezultatų duomenys 86
Priedas C: Testavimo rezultatų stebėjimo vartojo sąsajos 87
Priedas D: Pranešimai konferencijose magistrinio darbo tema 88

Lentelių sąrašas

2.1 lentelė. Mobilių technologijų įtaka programinės įrangos kūrimo procesui 12
3.1 lentelė. Mobilių sistemų architektūrų palyginimas.....50
4.1 lentelė. Metodo metrikos ir jų reikšmės.....60
4.2 lentelė. Reikšmių skaitinis įvertinimas 61
4.3 lentelė. Metodų palyginimas.....62
5.1 lentelė. Programinės įrangos ir modulių testų dydžiai.....69
9.1 lentelė. Darbo trukmės priklausomybė nuo prisijungimo būdo.....84
9.2 lentelė. Eksperimentų rezultatų duomenys 86

Paveikslų sąrašas

2.1 pav. Programinės įrangos gyvavimo ciklas. 18
2.2 pav. Modulių testavimo vieta programinės įrangos testavime 19
2.3 pav. Programinė įranga ir modulių testavimo karkasas 20
2.4 pav. Paprasto modulių testavimo karkaso klasių diagrama 21
2.5 pav. Modulių testavimo karkaso pavyzdinė realizacija 22
2.6 pav. Fizinis (a) ir loginis (b) mobilumas 25
2.7 pav. Testuojama klasė (b) ir jos modulių testai (a) 27
2.8 pav. Perteklinių testų procentinis santykis tarp įrankių sugenruotų testų 29
2.9 pav. Klasių santykis 30
2.10 pav. Modulių testavimo karkasas 31
2.11 pav. Nuotolinis modulių testavimo karkasas 33
2.12 pav. Modulio testo duomenų schema 35
2.13 pav. Testavimo karkaso mobiliajame įrenginyje struktūra 36
2.14 pav. Konkretaus valdiklio vykdymas 38
3.1 pav. Programinės įrangos panaudojimo atvejai 43
3.2 pav. Architektūros pateikimas 44
3.3 pav. Programinės įrangos išskaidymas į paketus 46
3.4 pav. Darbo būsenos diagrama 47
3.5 pav. Sistemos išdėstymo vaizdas 48
3.6 pav. Mobilių sistemų architektūros 49
3.7 pav. Pasirinkta sistemos architektūra 51
3.8 pav. Programinės įrangos duomenų bazės schema 55
3.9 pav. Darbo trukmės sutrumpėjimo priklausomybė nuo prisijungimo būdo 56
5.1 pav. Nuotolinio modulių testavimo karkaso realizacija 65
5.2 pav. Testų vykdytojo valdančiajame įrenginyje vartotojo sąsaja 66
5.3 pav. Modulio testo duomenų struktūra 67
5.4 pav. Modulio testo vykdymas 68
5.5 pav. Įvykdytų testų kiekis 71
5.6 pav. Testų vykdymo laikas 72
5.7 pav. Atminties sunaudojimas 72
5.8 pav. Ekraninių langų kiekis 73
9.1 pav. Darbo trukmės priklausomybė nuo prisijungimo būdo 84
9.2 pav. Darbo trukmės sutrumpėjimo priklausomybė nuo prisijungimo būdo 85

9.3 pav. Testavimo rezultatų stebėjimas mobiliajame įrenginyje.....87
9.4 pav. Testavimo rezultatų stebėjimas staliniame kompiuteryje 87

1. Įvadas

Patobulėjus mobilioms technologijoms, vystant mobiliuosius skaičiavimus *(Mobile Computing*) [5, 18, 34], visur egzistuojančius skaičiavimus (*Persvasive Computing*) [16, 33, 42, 45], vis daugiau kuriama programinės įrangos mobiliesiems įrenginiams, tokiems kaip delniniai kompiuteriai, protingi mobilieji telefonai. Programinė įranga mobiliesiems įrenginiams dažniausiai yra skiriama plačiam vartotojų ratui. Kuriant programinę įrangą šiems mobiliesiems įrenginiams susiduriama su daugeliu problemų jos kūrimo procese. Šias problemas sukelia mobiliųjų įrenginių specifika. Esminiai apribojimai yra riboti resursai, tokie kaip sąlyginai lėti procesoriai, maži atminties kiekiai, taip pat ribota vartotojo sąsaja [46].

Programinės įrangos projektavimo etape reikia ypatingą dėmesį atkreipti į šiuos apribojimus, taip pat svarbu atkreipti dėmesį ir į mobiliųjų įrenginių įvairovę. Egzistuoja įvairiausių konfigūracijų, skirtingų techninių galimybių, turinčių skirtingas architektūras, mobiliųjų įrenginių. Visa tai turi būti įvertinta projektuojant programinę įrangą mobiliems įrenginiams, ypač tuo atveju, kai programinė įranga skirta plačiam vartojimui.

Taip pat mobilios technologijos įneša ir savo specifiką programinės įrangos testavimo etape. Dėl techninės įrangos apribojimų sudėtinga ne tik sukurti programinę įrangą mobiliesiems įrenginiams, bet ir taip pat sudėtinga ją testuoti.

1.1. Tikslas

Mobilieji įrenginiai resursų atžvilgiu yra gana riboti. Juose yra nedideli atminties ištekliai, menkos didelių duomenų kiekų išvedimo galimybės.

Dėl resursų trūkumų yra sudėtinga išsamiai ištestuoti programinę įrangą, skirtą mobiliesiems įrenginiams. Testuojant susiduriama su nepakankamos vietos buvimo mobiliajame įrenginyje testiniams atvejams saugoti problema. Ypač tai aktualu taikant automatinį testinių atvejų generavimą [7]. Taip pat sudėtinga stebėti testavimo rezultatus mobiliajame įrenginyje, dėl jų didelio kiekio ir menkų vartotojo sąsajos galimybių [39].

Šiame darbe pateikiamas nuotolinis modulių testavimas mobiliesiems įrenginiams, kuris turėtų išspręsti anksčiau minėtas problemas veikiančias programinės įrangos mobiliesiems įrenginiams testavimą, pateikiamas jo eksperimentinis pagrindimas.

1.2. Pateikimas

Likusiose dokumento dalyse pateikiamas detalus problemos apibūdinimas, jos svarba programinės įrangos kūrimo procese. Analitinėje dalyje pateikiama programinės įrangos testavimo vieta programinės įrangos kūrimo procese, modulių testavimo vieta programinės įrangos testavimo procese. Pateikiamos mobiliųjų įrenginių techninės įrangos augimo tendencijos, taip pat pateikiamos programinės įrangos sudėtingumo ir dydžio augimo tendencijos mobiliuosiuose įrenginiuose. Analitinėje dalyje pateikiami egzistuojantys problemos sprendimai, taip pat pasiūlomas naujas būdas jiems išspręsti. Eksperimentinėje dalyje pateikiamos kelios realizacijos siūlomo būdo, jo eksperimentinis palyginimas su egzistuojančiais sprendimais. Išvadų dalyje apibendrinami darbo rezultatai.

2. Analitinė dalis

Mobilios technologijos daro didelę įtaką programinės įrangos kūrimui. Įvairūs autoriai literatūroje mini ir pateikia naujus programinės įrangos kūrimo modelius ir praktikas, orientuotas į programinės įrangos kūrimą mobiliesiems įrenginiams [54]. Autoriai teigia, kad kuriant programinę įrangą mobiliesiems įrenginiams, programinės įrangos kūrimas pasikeis iš skirto konkrečiam vartotojui ar užsakovui, į orientuotą rinkai [54]. Pastebimos tendencijos, kad programinės įrangos kūrimas krypsta nuo kūrimo vienam užsakovui, link programinės įrangos produktų šeimų kūrimo [22]. Produktų šeimos taip pat įtakoja programinės įrangos kūrimo procesus ir pateikia naujus programinės įrangos architektūros projektavimo metodus [30], tokius kaip COPA, FAST, FORM, KobrA, QADA.

Mobilios technologijos veikia ir tradicinį programinės įrangos kūrimo procesą (reikalavimai, projektavimas, kūrimas, testavimas, diegimas, palaikymas) [44]. Mobilių technologijų įtaka programinės įrangos kūrimui pateikta 2.1. lentelėje.

2.1 lentelė. Mobilių technologijų įtaka programinės įrangos kūrimo procesui

Fazė	Įtaka
Reikalavimų surinkimas	Iškeliama varotojo sąsajai daug reikalavimų, apribojimų. Techninės įrangos apribojimai [41]. Įrenginių įvairovė.
Projektavimas	Projektuojama atsižvelgiant į apribojimus reikalavimuose. Programinės įrangos bendravimas su aplinkinėmis sistemomis verčia pasirinkti kitokiais [31] architektūras, pvz., Client-Server [6], Peer to Peer [2], C2 [43]. Darbas esant ryšiui su tinklu, darbas atjungtoje aplinkoje, judėjimas tarp tinklų darbo metu [44]. Duomenų sinchronizavimas. Saugumas [44]. Patikimumas [44]. Energijos suvartojimas [49]. Visur esantys skaičiavimai [5]. Nuo konteksto priklausančios paslaugos [33]. Lokalizavimas.
Kūrimas	Kuriama įvairaus tipo įrenginiams. Vartotojo sąsajos apribojimai derinimui ir klaidų taisymui [37]. Riboti techninės įrangos resursai [46].
Testavimas	Testavimas esant ribotiems mobiliųjų įrenginių resursams [46], vartotojo sąsajai [46].
Diegimas	Programinės įrangos diegimas daugelyje skirtingų įrenginių[36]. Didelis kiekis vartotojų.
Palaikymas	Programinės įrangos atnaujinimas mobiliuosiuose įrenginiuose[36]. Didelis kiekis vartotojų.

Reikalavimų surinkimo metu, surenkama daug reikalavimų, kurie apriboja kūrimo galimybes, pvz., ribota vartotojo sąsaja (maži ekranai, nepatogus duomenų įvedimas). Pačio darbo mobilioje aplinkoje specifika reikalauja, kad programinė įranga dirbtų patikimai, t. y. dirbtų dingus tinklo ryšiui, dirbtų nesant tinklo ryšiui ilgą laiką. Taip pat minimizuotų

perduodamų duomenų kiekius, dėl santykinai didelių perduodamų tinklu duomenų kainų ir santykinai lėtų duomenų perdavimo spartų. Taip pat reikia atsižvelgti į mobiliųjų įrenginių įvairovę, ypač tuo atveju, jei programinė yra skirta plačiai rinkai. Programinė įranga turi dirbti įvairių tipų mobiliuosiuose įrenginiuose, turinčiuose skirtingas procesorių architektūras, skirtingus atminties kiekius, ne vienodas duomenų įvedimo galimybes.

Programinės įrangos projektavimo metu svarbu atsižvelgti į reikalavimus, kurie atsirado atsižvelgiant į mobiliųjų įrenginių specifiką. Programinė įranga skirta įranga mobiliesiems įrenginiams neegzistuoja izoliuota, ji nuolat bendrauja su aplinkos paslaugomis [33, 44] ir kitomis sistemos dalimis. Tai veikia ir programinės įrangos architektūros pasirinkimą. Dažniausiai pasirenkamos Client-Server [6] tipo architektūros, bet vystantis visur esantiems skaičiavimais [5], ir nuo konteksto priklausančioms paslaugoms [33], programinės įrangos kūrėjai renkasi kitokius architektūros būdus, pvz., Peer 2 Peer, C2 ir pan. Programinės įrangos mobilumas ir bendravimas su išorinėmis paslaugomis verčia ieškoti būdų kaip programinė įranga galėtų dirbti esant nepastoviam tinklui, tik retkarčiais sinchronizuodama duomenis. Projektavimo metu taip pat tenka spręsti duomenų sinchronizavimo problemas. Tai yra ypatingai aktualu pasirinkus sistemos realizaciją atjungtoje aplinkoje, kurioje tik retkarčiais vykdomas duomenų sinchronizavimas. Tuo atveju tenka spręsti duomenų sinchronizavimo problemas: kurį įrašą išsaugoti, jei keli skirtingi vartotojai atliko pakeitimus ir sinchronizavimo metu bando perkelti duomenis į pagrindinę saugyklą (joje saugoma pilna duomenų kopija). Projektavimo metu taip pat reikia įvertinti saugumo reikalavimus, užtikrinti tinklu perduodamų duomenų saugumą. Tam taikomas duomenų šifravimas, viešo rakto technologijos ir pan. Taip pat reikia atsižvelgti ir į pačio įrenginio apsaugą nuo pašalinių žmonių. Pametus mobiluįį įrenginį programinė įranga turi užtikrinti, kad ne sistemos vartotojai negalėtų pasinaudoti mobiliajame įrenginyje esančiais duomenimis. Tam įgyvendinti galima naudoti biometrines apsaugos priemones, slaptažodžius. Programinė įranga turėtų būti ir patikima, vien dėl to, kad mobilios technologijos veržiasi į mūsų gyvenimus [5], naudojama medicinoje [33]. Programinės įrangos kūrimas plačiai rinkai iškelia irgi aukštus patikimumo ir kokybės reikalavimus.

Literatūroje plačiai diskutuojama energijos suvartojimo problema mobiliuosiuose įrenginiuose [49]. Vieni autoriai teigia, kad nėra kaip didinti energijos šaltinių mobiliuosiuose įrenginiuose talpos. Siūloma kurti programinės įrangos skaičiavimo algoritmus atsižvelgiant ne tik į esamos atminties kiekius, procesoriaus resursus, bet ir į energijos suvartojamumą [44]. Algoritmo efektyvumo rodiklis tampa, ne skaičiavimo sparta, o kiek energijos resursų sunaudojama atlikti skaičiavimams. Kiti autoriai siūlo ir dirba siekdami sumažinti energijos suvartojimą, naudojant nesinchroniškai dirbančius lustus [13].

Kūrimo metu tenka pritaikyti programinė daugeliui skirtingų mobiliųjų įrenginių. Tam pasiekti naudojami kompiliatoriai, kurie pagal vieną programos kodą generuoja įvairių architektūrų įrenginiams. Naudojant virtualias mašinas (Java VM, Microsoft .NET) galima pritaikyti programinę įrangą įvairiems mobiliesiems įrenginiams. Pritaikymas atliekamas pateikiant visiems atvejams skirtą virtualią mobiliojo įrenginio mašiną, o kuriamas programas kurti pritaikytas virtualiai mašinai. Šiuo atveju kūrėjai turėtų sukurti tik vieną programos versiją, pritaikytą virtualiai mašinai. Vartotojo sąsajos kūrimas atliekamas transformuojant bendrai apibrėžtą vartotojo sąsajos modelį į konkretaus mobilaus įrenginio vartotojo sąsajos modelį.

Testavimo metu tenka susidurti su mobiliųjų įrenginių techninės įrangos apribojimais. Nedideli atminties kiekiai neleidžia sutalpinti reikiamą testų kiekį į mobilųjį įrenginį. Ribotos vartotojo sąsajos galimybės neleidžia efektyviai peržiūrėti testavimo rezultatų. Mobiliųjų įrenginių skirtingos techninės galimybės (įvairių dydžių, spalvų ekranai, skirtingos įvedimo galimybės, pvz., jautrus ekranas, mažos klaviatūros) reikalauja ne tik suprojektuoti vartotoją sąsają, kuri tiktų daugeliui įrenginių, bet taip pat sukelia problemų jų testavime. Rankinis testavimas sunaudoja daug laiko, o atliekant automatinį testavimą reikia atsižvelgti, kad ta pati vartotojo sąsaja gali būti skirtingai atvaizduota skirtinguose mobiliuosiuose įrenginiuose. Gali atsirasti net visiškai nauji vartotojo sąsajos langai, kurių nėra kituose mobiliuosiuose įrenginiuose. Pavyzdžiui viename mobiliajame įrenginyje duomenų įvedimo langas su dviem įvedimo laukais gali būti atvaizduotas kaip vienas langas, o kitame mobiliame įrenginyje išskaidytas du, nes tame įrenginyje neišeina pavaizduoti abiejų įvedimo laukų vienu metu.

Diegimo metų tenka atrinkti reikiamą programinės įrangos versiją skirtą konkrečiam mobiliajam įrenginiui. Taip pat reikalingi būdai kaip pateikti programinę įrangą vartotojams. Jos diegimas turi būti nesudėtingas, nes vartotojas pats dažniausiai turėtų įsidiegti programinė įrangą į savo mobilųjį įrenginį.

Programinės įrangos palaikymo metu išleidžiami programinės įrangos pataisymai. Dėl didelio vartotojų kiekio sudėtinga pateikti vartotojams naujas programinės įrangos versijas. Versijų pateikimui vartotojams galima naudoti automatinį programų atnaujinimą, kurio idėja yra ta, kad programinė įranga mobiliajame įrenginyje pasitikrina, ar nėra jos atnaujinimo internete ar dar kokioje nors kitoje saugykloje. Radus savo naujesnę versiją turėtų būti vykdomas atsinaujinimas, kurio metų būtų atsiunčiama nauja programinės įrangos versija, pašalinama sena ir įdiegiama nauja programinės įrangos versija.

2.1. Bandomasis projektas

Norint išanalizuoti mobilių technologijų įtaką programinės įrangos kūrimo procese buvo realizuotas projektas „Mobilus dėstytojas“. Jo vykdymo metu buvo analizuojama, kokią įtaką daro mobilių technologijų specifika reikalavimų surinkimui, projektavimui, realizavimui, testavimui, diegimui ir palaikymui.

Buvo siekiama sukurti mobilaus dėstytojo programinę įrangą. Dėstytojas galės visus darbus atlikti naudodamasis delniniu kompiuteriu. Naudodamasis delniniu kompiuteriu ir programine įranga dėstytojas galės sutaupyti laiko, kuris būtų išleidžiamas darbui su popieriais (pažymių suvedinėjimas į kompiuterį nuo popieriaus lapų, atsiskaitymo rezultatų žymėjimas, studentų lankomo suvedinėjimas į kompiuterį). Dėstytojas naudodamasis delniniu kompiuteriu galės naudotis programinės įrangos teikiamomis funkcijomis:

- Lankomumo žymėjimas, ataskaitos.
- Atsiskaitymų rezultatų žymėjimas, jų ataskaitos.
- Rezultatų publikavimas internete.
- Darbų skyrimas studentams, jų publikavimas internete.
- Studentų darbu priėmimas per internetą.

Studentai galėtų:

- Peržiūrėti jų darbų vertinimus internete
- Pateikti atliktas individualias užduotis dėstytojui per internetą.

Naudodamasis delninių kompiuteriu dėstytojas galės atlikti rutininius darbus daug greičiau ir galės daugiau laiko skirti svarbesniems dalykams.

Programinės įrangos kūrimas, pasirinkti sprendimai, alternatyvus sprendimai pateikti skyriuje „Projektinė dalis“.

2.2. Nagrinėjamos problemos

Šiame darbe nagrinėjame problemas, kurios atsiranda testuojant programinę įrangą skirtą mobiliesiems įrenginiams. Mobilieji įrenginiai su savo specifika sukuria tokias problemas programinės įrangos testavimo procese:

- Ribotas atminties kiekis mobiliame įrenginyje neleidžia jame saugoti testavimo atvejų. Automatiškai generuojant testus yra pagaminami dideli kiekiai testų [7] ir jiems saugoti reikia nemažų atminties resursų, kurių mobiliuosiuose įrenginiuose yra trūkumas.
- Nedideli ekranai mobiliuosiuose įrenginiuose ir ribotos vartojo sąsajos galimybės apsunkina testavimo rezultatų stebėjimą, programinės įrangos derinimą (*debug*) ar kūrimą.
- Dėl mobilių įrenginių įvairovės reikia ištestuoti programinę įrangą daugelyje įrenginių.
- Dėl apribotos vartotojo sąsajos galimybių reikia atkreipti daugiau dėmesio į programinės įrangos panaudojamumo testavimą (*usability testing*).
- Kadangi programinė įranga mobiliuosiuose įrenginiuose dažnai neegzistuoja izoliuota, o bendrauja su įvairiomis aplinkinėmis paslaugomis, kita programine įranga, svarbu atlikti išsamų bendradarbiavimo testavimą.

Darbe nagrinėjamos testavimo problemos sukeltos mobilių technologijų (jų resursų trukumų, vartotojo sąsajos nepatogumo). Konkretizuojamasi ties modulių testavimu (*Unit testing*) [15, 57].

Modulių testų nepavyksta sutalpinti į mobilųjį įrenginį kartu su pačia programine įrangą, nes mobilieji įrenginiai paprastai turi nedidelius atminties resursus. Nors mobilių technologijų atminties resursai laikui bėgant didės, bet tuo pačiu didėja ir jiems skirtos programinės įrangos sudėtingumas. Vystantis mobiliesiems skaičiavimams [18] ir visur esantiems skaičiavimams [45], programinė įranga skirta mobiliesiems įrenginiams bus glaudžiai susijusi su mūsų gyvenimu, kiekviename kasdieninės dienos žingsnyje mus lydės mobilios technologijos. Mobilios technologijos taip pat pradedamos taikyti medicinoje [33]. Programinei įrangai, glaudžiai susijusi su mūsų gyvenimu, turi būti keliami aukšti kokybės reikalavimai. Programinės įrangos testavimas prasideda nuo modulių testavimo, kuris yra glaudžiai susijęs su programos kūrimų (programavimu). Taip pat šis testavimas yra pagrindinė testais paremtos programinės įrangos kūrimo metodo dalis.

2.3. Programinės įrangos testavimas mobiliuosiuose įrenginiuose

Programinė įranga mobiliesiems įrenginiams testuojama pagal įvairiais aspektais [40]:

- Funkcionalumas

  Ar programinė įranga mobiliajame įrenginyje atitinka funkcinius reikalavimus, atlieka teisingai skaičiavimus. Atliekamas modulių, integracinis, sistemos testavimai.
- Suderinamumas

  Ar programinė įranga dirba įvairiose operacinėse sistemose, su skirtingai procesoriais, ribotais resursų kiekiais. Labai svarbu užtikrinti, kad programinė įranga veikia skirtinguose mobiliuosiuose įrenginiuose. Tai aiškiai matoma kai programinė įranga yra skirta plačiam vartotojų ratui, o ne kuriama konkrečiam užsakovui.
- Bendradarbiavimas

  Ar programinį įranga teisingai bendradarbiauja su kitais įrenginiais servisais, teisingai keičiasi informacija. Vystantis mobiliesiems skaičiavimams, visur esantiems skaičiavimams programinę įranga mobiliuosiuose įrenginiuose turi bendrauti su daugeliu aplinkoje esančių sistemų. Testuojant bendradarbiavimą patikrinama ar programinė įranga korektiškai dirba su aplinkinėmis sistemomis.
- Panaudojamumas

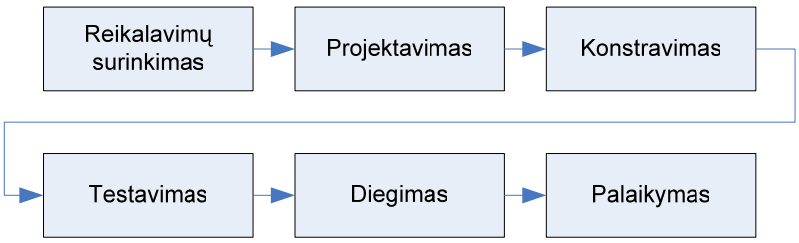
  Ar lengva naudotis programine įranga mobiliuosiuose įrenginiuose esant ribotoms vartotojo sąsajos galimybėmis ir skirtingomis vartotojo sąsajos galimybėmis skirtinguose įrenginiuose.
- Lokalizavimas

  Ar programinė įranga yra pritaikoma darbui įvairiomis kalbomis, esant skirtingam vartotojo sąsajos elementų išdėstymui (iš kairės į dešinę, iš dešinės į kairę)

Šiame darbe detaliai nagrinėjamos problemos susijusios su testavimo faze, kuriant programinę įranga mobiliesiems įrenginiams. Darbe nagrinėjama tematika susijusi su modulių testų [15] vykdymu mobiliuosiuose įrenginiuose.

2.4. Programinės įrangos testavimo vieta programinės įrangos kūrime

Programinės įrangos kūrimas susideda iš kelių fazių: reikalavimų surinkimas, projektavimas, konstravimas, testavimas, diegimas ir palaikymas [1]. Supaprastintas procesas pateiktas 2.1. paveikslėlyje.



2.1 pav. Programinės įrangos gyvavimo ciklas.

Programinės įrangos testavimas paprastai vykdomas po jos konstravimo. Tačiau modulių testavimas paprastai yra atliekamas programuotojų ir vykdomas konstravimo fazėje. Modulių testai sudaromi kartu su programiniais moduliais.

Vystantis Agile Software Development [17] programinės įrangos kūrime, vadovaujamasi principu, kad modulių testai yra sukuriami pirmiau nei patys programiniai moduliai. Toks programinės įrangos kūrimas taip pat yra vadinamas testais paremtas programinės įrangos kūrimas [4].

2.5. Testais paremtas programinės įrangos kūrimas

Dabartiniu metu gana sparčiai populiarėja testavimu paremtas programų kūrimas [14] (*Test Driven Development*). Jo principas yra tas, kad kiekvienam programinės įrangos komponento (objektinio programavimo atveju, klasei), sukuriami testai, kurie ištestuoja komponento korektišką funkcionavimą.

Testais paremtame programinės įrangos kūrime [4, 14], programuotojai kuria testus, kurie vadinami programuotojo testai, kiekvienam programiniam moduliui. Tai atliekama prieš pačio modulio kūrimą. Testų rašymas yra projektavimo veikla, jos metu pateikiami reikalavimai vykdomo programinio kodo pavidalu, kuris gali parodyti, kad programinė įranga veikia pagal jai iškeltus reikalavimus. Programinei įrangai taip augant, pastoviai atliekamas programos kodo perdirbimas [10, 21], tuo pagerinama programinės įrangos architektūra, išmetant pasikartojančius elementus ir patikslinant jos tikslus. Toks perdirbimas gali būti atliekamas nesibaiminantis, kad tai sugriaus programinę įrangą, nes ankščiau paruošti testai užtikrina neteisingo perdirbimo aptikimą. Testai toliau pastoviai vykdomi modifikavus modulį, tuo užtikrinant, kad įvesti pakeitimai nesugriovė modulio. Tokie testai, kurie testuoja programinės įrangos modulius, vadinami modulių testais [57] (*Unit tests*).

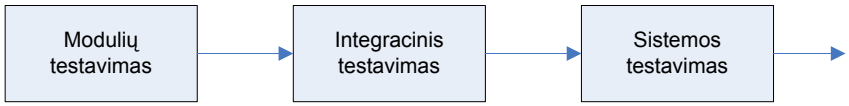
Modulio testas paprastai sukuria testuojamos klasės objektą, kviečia jos metodus su atitinkamais parametrais, tikrina grąžinamų parametrų reikšmes, lygina jas su laukiamais rezultatais. Taip pat patikrina objekto būseną, ar ji perėjo į tokią, kokią tikimąsi.

Testais paremtas programinės įrangos kūrimas pakeičia projektavimo procesą nuo paremto išradimu, kur kūrėjas mąsto, ką programinis modulis turėtų daryti ir po to jį realizuoja, į paremta atradimu, kur kūrėjas prideda mažus funkcionalumo gabalėlius ir vėliau ištraukia struktūrą iš veikiančio programinio kodo.

Testais paremtas programinės įrangos kūrimas turi daug privalumų, bet pagrindinis iš jų yra tas, kad jis verčia programuotoją galvoti apie programos kodo projektavimą atsižvelgiant į jo panaudojimą, negu atsižvelgiant į realizaciją. Testais paremto programinės įrangos kūrimo metu pagaminamas paprastesnis programinės įrangos kodas, nes jis būna orientuotas į esančius reikalavimus. Pagilinus dalykinės srities reikalavimų supratimą, perdirbimo galimybės leidžia pataisyti architektūros klaidas.

2.6. *Modulių testų vieta programinės įrangos testavime*

Modulių testai vaidina svarbų vaidmenį programinės įrangos testavime. Jie paprastai būna patys pirmi testai, kurie atliekami su programinė įranga. Juos sukuria pats programuotojas. Kai kuriose programinės įrangos kūrimo metodologijose reikalaujama iš pradžių sukurti modulių testus, o tik po to kurti pačius programinius modulius [4]. Modulių testavimo vieta programinės įrangos testavime pateikta 2.2. paveikslėlyje.



2.2 pav. Modulių testavimo vieta programinės įrangos testavime

Modulių testai, naudojami testais paremtame programinės įrangos kūrime, užtikrina aukštą programos padengimą testais. Tuo galima teigti, kad visa programinė įranga yra bent jau minimaliai ištestuota.

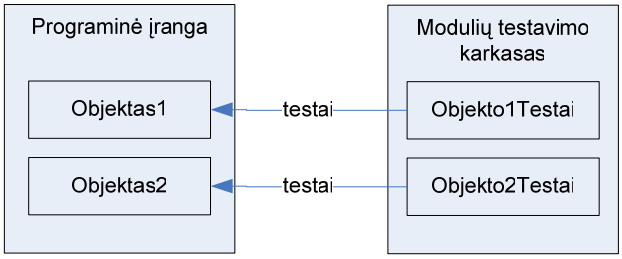
Kaip teigia Michael Feathers savo knygoje “ Working Effectively with Legacy Code“: „Programinis kodas be modulių testų yra liktinis kodas“ [9]. O mes juk nenorime rašyti liktinio kodo.

Modulių testai vaidina svarbų vaidmenį užtikrinant programinės įrangos kokybę, atliekant regresinį programinės įrangos testavimą, jie taip pat yra pagrindinai programinės įrangos kūrimo veiksniai testais paremtose programinės įrangos kūrimo metodologijose.

2.7. Modulių testavimo karkasas

Modulių testavimo karkasai yra programiniai įrankiai, leidžiantys rašyti ir vykdyti modulių testus. Jie taip pat susideda iš pagrindo, kuriuo remiantis kuriami modulių testai, ir testų vykdymo ir rezultatų pranešimo funkcionalumo. Jie nėra skirti vien tik testavimui, jie gali būti lygiai taip pat naudojami kaip programinės įrangos kūrimo įrankiai, tokie kaip kompiliatoriai ar derintojai. Modulių testavimo karkasai gali prisidėti prie beveik visų programinės įrangos kūrimo žingsnių, tokių kaip architektūros kūrimas, kodavimas, derinimas, našumo optimizavimas ir kokybės užtikrinimas.

Modulių testai paprastai yra kuriami lygiagrečiai su programinės įrangos kodu, bet nėra įtraukiami į galutinę programinės įrangos versiją. Modulių testų ir programinės įrangos sąryšis pateiktas paveikslėlyje 2.3.



2.3 pav. Programinė įranga ir modulių testavimo karkasas

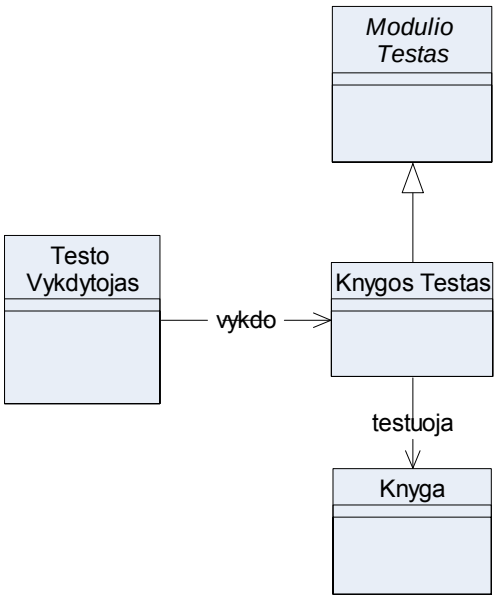
Programinė įranga yra sukonstruojama iš tarpusavyje sujungtų programinių objektų. Modulių testai naudoja programinės įrangos objektus, bet patys egzistuoja modulių testavimo karkaso viduje. Toks būdas turi privalumų. Programinės įrangos kodas nėra sumaišytas su į ją įdėtais modulių testais. Sukompiliuotos programos dydis bus mažesnis. Modulių testai gali būti vykdomi atskirai nuo programinės įrangos, t.y. objektai gali būti izoliuotai ištestuoti.

Vienas modulio testas skirtas testuoti tam tikrą programinės įrangos objekto elgseną. Jo sėkmė ar nesėkmė patikrina vieną programinės įrangos kodo modulį. Gerai parašyti testai nustato pradines objekto sąlygas, aplinkos sąlygas ar scenarijų, kuris yra nepriklausomas nuo visų kitų sąlygų, tada atlieka vieną veiksmą ir patikrina apibrėžtą rezultatą. Rašant tokius testus, turėtų būti vengiama priklausomybės nuo kitų testų rezultatų, jie turėtų būti trumpi ir paprasti. Pradedant kurti modulių testus, testuojančius paprasčiausią programinės įrangos funkcionalumą, po to tolygiai kuriant testus vis sudėtingesniems objektams ir jų elgsenai, modulių testavimo karkasas gali būti panaudotas testuoti programinę įrangą turinčią ir labai sudėtingą architektūrą. Kurti tokį modulių testavimo karkasą yra ne tik lengviau nei kurti atskirai testus, bet tuo pačiu, jo kūrimo metu sukuriami išsamesni ir efektyvesni testai. Turint

išsamų modulių testų kiekį, galima greitai kurti programinę įrangą, nes kiekvienas jos pakeitimas gali būti greitai ir pilnai patikrintas.

Modulių testai paprastai yra laikomi baltos dėžės testais. Jie pasiekia programinio objekto (klasės) vidinę struktūrą. Daugelis objektiškai orientuotų programavimo kalbų teikia priemones uždrausti priėjimą prie objektų vidinės struktūros (vidinių laukų). Modulių testai tada testuoja tik tai, ką galima pasiekti per viešai prieinamą objekto sąsają. Dėl šios priežasties yra svarbu kurti ir pačią programinę įrangą, atsižvelgiant į tai, kad ją būtų lengva testuoti, kas yra laikoma gera objektiškai orientuoto programinės įrangos kūrimo praktika.

Modulių testavimo karkasas paprastai susideda iš kelių dalių. Modulio testavimo karkaso pavyzdys pateiktas 2.4 paveikslėlyje.



2.4 pav. Paprasto modulių testavimo karkaso klasių diagrama

Modulių testavimo karkasas susideda iš kelių dalių:

- Modulio testas yra abstrakti klasė, kuri apibrėžia modulio testo sąsają.
- Modulio testas yra klasė paveldinti Modulio Testas klasę ir realizuojanti jos apibrėžtą sąsają, ir atliekanti testuojamos klasės testavimą. Modulio testo, kuris atlieka klasės Knyga testavimą, pavyzdys pateiktas 2.5(a) paveikslėlyje.
- Testų vykdytojas yra klasė, kuri analizuoja programinę įrangą, suranda modulių testus ir vykdo jų testavimo metodus, įvykdžius jų metodus, formuoja testavimo ataskaitą. Jo pavyzdys pateiktas 2.5 (c) paveikslėlyje.

- Testuojama klasė yra paprasta programinės įrangos klasė, kurią naudoja modulio testas ir tikrina ar jos realizacija yra korektiška. Jos pavyzdys pateiktas 2.5 (c) paveikslėlyje.

KnygosTestas.java

```
public class KnygosTestas extends ModulioTestas
{
        public void runTest( ) throws Exception
        {
                Knyga knyga = new Knyga("Lia");

                assertTrue(knyga.title.equals("Lia"), "tikrianame pavadinimą");
        }
}
```

(a)

Knyga.java

```
public class Knyga
{
   public String title = "";

   Knyga(String title)
   {
   }
}
```

(b)

TestųVykdytojas.java

```
public class TestųVykdytojas
{
   public static void main(String[] args)
   {
      TestųVykdytojas tester = new TestųVykdytojas( );
   }

   public TestųVykdytojas ( )
   {
      try
      {
         ModulioTestas test = new KnygosTestas( );

         test.runTest( );

         System.out.println("PAVYKO!");

      }

      catch (Exception e)
      {
         e.printStackTrace( );

         System.out.println("KLAIDA!");
      }
   }
}
```

(c)

2.5 pav. Modulių testavimo karkaso pavyzdinė realizacija

Modulių testavimo karkasai yra esminiai elementai populiariose programinės įrangos kūrimo metodologijose, tokiose kaip extreme programming (XP) [35] ir Agile Development [17]. Modulių testavimas jau išėjo iš extreme programming metodologijos ribų ir yra dabar dažnas įvairiose kitose programinės įrangos kūrimo metodologijose. Modulių testais galima užtikrinti žemiausio lygio programinės įrangos kodo teisingumą, sumažinti programinės įrangos kūrimo ciklo trukmę, padidinti produktyvumą ir sukurti labiau patikimą programinę įrangą.

Modulių testai gali būti sukurti rankomis atidžiai nagrinėjant reikalavimų specifikacijas, realizuotus programinius modulius. Taip pat modulių testams galima naudoti automatinius testų generavimo būdus [7, 23]. Automatiškai generuoti modulių testai sudaro žymiai daugiau programinio kodo nei pati testuojama programinė įranga. Todėl atsiranda atminties sunaudojimo problemos. Tai ypač aktualu programinei įrangai skirtai mobiliesiems įrenginiams.

2.8. Resursų augimo tendencijos mobiliuosiuose įrenginiuose

Mobilių telefonų ir delninių kompiuterių resursai pastoviai auga. Pradžioje jie teturėdavo tik po kelis kilobaitus atminties. Dabar situacija pasikeitė, mobilieji telefonai turi megabaitus atminties, o delniniai kompiuteriai - dešimtis megabaitų atminties. Šiuo metų

rinkoje populiarūs delniniai kompiuteriai, turintys 32-64 MB atminties, bei mobilieji telefonai, turintys 1-2 MB atminties.

Nors atminties resursai ir procesoriaus greičiai mobiliuosiuose įrenginiuose pastoviai auga, visai kitokia situacija yra su vartotojo sąsaja. Vartotojas turi įvesti informaciją naudodamasis mobiliuoju įrenginiu. Informacija gali būti įvedama:

- Klaviatūra (rodoma ekrane, ar klavišai ant įrenginio).
  Klaviatūros būna per mažos įvesti duomenims. Padidinti neina, nes padidėja ir pats įrenginys. Galima naudoti išorines, papildomai prijungiamas klaviatūras.
- Rašant ranka pieštuku ekrane.
  Žmogui patogu būtų tekstą įvedinėti tiesiog jį užrašant ranka. Bet daugelio įrenginių procesoriai dar per lėti, kad galėtų atpažinti tekstą.
  Vartotojui gali tekti išmokti nauja abėcėlę kai tekstas užrašomas gestais.
- Diktuoti žodžiu.
  Reikia greito procesoriaus.
  Prastai veikia triukšmingose aplinkose.

Mobilieji įrenginiai turi nedidelius, bet spalvotus ir nemažos raiškos ekranus. Tačiau jų nedideli dydžiai neleidžia patogiai išversti duomenų. Žinoma, galimi sprendimų būdai būtų tokie kaip ekranų didinimas, bet jei didinsime ekranus mobiliuosiuose įrenginiuose, jie taps nemobiliais. T. y. mobilusis įrenginys bus mobilus tik tol, kol jo dydis bus pakankamai mažas.

2.9. Mobiliųjų įrenginių programinės įrangos sudėtingumo, dydžio augimas

Programinė įranga mobiliuosiuose įrenginiuose pradžioje buvo gana paprasta. Ji apsiribojo tik užrašų knygutėmis. Dabar pradedamos įgyvendinti mobilių skaičiavimų ir visur esančių skaičiavimų idėjos, mobilieji įrenginiai pradedami naudoti medicinoje. Remiantis visur esančių skaičiavimų idėja, programinės įrangos kiekis mobiliesiems įrenginiams labai išaugs, jos sudėtingumas taip pat žymiai padidės. Mobilios technologijos mus lydės kiekviename mūsų gyvenimo žingsnyje ir bus glaudžiai susijusios su kasdienine mūsų veikla. Dėl mobilių technologijų glaudaus susiejimo su mūsų gyvenimu, bus keliami ir aukšti kokybės reikalavimai programinei įrangai.

Techninės įrangos pajėgumų didėjimas leidžia programinės įrangos sudėtingumui didėti sparčiais tempais, tačiau programinės įrangos testavimas nespėja judėti tais pačiais

tempais [50]. Tad išlieka testavimo problemos susijusios su ribota vartotojo sąsaja ir ribotais atminties kiekiais.

2.10. Siūlomi sprendimo būdai

Sprendžiant resursų trūkumo ir nepatogios vartotojo sąsajos problemas, buvo pasiūlyti keli variantai, efektyviai atlikti testavimą mobiliuosiuose įrenginiuose, išvengiant jų apribojimų. Vienas iš jų yra siūlymas naudoti mobilių įrenginių emulatorius [46]. Šis variantas leidžia sukurti mobilųjį įrenginį paprastame kompiuteryje, įraukiant kiek reikia resursų. Tokiu būdu galima turėti didelį kiekį atminties, taip pat patogias vartotojo sąsajos galimybes. Tačiau šio sprendimo trūkumas yra tas, kad emulatorius bus tos pačios architektūros kaip ir paprastas kompiuteris, kuris daugeliu atvejų skirsis nuo mobiliajame įrenginyje esančios. Taip pat resursų padidinimas neatspindės realios situacijos mobiliuosiuose įrenginiuose ir neleis pilnai ištestuoti (pavyzdžiui atminties trūkumo situacijos).

Taip pat siūlomas variantas naudoti automatinį perteklinių testų pašalinimą [51]. Naudojant automatinius testų generavimo įrankius sugeneruojami dideli kiekiai testų, iš kurių daugelis yra pertekliniai, tad siūloma juos aptikti ir pašalinti. Šis sprendimas būtų gana efektyvus, bet jis visiškai nenaudingas, jei testų kiekis yra didelis ir perteklinių nėra.

Sekančiuose skyreliuose detaliau aptariamas kiekvienas iš siūlomų problemų sprendimo būdų.

2.10.1. Emuliatoriaus sprendimo būdas

Vienas iš būdų išspręsti resursų trūkumo problemas mobiliuosiuose įrenginiuose testavimo metu yra naudoti emuliatorius [46].

Emuliatorius, bendrąja prasme, dubliuoja vienos sistemos funkcijas naudojantis kita sistema, tuo padarant, kad antros sistemos elgsena būtų tokia pati kaip ir pirmosios.

Teoriškai galima teigti, kad bet kokia darbinė aplinka gali būti emuliuota naudojant bet kokią kitą aplinką. Praktiškai tai atlikti gali būti sudėtinga, ypatingai tuo atveju, kai emuliuojamos sistemos elgsena nėra tiksliai dokumentuota arba ji yra nustatyta remiantis atvirkštinės inžinerijos metodais. Taip pat emuliatoriai dažniausiai nedirba tokia pačia sparta kaip ir emuliuojamos sistemos. Emuliatoriai paprastai emuliuoja techninę įrangą, tokią kaip procesorius, atminties posistemė, įvedimo/išvedimo įrenginiai, o juose įdiegiama originali operacinė sistema.

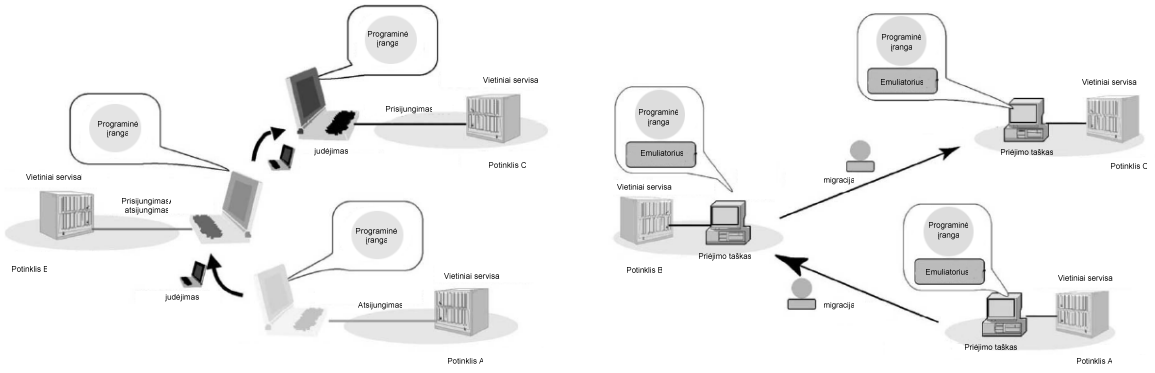
Šis pasiūlytas būdas siūlo vietoje realių mobilių įrenginių naudoti emuliatorius, atliekant programinės įrangos testavimą mobiliuosiuose įrenginiuose. Emuliatoriaus pagalba sukuriama mobilaus įrenginio imitacija, kuri beveik atitinka realų įrenginį. Testuojama programinė įranga būna patalpinama į emuliatorių ir jame testuojama. Emuliatorius paprastai yra vykdomas staliniame kompiuteryje. Kita vertus, mobilieji įrenginiai pasižymi būtent ta mobilumo savybę. Jie juda įvairiai aplinkoje, bendrauja su kitomis sistemomis, pereina iš vienos aplinkos į kitą. To padaryti emuliatorius negali, nes jis yra pririštas prie vieno kompiuterio, kuriame yra emuliuojamas mobilus įrenginys.

Autoriai pasinaudoja mobilių agentų [27, 32] idėja, kuri teigia, kad programinė įranga gali migruoti tarp įvairų tinklo resursų. T. y., programa yra vykdoma viename kompiuteryje, vėliau jos vykdymas gali būti laikinai sustabdytas, programa su jos būsena tinklu perkelta į kitą tinklo kompiuterį ir jos vykdymas toliau pratęstas.

Autoriai pateikia dvi mobilumo sąvokas:

- Fizinis mobilumas.
- Loginis mobilumas.

Fizinio mobilumo atveju programinė įranga juda aplinkoje su pačiu mobiliu įrenginiu. Loginio mobilumu atveju tinklu juda tik programinė įranga. Judėjimas realizuojamas naudojantis mobiliais agentais [27, 32]. Fizinis ir loginis mobilumas vaizdžiai pateikti paveikslėliuose 2.6 (a) ir 2.6 (b).



(a) (b)

2.6 pav. Fizinis (a) ir loginis (b) mobilumas

Emuliatorius turi būti sukurtas kaip mobilaus agento programinė įranga. Tada emuliatorius patalpinamas į mobilaus agento infrastruktūrą (programinė įranga, kuri valdo

mobilaus agento darbą, judėjimą tinklų, yra atsakinga už mobilaus agento laikiną sustabdymą, paleidimą, išjungimą), o į emuliatorių patalpinama programinė įranga, testavimo karkasas, testai ir paleidžiama. Veikimo metu mobilusis agentas migruoja po tinklą, tuo sudarydamas realaus mobiliojo įrenginio judėjimą aplinkoje.

Naudojant emuliatorių galima emuliuoti reikiamą mobilųjį įrenginį ir atmesti jo apribojimus. T. y., emuliuoti įrenginį ir priskirti emuliatoriui didesnį atminties kiekį, tuo išsprendžiant testų sutalpinimo problemas. Taip pat emuliuoti mobilųjį įrenginį su patogesne vartotojo sąsaja (didesniu ekranu), tuo išsprendžiant testų stebėjimo problemą.

Tačiau pakeitus mobiliojo įrenginio charakteristikas emuliatorius nebeatitiks emuliuojamo realaus mobiliojo įrenginio galimybių. Vartotojo sąsaja gali visiškai pasikeisti (pavyzdžiui, išnykti tarpiniai langai ir pan.), o tai gali įtakoti ir testų vykdymą, kurie yra pritaikyti konkrečiai mobiliojo įrenginio vartotojo sąsajai. Taip pat padidinus atminties resursus, testai gali neaptikti situacijų, kai viršijamas atminties sunaudojimas mobiliajame įrenginyje (tektų modifikuoti testus, atsižvelgiant į emuliatoriaus galimybes). Taip pat iškyla kita problema, susijusi su tuo, kad daugelis emuliatorių nevisiškai emuliuoja mobilius įrenginius. Dažnai pasitaiko, kad, pavyzdžiui, mobiliojo įrenginio procesorius yra ARM architektūros, o emuliatorius emuliuoja x86 architektūros įrenginį. Šiuo atveju tektų pritaikyti testuojamą programinę įrangą ir modulių testavimo karkasą kitai procesoriaus architektūrai. Be to, įvedus papildomą lygį programinėje įrangoje (šiuo atveju emuliatorių), atsiranda ir daugiau galimybių pasireikšti programinės įrangos klaidoms. Jei testavimas vykdomas mobiliajame įrenginyje, klaidas gali įtakoti tik mobiliojo įrenginio techninė įranga, operacinė sistema ir testuojama programinė įranga. O naudojant emuliatorius, atsiranda dar emuliatoriaus ir operacinės sistemos, kurioje vykdomas emuliatorius, lygiai, kurie savo ruožtu taip pat gali turėti klaidų.

Privalumai:

- Patogus ir lengvai kontroliuojamas mobilumas (ypač atliekant daugkartinį testavimą).
- Išsprendžiama atminties trūkumo problema.
- Išsprendžiama testavimo rezultatų stebėjimo problema.

Trūkumai:

- Emuliuotas įrenginys gali pilnai neatitikti realaus mobilaus įrenginio.
- Emuliatorius gali turėti savo klaidų.
- Vartotojo sąsajos testavimas neatitiks realios situacijos.
- Mobiliųjų agentų programinė įranga pati gali turėti klaidų [55].

2.10.2. Perteklinių testų pašalinimas

Modulių testai objektinei (*Object-Oriented*) programinei įrangai susideda iš metodų iškvietimų sekų. Objekto elgesys priklauso nuo kviečiamo metodo parametrų ir objekto būsenos. Egzistuojantys automatiniai modulių testų generavimo įrankiai ignoruoja būseną ir sugeneruoja daug perteklinių testų, kurie išbando tą pačią metodų elgseną, tuo prailgina testų vykdymo laiką ir padidina atminties sunaudojimą testams saugoti.

Tao Xie, Darko Marinov ir David Notkin pasiūlė būdus, kurie aptinka ir pašalina perteklinius testus[51]. Autoriai pasiūlė Rostra [51] metodiką, kuri aptinka perteklinius testus. Ją siūloma integruoti į automatinių testų generavimo įrankius, kad jie tiesiog generavimo metu atmestų perteklinius testus.

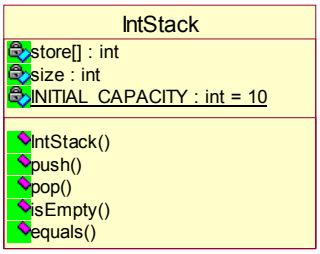
Paveikslėlyje 2.7 pateikiami pertekliniai testai, kurie testuoja klasę IntStack.

```
Test 1 (T1):
       IntStack s1 = new IntStack();
       s1.isEmpty();
       s1.push(3);
       s1.push(2);
       s1.pop();
       s1.push(5);

Test 2 (T2):
       IntStack s2 = new IntStack();
       s2.push(3);
       s2.push(5);

Test 3 (T3):
       IntStack s3 = new IntStack();
       s3.push(3);
       s3.push(2);
       s3.pop();
```

(a)



(b)

2.7 pav. Testuojama klasė (b) ir jos modulių testai (a)

Paveikslėlyje 2.7 (a) pateikti testai T2 ir T3 yra pertekliniai, nes atlieka tą patį testavimą kaip ir testas T1. Kiekvienas modulio testo vykdymas susideda iš objekto būsenos perėjimų. Perėjimai įvyksta iškviečiant objekto metodus. Kiekvienas metodo vykdymas yra apibūdinamas kaip vykdomo metodo vardas, metodo parametrai ir esama objekto būsena. Dviejų metodų vykdymas yra ekvivalentiškas jei jų pavadinimai sutampa, objektas yra toje pačioje būsenoje ir perduodami tokie patys parametrai.

Autoriai siūlo penkias technikas, kurie palaipsniui pašalina perteklinius modulių testus, skirtus objektiškai orientuotai programinei įrangai:

- WholeSeq

Testai yra laikomi ekvivalenčiais, jei jie yra identiški. Testas vaizduojamas kaip objekto būsenų ir metodų vykdymo seka. Jei dviejų testų sekos yra vienodos, tai tie testai yra laikomi ekvivalenčiais ir vieną iš jų reikia pašalinti kaip perteklinį. Ši technika pavyzdyje pateiktuose testuose perteklinių testų nerado.

- ModyfyingSeq

  Objekto būsenų ir metodų vykdymo seka yra sudaroma įtraukiant tik tuos metodus, kurie modifikuoja objekto būseną. Tad atliekamas sekų palyginimas ir testai, kurie turi tokias pačias sekas, yra laikomi ekvivalentiški. Ši technika pavyzdyje pateiktuose testuose aptinka, kad testas T3 yra perteklinis T1 testo atžvilgiu.

- WholeState

  Lyginamos objekto būsenos esančios po kiekvieno metodo vykdymo ir sudaroma jų perėjimo seka. Jei sekos sutampa, testai laikomi ekvivalenčiais. Ši technika pavyzdyje pateiktuose testuose aptinka, kad testas T3 yra perteklinis T1 testo atžvilgiu, tačiau ji neaptinka, kad testas T2 yra perteklinis T1 testo atžvilgiu.

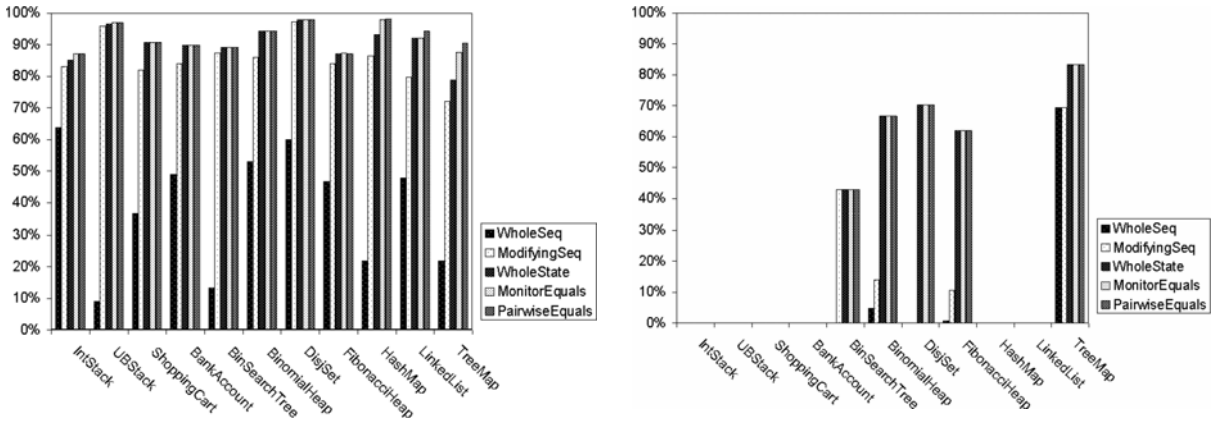
- MonitorEquals

  Ši technika veikia panašiai kaip ir WholeState technika, tik analizuojamos objekto būsenos parametrai, kurie yra nagrinėjami to testo vykdymo metu. Jei būsenos nagrinėjami parametrai sutampa, tai būsenos laikomos ekvivalenčiomis. Ši technika pavyzdyje pateiktuose testuose aptinka, kad testas T3 yra perteklinis T1 testo atžvilgiu, taip pat aptinka, kad testas T2 yra perteklinis T1 testo atžvilgiu.

- PairwiseEquals

  Technika veikia panašiai kaip ir MonitorEquals tik, kad objektų būsenos lyginamos paporiui. Pavyzdžiui, jei objekto būseną sudaro masyvas ir vienoje būsenoje masyvo reikšmės yra < 1, 2, 3 >, o kitoje < 3, 2, 1 >, tai kai kuriais atvejais galima teigti, kad būsena yra ta pati. Ši technika pavyzdyje pateiktuose testuose aptinka, kad testas T3 yra perteklinis T1 testo atžvilgiu, taip pat aptinka, kad testas T2 yra perteklinis T1 testo atžvilgiu.

Kad pavaizduoti technikų galimybes, autoriai pateikė įrankių JTest ir JCrasher automatiškai sugeneruotų modulių testų kiekio ir jų perteklinių testų kiekių procentinį santykį, kuris buvo apskaičiuotas naudojant minėtas technikas jiems aptikti [51]. Įvertinimo rezultatai pateikti paveikslėliuose 2.8 (a) JTest ir 2.8 (b) JCrasher.



2.8 pav. Perteklinių testų procentinis santykis tarp įrankių sugenruotų testų

Privalumai:

- Pašalina daug perteklinių testų.
- Sumažina testams saugoti reikalingą atminties kiekį.

Trūkumai:

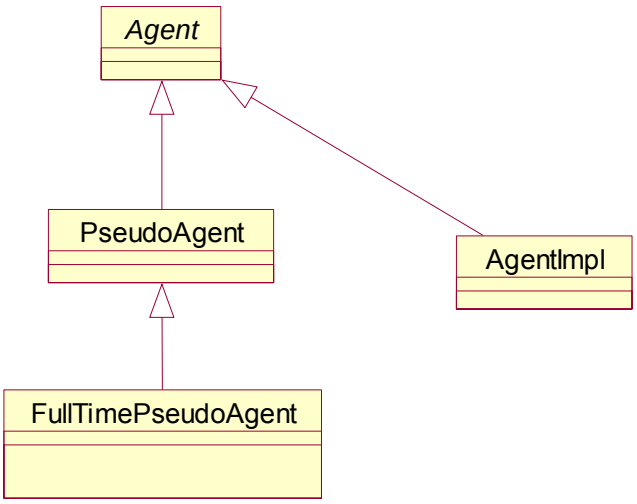
- Neveiks nebesant pertekliniams testams.
- Neišsprendžia stebėjimo problemos.
- Vykdant regresinį testavimą, testai gali neaptikti naujų situacijų, kurios gali būti įtakotos pasikeitimų programinės įrangos kode.

2.10.3. Mock objektai

Modulio testas paprastai susideda iš testuojamo objekto metodų iškvietimų sekos. Sekos pavyzdys pateiktas 2.7 (a) paveikslėlyje. Testuojamas objektas turi paprastai įvairių metodų, kurių nereikia išbandyti konkrečiame modulio teste. Autoriai siūlo naudoti Mock objektus vietoj realių objektų, tuo pašalinant netestuojamus metodus konkrečiame modulio teste.

Mock objektas savo prasme yra labai panašus į kamščius. Bet jo esminis skirtumas yra, kad jis pateikia kažkokią tai metodų realizaciją, kuri imituoja arba realiai realizuoja reikiamą funkcionalumą [47]. Mock objektų pagalba galima pagreitinti testavimo procesą, pavyzdžiui, realizavus netikrą metodą, kuris turėtų tik vieną kartą iškvietus suveikti, o kviečiant kitą kartą – ne. Pirmas to metodo kvietimas atliktų ilgus skaičiavimus ir gražintų rezultatą. Jei tokį objektą pakeistumėme Mock objektu, užtektų realizuoti, kad pirmą kartą grąžina kažkokį tai rezultatą, neatlikdamas skaičiavimų, o sekančius kartus nesuveikia [29].

Naudojant Mock objektus iš modulio teste naudojamo metodo pašalintumėme nereikalingus metodus, tuo supaprastindami testuojama objektą. Automatinio testų generavimo metu būtų sugeneruojama mažiau metodų kvietimų sekų, reikalingų ištestuoti objekto funkcionalumą. Testuojamo objekto Mock objekto realizacijos pavyzdys pateiktas 2.9 paveikslėlyje.



2.9 pav. Klasių santykis

Paveikslėlyje pateiktame pavyzdyje, turime apibrėžtą objekto sąsają, tada ją realizuojame Mock objektu, kuris nenaudojamų metodų realizacijas pateikia kaip išskirtinės situacijos (*Exception*) sudarymą, kuri praneša, kad metodas nerealizuotas. Metoduose, kurie naudojami testavimo metu, pateikiama jų realizacija arba realizacija imituojanti jų veikimą.

Mock objektų sukūrimas gali būti realizuotas naudojant automatinius įrankius [11]. Vienas iš tokių įrankių yra JMock [11].

Privalumai:

- Sumažinamas testų kiekis.
- Reikia mažiau atminties testams saugoti.

Trūkumai:

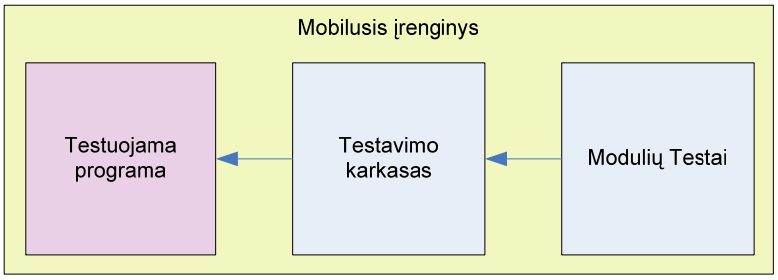
- Rankiniu būdų reikia realizuoti kai kurias Mock objektų dalis.
- Neišsprendžia stebėjimo problemos.

2.10.4. Nuotolinis modulių testavimas

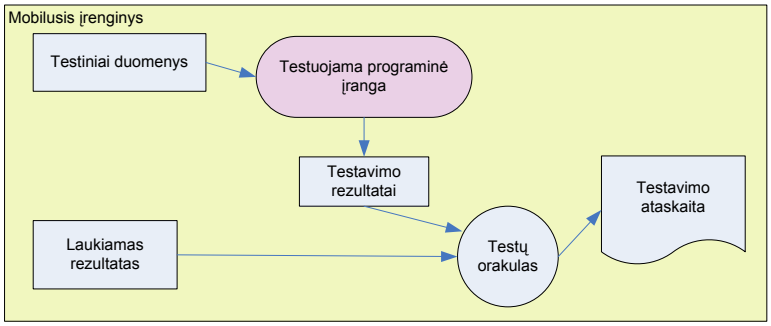
Modulių testai yra kuriami programuotojų kartu su pačiu programiniu kodu, kurį modulių testai testuoja. Paprasčiausiu atveju modulių testai ir programinis kodas gali būti patalpinti į tą pačią programą kartu (į tą patį vykdomą modulį ar biblioteką). O vėliau, kuriant galutinę programinės įrangos versiją, testai pašalinami iš programos (naudojant preprocesoriaus makro direktyvas ir pan.).

Šiuo metu siūloma atskirti modulių testus nuo testuojamo programinio kodo [19], patalpinant testus į atskiras bibliotekas (.dll, .so ir pan.). Atskirtas kodas yra aiškesnis, diegimo metu užtenka tik neįtraukti bibliotekų su testais, nereikalingas programinės įrangos kodo perkompiliavimas.

Paprastai modulių testai ir programinė įranga yra sudedama į vieną programą (testai įkompiliuojami) ir atliekant testavimą vykdoma ta pati programa, tik kviečiamas ne jos vykdymas, o testai, esantys joje. Tokio testavimo atveju visas modulių testavimo karkasas turi būti patalpintas į mobilųjį įrenginį ir vykdomas jame. Mobilaus įrenginio modulių testavimo karkasas pateiktas 2.10 paveikslėlyje.



(a)



(b)

2.10 pav. Modulių testavimo karkasas

Paveikslėlyje 2.10 (a) matome modulių testavimo karkaso principinę schemą nesigiliant į jo realizaciją. Testai nors ir yra atskirti nuo testuojamos programinės įrangos, jie vis tiek yra mobiliajame įrenginyje. Paveikslėlyje 2.10 (b) pateikta modulių testavimo karkaso struktūra.

Joje matome, kad rezultatų palyginimas, ataskaitos formavimas ir vaizdavimas atliekami mobiliajame įrenginyje.

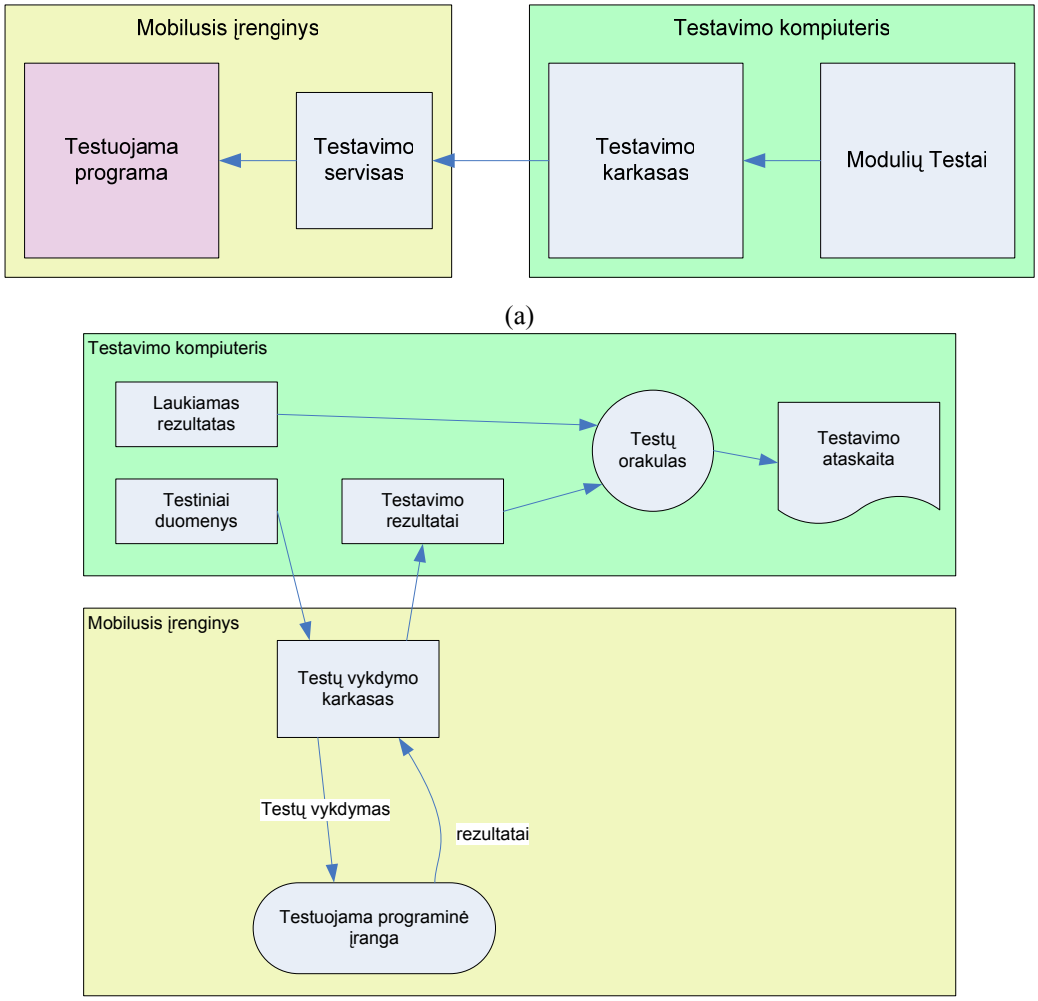
Pagrindinis trūkumas, kad viskas turi būti sutalpinta į mobilųjį įrenginį. Deja, mobiliuosiuose įrenginiuose atminties resursai yra riboti ir pakankamo testų kiekio nepavyksta sutalpinti. Taip pat dėl ribotų vartotojo sąsajos galimybių testų rezultatų stebėjimas yra problematiškas.

Modulių testų atskyrimas, išskaidant programą į bibliotekas, padeda palengvinti programos diegimo procesą, palengvina kūrimo ir palaikymo procesus. Tačiau, atskyrus taip testus nuo programinės įrangos, testai vis tiek bus patalpinti mobiliajame įrenginyje, modulių testavimo karkasas irgi bus mobiliajame įrenginyje, testavimo rezultatų vaizdavimas taip pat bus atliekamas mobiliajame įrenginyje.

Tad testų atskyrimas palengvina diegimo, kūrimo ir palaikymo procesus, bet nepadeda išspręsti vietos sunaudojimo ir rezultatų stebėjimo problemos. Šioms problemoms išspręsti siūlome atskirti modulių testus nuo programos, atskiriant juos ne tik į atskiras bibliotekas, bet ir iškeliant testus iš mobilaus įrenginio į kitą kompiuterį.

Mobiliajame įrenginyje turi būti paliktas tik testų vykdomo karkasas, kurio paskirtis būtų gauti reikiamą testą, jį įvykdyti ir grąžinti rezultatus. Paprasčiausiu atveju tai būti tiesiog funkcija, kuri per parametrus gauna testą, testuojamo objekto vardą, metodą, paduoda testus testuojamo objekto metodui, pasiima vykdymo rezultatus ir gražina į testavimo serverį. Visi testavimo duomenys, laukiami rezultatai, rezultatų formavimas, atvaizdavimas būtų atliekami išorėje tuo nenaudojant mobilaus įrenginio resursų.

Paveikslėlyje 2.11 pateiktas nuotolinio modulio testavimo karkaso struktūra.



2.11 pav. Nuotolinis modulių testavimo karkasas

Palyginus 2.10 ir 2.11 paveikslėliuose pateiktus modulių testavimo karkasus matome nuotolinio modulio testavimo [38] karkaso idėją. Taupydami vietą mobiliajame įrenginyje ir pagerindami testavimo rezultatų stebėjimo galimybes, iš mobilaus įrenginio iškeliame nereikalingas dalis (laukiami rezultatai, testiniai duomenys, testų orakulas, testavimo ataskaitos formavimas) ir paliekame tik testuojamą programinę įrangą ir supaprastintą testavimo karkaso versiją. Toks variantas reikalauti žymiau mažiau atminties resursų mobiliajame įrenginyje ir leidžia patogiau peržiūrėti testavimo rezultatus testavimo kompiuteryje.

Toks testavimo karkaso pakeitimas pakeičia ir modulių testų vykdymo procesą. Jis galėtų vykti tokiais žingsniais:

1. Iš testinių duomenų sąrašo imamas vienas testinis atvejis (vienas testas, su laukiamu rezultatu).
2. Jis siunčiamas testų vykdymo karkasui, mobiliajame įrenginyje.
3. Mobiliajame įrenginyje įvykdomas testas.

4. Grąžinamas testo vykdymo rezultatas iš mobilaus įrenginio
5. Sulyginamas gaunas rezultatas su laukiamu rezultatu.
6. Imamas sekantis testinis atvejis.
7. Kartojamas 2-6 žingsniai, kol nesibaigė testiniai atvejai.
8. Formuojama testavimo ataskaita.

Testinio atvejo atvaizdavimas gali būti realizuotas kelias būdais:

- Perduodant objekto vardą, testuojamo metodo vardą ir parametrus metodui.
  Šiuo būdų Testų vykdymo karkasas mobiliajame įrenginyje turėtu mokėti sukurti testuojamos programinės įrangos nurodytą objektą ir iškviesti nurodytą metodą. Šiai galimybei realizuoti būtu reikalinga galimybė matyti programinės įrangos meta duomenis ir aplinka turėtu teikti atspindžių (*Refelctions*) [3] galimybes.
- Generuojat testavimo kodą vienam testiniam atvejui serverio pusėję ir perduodant testų vykdymo karkasui mobiliajame įrenginyje.
  Šiuo būdų testavimo karkaso dalis serverio pusėje turėtu sukompiliuoti programos vieneto testinio atvejo kodą, kuris turėtų būti pritaikytas prie konkrečios procesoriaus architektūros esančios mobiliajame įrenginyje. Paruošus kompiliuotą, kodą jis turėtų būti persiunčiamas į mobilųjį įrenginį ir vykdomas.

Nuotolinis modulių testavimo būdas padeda išspręsti problemas susijusias su resursų apribojimais, nes testai saugomi kompiuterio pusėje, rezultatų atvaizdavimas taip pat atliekamas kompiuterio pusėje. Tačiau vienas iš šio būdo minusų yra, kad platforma turi palaikyti atspindžius (*Reflections*) [3], ir tinko gedimai gali įtakoti testavimo procesą. Tinklo gedimams aptikti reiktų realizuoti tam tikras priemones, kurios aptiktų tinklo gedimą ir perleistų testavimo atvejį iš naujo.

Privalumai:

- Galime testuoti netgi programinės įrangos eksploatacijos metu, jei paliksime testų serverį.
- Patogus testavimo rezultatų stebėjimas
- Reikalauja nedaug atminties testams saugoti mobiliajame įrenginyje.

Trūkumai:

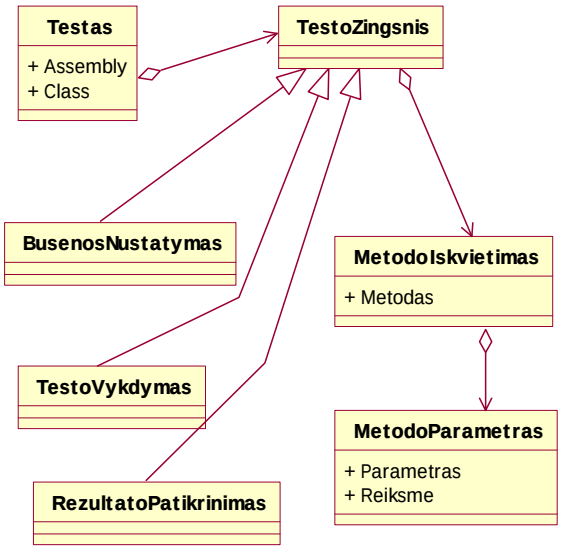
- Tinklo gedimas gali sutrikdyti testavimo procesą.
- Lėtesnis testavimo vykdymas.

Iškėlus modulio testus ir modulių testavimo karkasą į kitą kompiuterį, reikia apibrėžti kaip testai perduodami į mobilųjį įrenginį. Numatėme du variantus:

- Naudojant universalų valdiklį.
- Naudojant konkretų valdiklį.

2.10.4.1. Universalaus valdiklio būdas

Universalaus valdiklio variantu, testavimo servisas mobiliajame įrenginyje priima duomenis, kurie deklaratyviai aprašo modulio testą. Duomenų schema pateikta 2.12 paveikslėlyje.

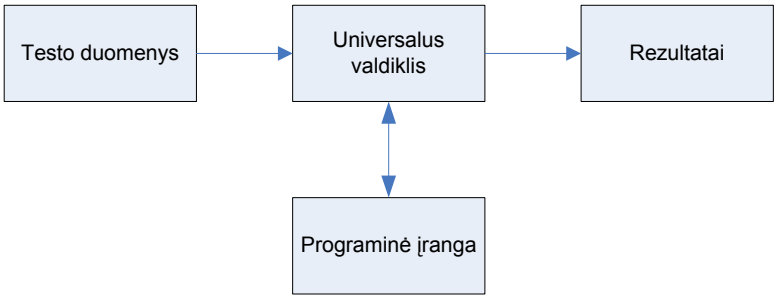


2.12 pav. Modulio testo duomenų schema

Duomenyse nurodomas testuojamo objekto vardas, veiksmų seka reikalinga nustatyti objektui pradinę būseną, Testo vykdymo veiksmų seka, ir rezultato tikrinimo veiksmų seką. Pradinės būsenos nustatymo veiksmai sudaryt iš metodų iškvietimų, kurie nustato objekto būseną. Testo vykdymo veiksmų seka susideda iš metodų iškvietimų, kurie atlieka testo metu numatytas operacijas su testuojamu objektu. Rezultatų tikrinimo veiksmų seka susideda iš

metodų iškvietimų, kurie iškviečia metodus, kurie gražina objekto būsenos reikšmes. Metodo iškvietimas susideda ir metodo vardo ir perduodamos parametrų reikšmių (pvz.: Sumuoti, x, y). Parametras susideda iš jo vardo, tipo ir perduodamos reikšmės (pvz.: x , float, 10.1).

Testų vykdymo metu kiekvieno testo duomenys yra persiunčiami į mobilųjį įrenginį, juos priima testavimo serveris ir perduoda juos universaliam valdikliui. Universalaus valdiklio struktūra ir veikimas pateikti 2.13 paveikslėlyje.



2.13 pav. Testavimo karkaso mobiliajame įrenginyje struktūra

Universalus valdiklis, gavęs testo duomenis, sukuria nurodytos klasės objektą, įvykdo objekto pradinės būsenos nustatymo veiksmus, vykdo testavimo veiksmus ir gauna veiksmų vykdymo rezultatus.

Norint iškviesti nurodytus metodus reikia, kad testuojama programinė įranga būtų realizuota naudojant technologijas, kurios palaiko atspindžius (*reflections*) [3] ir tipų identifikavimą vykdymo metu (RTTI) [20]. Labiausiai paplitusios technologijos pasižyminčiomis šiomis savybėmis yra Java ir Microsoft .NET.

Visas testavimo procesas vyksta taip:

1. Ištenkamas vienas modulio testas.
2. Testo duomenys siunčiami tinklu testavimo serveriui į mobilųjį įrenginį.
3. Testavimo serveris mobiliajame įrenginyje perduoda testus universaliam valdikliui.
4. Universalus valdiklis sukuria testuojamą objektą.
5. Universalus valdiklis nustato pradinę testuojamo objekto būseną.
6. Universalus valdiklis vykdo testo vykdymo seką.
7. Universalus valdiklis vykdo rezultatų tikrinimo veiksmų seką.
8. Universalus valdiklis perduoda testavimo rezultatus testavimo serveriui.
9. Testavimo serveris perduoda rezultatus tinklu modulių testavimo karkasui, esančiam kitame kompiuteryje.

10. Modulių testavimo karkasas atlieka rezultatų palyginimą su laukiamomis reikšmėmis.
11. Kartojamas 1-10 žingsniai kol nesibaigė modulių testai.
12. Formuojama testavimo ataskaita.

Modulių testavimo karkasas, esantis išoriniame kompiuteryje, atlieka 1-2, 10-12 žingsnius. Testavimo serveris esantis mobiliajame įrenginyje atlieka 3-9 žingsnius.

Modulių testai gali būti saugomi kokioje nors duomenų bazėje, nes jie yra pateikiami deklaratyvia forma. Naudojant deklaratyvia forma testus, modulių testai nepriklauso nuo programavimo kalbos kuria jie parašyti. Atliekant programinės įrangos palaikymo darbus, gali tekti atlikti programinė įrangos reinžinerija. Vienas iš galimų variantų gali būti programavimo kalbos pakeitimas. Jei nauja programavimo kalba yra objektiškai orientuota tai galima panaudoti ir deklaratyviai išreikštus testus ir perdarytai programinei įrangai testuoti. Tam atlikti užtektų tik perrašyti testavimo serverį.

Privalumai:

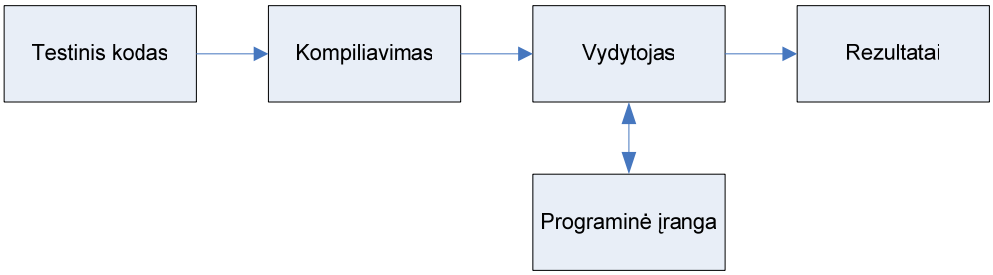
- Reikalingas atminties kiekis įvykdyti tik vienam modulio testui.
- Deklaratyviai pateiktus testus lengviau generuoti.
- Modulių testai gali būti pritaikyti įvairioms programavimo kalboms.
- Testai nepriklausomi vienas nuo kito (kas leidžia mums sukurti lengvą ir paprastą sistemą, kitu atveju galima sukurti vieną didelę monolitišką testavimo sistema).
- Testus galime naudoti regresiniam testavimui, net jie sugalvojome pakeisti platformą, programavimo kalbą.

Trukumai:

- Tinklo gedimas gali nutraukti testų vykdymą.
- Technologija, kuria parašyta programinė įranga, turi palaikyti atspindžius (*reflections*) [3] ir tipų identifikavimą vykdymo metu (RTTI) [20].
- Jei testai pateikti standartiniu būdu, kaip daromi modulių testai t.y. kodu, reiktu atlikti kodo analizavimą ir sugeneruoti testo duomenis(deklaratyvius aprašus).

2.10.4.2. Konkretaus valdiklio būdas

Kitas variantas yra modulių testus išreikšti kaip ir yra naudojama visais atvejais, programiniu kodu. Tokio modulio testo pavyzdys pateiktas 2.5 (a) paveikslėlyje. Testo vykdymo metu serverio pusėje išrenkamas modulio testo kodas, sukompiliuojamas pritaikant jį mobiliojo įrenginio platformai, gautas vykdomas kodas persiunčiamas į mobilųjį įrenginį testavimo servisui. Testavimo servisas įvykdo atsiųstą kodą ir gražina jo patiektus rezultatus. Modulio testo vykdymas pateiktas 2.14 paveikslėlyje.



2.14 pav. Konkretaus valdiklio vykdymas

Visas testavimo procesas vyksta taip:

1. Ištenkamas vienas modulio testas.
2. Sukompiliuojamas modulio testas pritaikant jį konkrečiam mobiliam įrenginiui.
3. Sukompiliuotas kodas perduodamas testavimo servisui mobiliajame įrenginyje.
4. Testavimo servisas vykdo atsiųstą modulio testo kodą mobiliajame įrenginyje.
5. Testavimo servisas surenka modulio testo vykdymo rezultatus
6. Testavimo serveris perduoda rezultatus tinklu modulių testavimo karkasui, esančiam kitame kompiuteryje.
7. Modulių testavimo karkasas atlieka rezultatų palyginimą su laukiamomis reikšmėmis.
8. Kartojamas 1-8 žingsniai kol nesibaigė modulių testai.
9. Formuojama testavimo ataskaita.

Modulių testavimo karkasas, esantis išoriniame kompiuteryje, atlieka 1-3, 7-9 žingsnius. Testavimo serveris esantis mobiliajame įrenginyje atlieka 4-6 žingsnius.

Modulių testai saugomi kaip programiniai kodai, norint pagreitinti modulių testų vykdymą , galima kompiliavimą atlikti tik vieną karta, o vykdant tą patį testą, tame pačiame mobiliajame įrenginyje pasinaudoti ankščiau sukompiliuotu modulio testo kodu. Šiuo atveju dar tektų stebėti ar nebuvo modifikuotas modulio testo kodas ir ar dėl to neriekia atlikti modulio testo perkompiliavimo.

Privalumai:

- Reikalingas atminties kiekis įvykdyti tik vienam modulio testui.
- Testai nepriklausomi vienas nuo kito (kas leidžia mums sukurti lengvą ir paprastą sistemą, kitu atveju galima sukurti vieną didelę monolitišką testavimo sistema)
- Testus galime naudoti regresiniam testavimui.
- Nereikalingi atspindžiai.

Trukumai:

- Tinklo gedimas gali nutraukti testų vykdymą.
- Modulio testai turi būti ta pačia programavimo kalba kaip ir testuojama programa.
- Reikalingas modulio testo kompiliavimas.

2.10.4.3. Darbas aplinkoje be atspindžių

Jei technologija, kuria realizuota testuojama programa, nepalaiko atspindžių (*reflections*) [3] ir tipų identifikavimo vykdymo metu (RTTI) [20], sudėtinga atlikti modulių testų atskyrimą nuo testuojamos programinės įrangos ne tik atskiriant į skirtingus kompiuterius, bet problemų sukuria ir bandymas atskirti modulių testus nuo programinės įrangos išskaidant į bibliotekas. Vienas iš būdų yra įkompiliuoti testuojamos programinės įrangos elementus, kurie yra testuojami konkrečiame modulio teste, į patį modulio testą [24]. Žinoma toks sprendimas žymiai padidins modulio testo dydį.

3. Projektinė dalis

3.1. Sistemos paskirtis

Projekto tikslas yra sukurti mobilaus dėstytojo programinę įrangą. Dėstytojas galės visus darbus atlikti naudodamasis delniniu kompiuteriu. Naudodamasis delniniu kompiuteriu ir programine įranga dėstytojas galės sutaupyti laiko, kuris būtų išleidžiamas darbui su popieriais (pažymių suvedinėjimas į kompiuterį nuo popieriaus lapų, atsiskaitymo rezultatų žymėjimas, studentų lankomumo suvedinėjimas į kompiuterį). Dėstytojas naudodamasis delniniu kompiuteriu galės naudotis programinės įrangos teikiamomis funkcijomis:

- Lankomumo žymėjimas, ataskaitos.
- Atsiskaitymų rezultatų žymėjimas, jų ataskaitos.
- Rezultatų publikavimas internete.
- Darbų skyrimas studentams, jų publikavimas internete.
- Studentų darbų priėmimas per internetą.

Studentai galėtų:

- Peržiūrėti jų darbų vertinimus internete.
- Pateikti atliktas individualias užduotis dėstytojui per internetą.

Naudodamasis delninių kompiuteriu dėstytojas galės atlikti rutininius darbus daug greičiau ir galės daugiau laiko skirti svarbesniems dalykams.
Sprendimas yra skirtas universitetams, kurie nori automatizuoti ir palengvinti dėstytojų darbą naudojant mobilias technologijas.

3.2. Panašūs projektai

Yra keletas komercinių realizacijų, kurios įgyvendina mobilaus dėstytojo idėją ir taiko mobilias technologijas joms įgyvendinti.

3.2.1. Kowloon Wan Yah koledžo sprendimas

Sprendė problemą susijusią su studentų lankomumo registravimu. Problemos esmė buvo, kad neužtekdavo suregistruoti studentus dienos pradžioje, nes studentai turėjo galimybę rinktis kokius dalykus mokintis ir tekdavo suregistruoti studentus kiekvieno užsiėmimo metų, taip pat tekdavo nustatyti, kokie gi studentai turi dalyvauti užsiėmime.

Kad palengvinti dėstytojų darbą ir išspręsti šią problemą, buvo nuspręsta naudoti mobilius įtaisus:
Visame koledže buvo įdiegtas bevielis tinklas, dėstytojai studentų lankomumo registravimui naudojo delninius kompiuterius. Buvo naudojama kompanijos North22 TAAS (Teacher Administration Assistant System) programinė įranga, kuri teikė galimybes naudojant delninius kompiuterius:

- Registruoti studentų lankomumą naudojant delninius kompiuterius
- Registruoti mokinių rezultatus
- Skirti studentams užduotis
- Studentams galimybės peržiūrėti užduotis.
- Dėstytojams galimybė peržiūrėti studentų atliktas užduotis
- Peržiūrėti lankomumo ataskaitas
- Peržiūrėti studentų rezultatų suvestines.

Projektą realizuojant buvo pasirinkta naudoti PocketPC delninius kompiuterius, nes

- Juose yra visiem pažįstama Windows aplinka
- Galimybė lengvai keistis Microsoft Word, Microsoft Excel, Microsoft Outlook dokumentais su staliniais kompiuteriais.

Projekto rezultatai:

- Rankinis studento pamokų lankymo rezultatų suvedimas eliminuotas sutaupant vieną žmogaus darbo valandą kiekvieną dieną.
- Detalios paskaitų lankymo ataskaitos gali būti lengvai sugeneruotos, kad išanalizuoti tendencijas ir efektyviau paskirstyti resursus.

3.2.2. „Mobile teacher“ programinis sprendimas

Programinės įrangos gamintojai teigia, kad ši programinė įranga pakeis klasės valdymą. Programinės ir techninės įrangos derinys atlaisvina dėstytojus nuo laiką gaišinančių administracinių darbų, bereikalingų bėgiojimų į ofisą ir atgal. Vien dėl to, kad visa reikalinga informacija yra lengvai pasiekiama naudojat delninį kompiuterį. Naudojant „Mobile Teacher“ programinę įrangą dėstytojai galės dėmesį skirti paskaitoms, jų vedimui, o ne darbui su popieriais, tuo padėdami studentams geriau įsisavinti mokomąją medžiagą.

Dėstytojai gali turėti visą informacija apie paskaitas, lankomumo ataskaitas, pažymius ir t.t. būdami universitete ir nebūtinai ten. Pildyti lankomumo ataskaitas lengvai ir greitai.

Įvedinėti studentų pažymius esant bet kurioje vietoje (pvz.: namie, valgykloje, paskaitos metu), visada naudojantis tuo pačiu įrenginiu.

Taip pat gauti, naudojantis delniniu kompiuteriu, informaciją apie konkretaus studento mokymosi rodiklius, lankomumo statistiką.

Programinė įranga veikia įvairiuose įrenginiuose (Symbol SPTxxxx, Palm III, Palm IV, Compaq IPAQ, HP Jordana, Cassio Cassiopedia). Vartotojo sąsaja yra realizuota žinatiklio naršyklėje. Veikia naudojant Internet Explorer ir Netscape Navigator naršykles.

Programinė įranga teikia tokias funkcijas:

- Studentų sąrašai.
- Darbuotojų sąrašai.
- Lankomumo registravimas.
- Pažymių įvedimas.
- Darbų atlikimo būsenos įvedimas (kiek studentai atliko užduoties ir t.t.).
- Studentams skirtų užduočių įvedimas.
- Studentų informacijos peržiūra.
- Galimybė visą informaciją pasiekti ir per internetą naudojantis staliniu kompiuteriu.
- Tvarkaraščių sudarymas.

3.2.3. eScholar

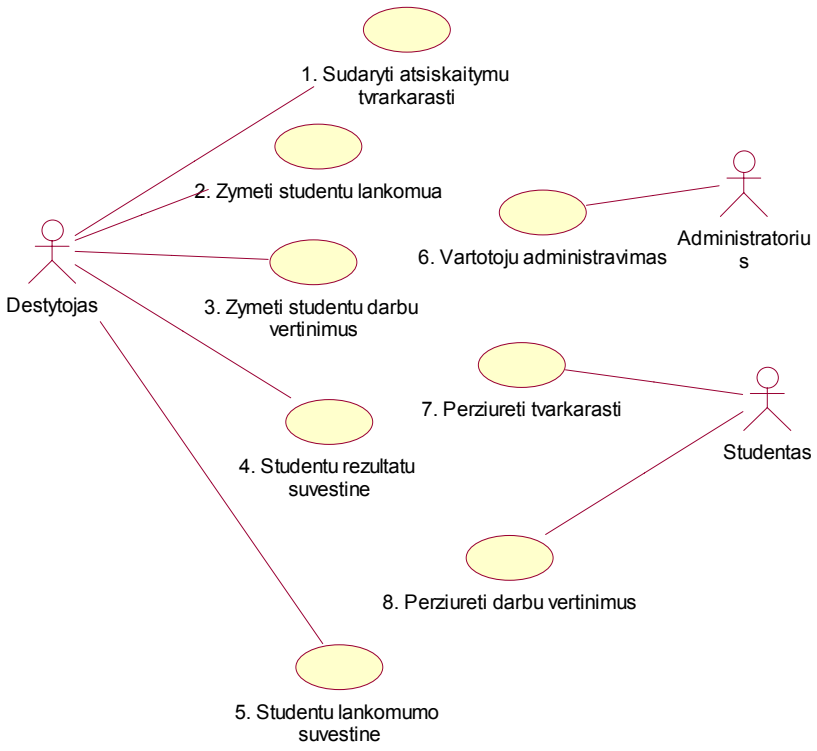
Realizavo mobilaus dėstytojo idėją naudojant Palm delninius kompiuterius ir DB2 Everyplace duomenų bazę. Universitetuose jau būna kažkokia anksčiau egzistuojanti informacinė sistema ir šis IBM ir eScholar sprendimas buvo paremtas tuo, kad dėstytojui nereikia visos informacijos iš karto, tad į mobilųjį įrenginį buvo įdiegta mažytė duomenų bazė, kuri tam tikrais laiko momentais sinchronizuojasi su pagrindinėmis duomenų bazėmis ir atsisiunčia rūpimus duomenų bazės fragmentus ir tuo pačiu, nusiunčia į duomenų bazę per tam tikrą laiko tarpą surinktus duomenis. Toks sprendimas įgalina naudoti paprastesnius ir pigesnius įrenginius, nes įrenginiams nebūtinos bevielio tinklo galimybės, taip pat nereikia ir bevielio tinklo infrastruktūros.

Programinė įranga teikia tokias galimybes:

- Lankomumo registravimas
- Pažymių registravimas
- Studento informacijos peržiūrėjimas
- Studento pažangumo stebėjimas.

3.3. *Esminiai reikalavimai*

Sistemai keliami funkciniai reikalavimai gali būti išreikšti Volere šablono lentelėmis, kuriose nurodomi reikalavimo pavadinimas, svarbumas, ryšiai su kitais ir panašiai. Kitos projektavimo metodologijos siūlo reikalavimus pateikti kaip panaudojimo atvejų diagramas, kurios atspindi ne sistemos realizacijos funkcijas, o vartotojų atliekamas funkcijas. Panaudojimo atvejų vaizdas vienas pats neteikia pilnos informacijos, bet teikia tik paprastą ir vaizdžią reikalavimų suvestinę. Kiekvieną panaudojimo atvejį detalizavus scenarijumi, galime turėti pakankamai išsamų reikalavimų dokumentą. Tokį būdą reikalavimams dokumentuoti siūlo Microsoft Solution Framework ir Unified Process metodologijos. Kuriamos sistemos funkciniai reikalavimai pateikti 3.1 paveikslėlyje, panaudojimo atvejų diagramoje.



3.1 pav. Programinės įrangos panaudojimo atvejai

Tačiau, kadangi ši sistema skirta realizuoti mobilumo idėjoms, didžiausią svorį sudaro nefunkciniai reikalavimai, kurie yra įtakoti mobilių technologijų panaudojimo:

- Darbas įrenginiuose su nepatogia vartotojo sąsaja

  Delniniame kompiuteryje ekranas yra tik 240x320 taškų dydžio. Duomenų įvedimui naudojamas jautrus ekranas, klaviatūra su mažu klavišų kiekiu rodoma ekrane. Darbas triukšmingoje aplinkoje. Darbas viena ranka.
- Saugumas

Dėl to, kad duomenys perduodami bevieliu tinklu, reikia užtikrinti duomenų saugumą ir integralumą.

Praradus delninį kompiuterį reikia užtikrinti, kad treti asmenys nepasinaudotų jame esančiais duomenimis.

- Energijos taupymas

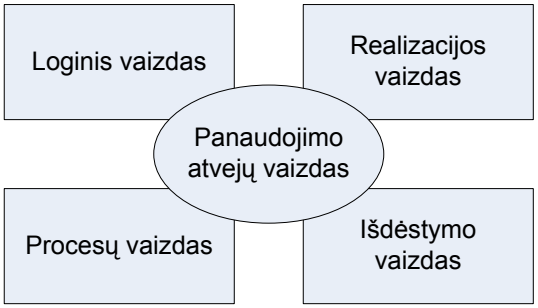
  Dirbant su programine įranga, esančia delniniame kompiuteryje, turėtų būti taupoma energija, kad energijos šaltinis, esantis delniniame kompiuteryje, neišsikrautų ankščiau negu baigiasi darbo diena.

- Pritaikymas įvairaus tipo mobiliesiems įrenginiams

  Delniniai kompiuteriai būna įvairių architektūrų, reikia pritaikyti programinę įrangą, kad ji veiktų skirtingas architektūras turinčiuose delniniuose kompiuteriuose.

3.4. Architektūra

Sistemos architektūrą pateikiama keliais vaizdais: panaudojimo atvejų vaizdu, procesų vaizdu, išdėstymo vaizdu ir realizavimo vaizdu. Šie vaizdai yra pateikiami kaip Rational Rose Modeliai ir juose naudojama unifikuota modeliavimo kalba (UML). Sistemos architektūra pateikta remiantis RUP (Rational Unified Process) rekomendacijomis ir Rational Architecture Practice gairėmis [26]. Sistemos specifikacija pateikta šiais vaizdais, kuriems įgyvendinti reikia UML diagramų. Vaizdai pateikti 3.2 paveikslėlyje.



3.2 pav. Architektūros pateikimas

Programinės įrangos architektūra pateikiama kaip aibė UML diagramų:

- Panaudojimo atvejų vaizdas

  Panaudojimo atvejų diagrama.

- Loginis vaizdas

  Klasių diagramos.

Sistemos išskaidymas į paketus.

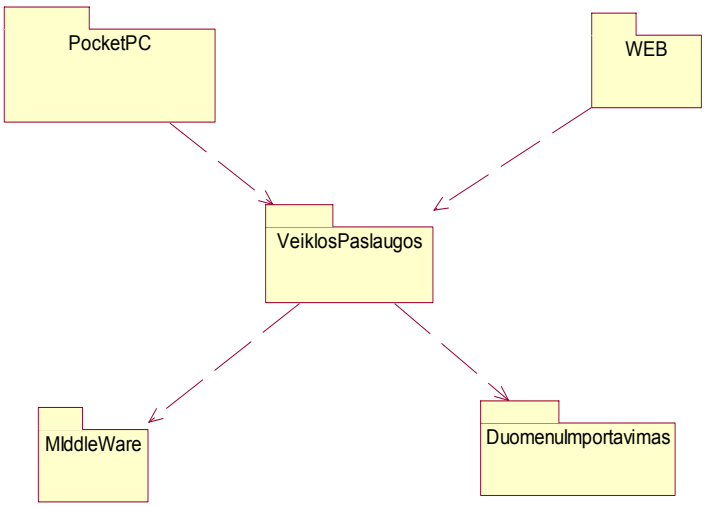
- Procesų vaidas
  Būsenų diagramos.
  Sekų diagramos.
  Bendradarbiavimo diagramos.
- Išdėstymo vaizdas
  Išdėstymo diagrama.
- Realizacijos vaizdas
  Komponentų diagrama.

Yra keletas reikalavimų ir apribojimų, kurie turi įtakos sistemos architektūrai. Tai yra:

- Sistema turi veikti delniniame kompiuteryje, kuris neturi pastovaus ryšio su tinklu.
- Sistema turi turėti galimybę pasiimti duomenis iš KTU A IS, taip pat turi būti atspari tos sistemos sąsajos pasikeitimams.
- Sistema turi pasiimti duomenis iš HTML dokumentų ir turėti galimybę išplėsti duomenų pasiėmimą iš visokiausių kitokių formatų dokumentų.
- Sistema neturi leisti neautorizuotiems vartotojams prie jos prisijungti.
- Sistema bus realizuota kliento-serverio modelyje. Serverinė dalis bus web serveryje, o klientinė - bet kokia tinklo naršyklė (pvz.: Internet Explorer, Mozilla).
- Programinė įranga turi naudoti Pocket PC delniniame kompiuteryje esančią kalendoriaus ir susitikimų valdymo sistemą, perduoti jai informaciją apie įvykstančius užsiėmimus. Su delniniame kompiuteryje esančia kalendoriaus valdymo programine įranga bendraujama per Pocket Outlook sąsają.
- Programinė įranga turi naudoti Pocket PC delniniame kompiuteryje esančią užduočių valdymo sistemą, perduoti jai informaciją apie reikiamus atlikti vartotojo darbus. Su delniniame kompiuteryje esančia užduočių valdymo programine įranga bendraujama per Pocket Outlook sąsają.
- Dalis sistemos veiks įvairaus tipo delniniuose kompiuteriuose (skirtingi procesoriai).
- Kuriama programinė įranga bus pateikta kaip atviro kodo programinė įranga visiems norintiems ja naudotis.
- Dėl įvairių delninių kompiuterių, programinė įranga jiems bus kuriama naudojant virtualią mašiną (konkrečiai Microsft .NET).
- Dėl delninio kompiuterio neturėjimo pastovaus ryšio su tinklu, jame bus naudojama Microsoft SQL CE Server duomenų bazė, kuri turės pagrindinės duomenų bazės

kopiją ir įrenginiui tam tikrais laiko tarpais pasiekus tinklą, duomenis sinchronizuos su pagrindine duomenų baze.

Programinė įranga yra išskaidoma į paketus, kurie po to detalizuojami klasių diagramomis. Sistemos išskaidymas į paketus pateiktas 3.3 paveikslėlyje.



3.3 pav. Programinės įrangos išskaidymas į paketus

Paketas PocketPC atitinka sistemos dalį, veikiančią delniniame kompiuteryje. Jame yra Klasės skirtis darbui su lokalia duomenų baze, įvedimo formos, ataskaitų gavimo langų realizavimas. Taip pat yra priemonės sinchronizuoti dėstytojo kalendorių su delniniame kompiuteryje esančiais komponentais kaip Pocket Calendar ir Pocket Tasks. Taip pat pateikiamos klasės duomenų sinchronizavimui su pagrindine duomenų baze, kai delninis kompiuteris prisijungia prie tinklo.

Pakete WEB pateikiamos klasės skirtos bendravimui su vartotoju. Jame yra tik sistemos vaizdavimo lygio klasės (duomenų išvedimo langai ir įvedimo langai). Šiame pakete esančios klasės visoms funkcijoms atlikti naudoja klases, esančias VeiklosPaslaugos pakete. Pakete pateikiamos klasės, realizuojančios studento, dėstytojo ir administratoriaus vartotojo sąsajas, jos yra detaliau suskirstytos į žemesnio lygio paketus.

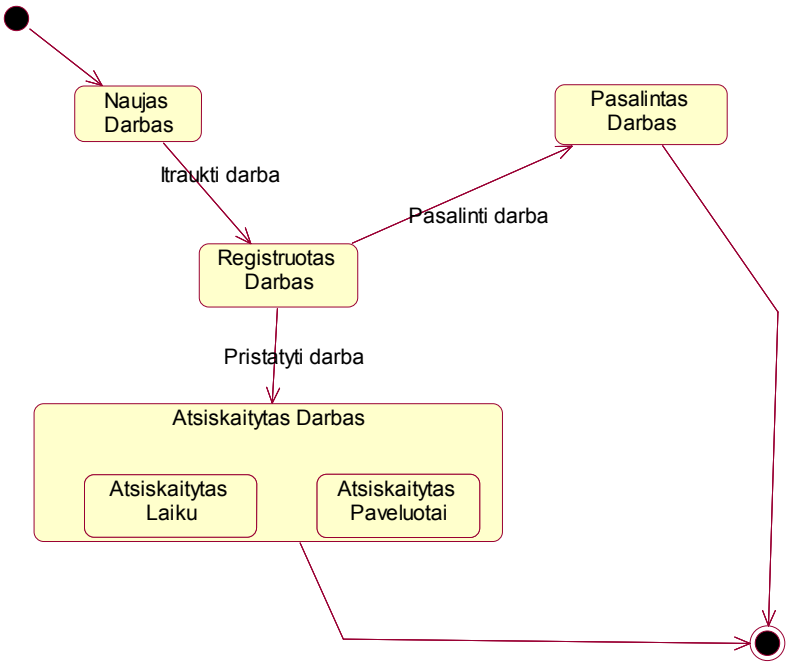
Pakete VeiklosPaslaugos pateikiamos klasės realizuojančios visą sistemos funkcionalumą. Pakete nėra klasių realizuojančių vartotojo sąsają. Jame yra tik klasės realizuojančios sistemą. Jas naudoja kitų paketų klasės atlikti duomenų vaizdavimą. Naudojant šitame pakete esančias klases galima realizuoti skirtingus vaizdavimo lygius. Šioje realizacijoje naudojant šiame pakete esančias klases, sukurti vaizdavimo lygiai, esantys

PocketPC pakete ir WEB pakete (tas pats funkcionalumas, tik skiriasi vaizdavimas, vienas yra delniniame kompiuteryje, o kitas - WEB naršyklėje).

Pakete MiddleWare pateikiamos klasės skirtos darbui su SQL Server duomenų baze. Šio paketo klasės naudoja klases, esančias VeiklosPaslaugos pakete. Paketo klasės yra suskirstytos į žemesnio lygio paketus.

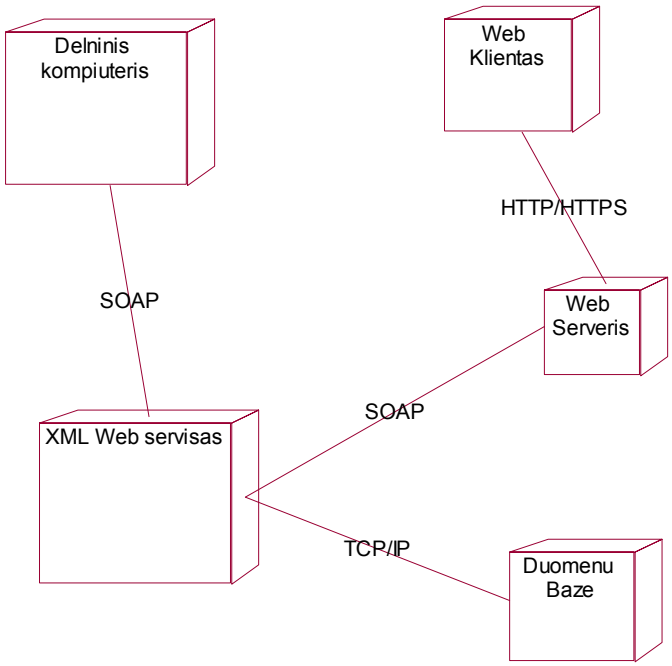
Pakete DuomenuImportavimas pateikiamos klasės skirtos duomenų importavimui iš įvairių formatų failų ir taip pat iš gretimų sistemų. Pakete apibrėžiama duomenų importavimo sąsaja, tuo leidžiant, esant poreikiui prijungti naujus duomenų importavimo formatus.

Programinė įranga taip pat aprašyta kaip aibė programinės įrangos objektų būsenų diagramų. Vienas tokios diagramos pavyzdys pateiktas 3.4 paveikslėlyje



3.4 pav. Darbo būsenos diagrama

Programinė įranga yra diegiama įvairiuose kompiuteriuose. Paveikslėlyje 3.5 pateikta programinės įrangos išdėstymo diagrama.



3.5 pav. Sistemos išdėstymo vaizdas

Duomenų bazė yra diegiama Microsoft Windows 2003 Server kompiuteryje. Duomenų bazės valdymo sistemai naudojama Microsoft SQL Server Desktop Engine. Serveriui reikia tinklo palaikymo. Microsoft SQL Server 2000 Desktop Engine teikia paslaugas naudojant TCP/IP protokolą.

XML WEB servisas yra diegiamas Microsoft Windows 2003 Server kompiuteryje. Jam įdiegti naudojama Microsoft Infomation Services 6.0 web serverio versija.
Serveriui reikia tinklo palaikymo. XML Web servisas teikia paslaugas naudojant SOAP protokolą, kuris yra perduodamas naudojant HTTP protokolą, o HTTP naudoja TCP/IP protokolą perduoti duomenis žemame lygyje. XML WEB servisas realizuojamas .NET platformoje ir naudoja Microsft .NET Framework virtualią mašiną. Kompiuteryje turi būti įdiegta Microsoft .NET Framework 1.1. Jame diegiami komponentai iš paketo VeiklosPaslaugos.

WEB servisas diegiamas Microsoft Windows 2003 Server kompiuteryje. Jam įdiegti naudojama Microsoft Infomation Services 6.0 žiniatinklio serverio versija. Serveriui reikia tinklo palaikymo. Web servisas teikia paslaugas naudojant HTTP protokolą, o HTTP naudoja TCP/IP protokolą perduoti duomenis žemame lygyje. WEB servisas realizuojamas .NET platformoje ir naudoja Microsft .NET Framework virtualią mašiną. Kompiuteryje turi būti įdiegta Microsoft .NET Framework 1.1. Jame diegiami komponentai iš paketo WEB.

Delninis kompiuteris. Vartotojo klientinė programinė įranga veikia PocketPC tipo delniniame kompiuteryje.

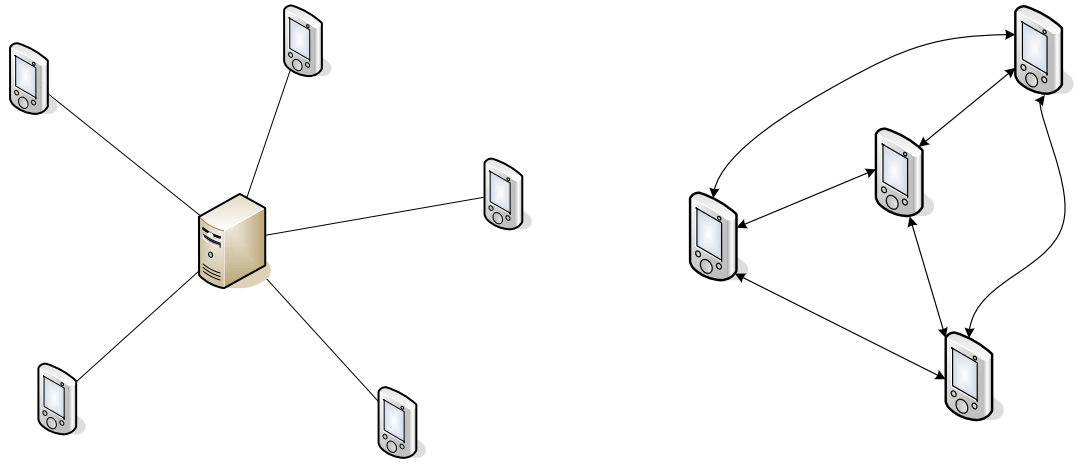
Delninis kompiuteris pasiekia XML web servisą naudojantis SOAP protokolu. Tam reikalingas HTTP ir TCP/IP protokolų palaikymas. Klientinė programinė įranga realizuojama naudojant Microsoft .NET Framework platformą. Kompiuteryje turi būti įdiegta Microsoft .NET Compact Framework 2.0. Jame diegiami komponentai iš paketo PocketPC.

Web klientas naudosis sistemos funkcijomis per tinklo naršyklę. Kompiuteris gali būti su Linux, Windows operacinėmis sistemomis. Sistemos paslaugoms pasiekti galės naudoti Microsoft Internet Explorer 5.0 ir naujesnes naršykles, taip pat bus galima naudotis Netscape Navigator 7.0 arba naujesne naršykle.

3.5. Mobilių sistemų architektūros

Programinė įranga, skirta mobiliesiems įrenginiams, neegzistuoja izoliuota, ji nuolat bendrauja su aplinkos paslaugomis [33, 44] bei kitomis sistemos dalimis. Tai įtakoja ir programinės įrangos architektūros pasirinkimą. Vyrauja dažniausiai Client-Server [6] tipo architektūros, bet vystantis visur esantiems skaičiavimams [5] bei nuo konteksto priklausančioms paslaugoms [33], programinės įrangos kūrėjai renkasi kitokius architektūros būdus: Peer 2 Peer, C2 [31, 43] ir pan.

Paveikslėlyje 3.6 (a) pavaizduota Client-Server tipo architektūra.



(a) (b)

3.6 pav. Mobilių sistemų architektūros

Pasirinkus Client-Server tipo architektūrą [31], mobilieji įrenginiai bendrauja su pagrindiniu serveriu. Pagrindiniame serveryje saugomi visi duomenys. Praradus mobilųjį įrenginį, nedingsta visa informacija, ją galima sėkmingai atstatyti iš pagrindinio serverio. Mobilieji įrenginiai pastoviai naudoja duomenis iš pagrindinio serverio arba tam tikrais laiko

tarpais sinchronizuoja duomenis, esančius mobiliajame įrenginyje su duomenimis, esančiais pagrindiniame serveryje. Sugedus pagrindiniam serveriui gali sutrikti visos sistemos darbas.

Pasirinkus Peer 2 Peer [31] architektūrą (3.6 (b) paveikslėlis), mobilieji įrenginiai bendrauja tarpusavyje, nėra pagrindinio serverio. Esant poreikiui, duomenys yra perduodami iš vieno įrenginio į kitą. Praradus mobilųjį įrenginį gali būti prarasti ir visi to vartotojo duomenys, kurių atstatyti gali būti neįmanoma.

C2 [31, 43] architektūra yra gana panaši, tik vienu metu vienas mobilus įrenginys gali bendrauti su kelias kitais įrenginiais.

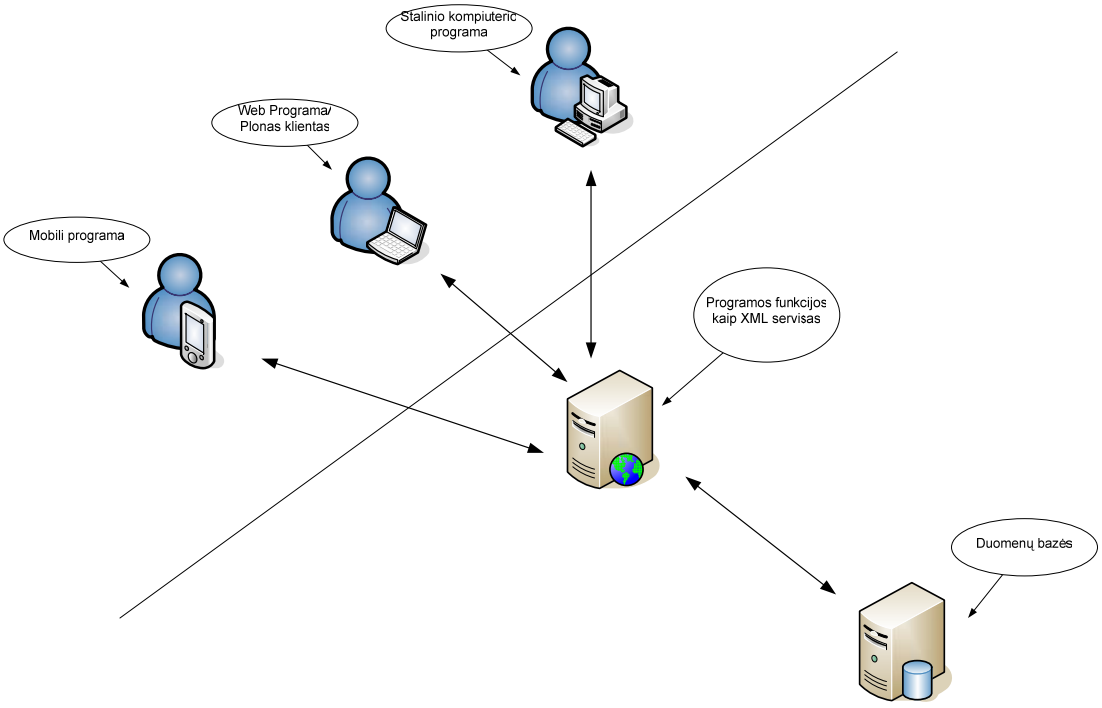
Lentelėje 3.1 pateiktas mobilių sistemų architektūrų palyginimas.

3.1 lentelė. Mobilių sistemų architektūrų palyginimas

Kriterijus	Client-Server	Peer-to-peer
Duomenų saugojimas	Serveryje	Kiekviename įrenginyje
Sistemos gedimas	Sugedus serverui gali dingti visi duomenys.	Sugedus mobiliam įrenginiu gali dingti tik dalis informacijos.
Sinchronizavimas	Klientas-Serveris	Klientas-Klientas
Mobilaus įrenginio pametimas	Neprarandami duomenys.	Galima prarasti konkretaus vartotojo duomenis.
Energijos suvartojimas (prisijungimas prie tinklo)	Pastovus arba tik sinchronizavimui prisijungimo prie tinklo.	Tik duomenų pasikeitimui tarp įrenginių.

3.6. Realizacija

Kuriamai sistemai buvo pasirinkta kliento-serverio architektūra. Jos detalizacija pateikta 3.7 paveikslėlyje.



3.7 pav. Pasirinkta sistemos architektūra

Programinė įranga realizuota kaip trijų lygių sistema:

1 Lygis - Duomenų bazės lygis

Saugoma visa informacija, kontaktai, tvarkaraščiai, studentų sąrašai, pažymiai ir t.t.

2 Lygis – visas funkcionalumas realizuotas serveryje, teikiamos klientui programinės funkcijos (XML web paslauga)

Realizuojamos programinės įrangos funkcijos tokios kaip informacijos išgavimas, duomenų įvedimas, ataskaitų generavimas.

3 Lygis – vaizdavimo lygis, naudojant antro lygio funkcijas, galima lengvai realizuoti ploną klientą (web sąsaja), stalinio kompiuterio programinė įranga, delninio kompiuterio programinė įranga.

Duomenų vaizdavimas, vartojo sąsaja naudoja funkcijas, kurias teikia 2-tras lygis. Galima lengvai realizuoti:

- Web programas.
- Stalinio kompiuterio programas.
- Delninio kompiuterio programas.

Visas funkcionalumas yra antrame lygyje ir naujo vaizdavimo būdo įtraukimas yra pakankamai lengvas.

Duomenų bazės realizuojamos Microsoft SQL Server programinėje įrangoje, nes:

- Patogus ir lengvas duomenų bazės administravimas
- Lengvai plečiamos duomenų bazės galimybės aptarnauti didesnį vartotojų kiekį (pastatant daugiau serverių ir naudojant duomenų bazės turinio replikavimą tarp jų)
- Galimybė sinchronizuoti SQL CE duomenų bazę esančia delniniame kompiuteryje su serveriu.
- Nuolatinis gamintojo palaikymas.
- Geras kainos, našumo, kokybės santykis.

Biznio logikos lygis realizuojamas kaip XML web paslauga, naudojant Microsoft ASP.NET technologiją, nes:

- XML web paslauga įgalina kurti įvairius klientus vaizdavimo lygiui. Juos galima kurti naudojat kitokias programavimo kalbas, technologijas (Java, VB, php ir panašiai).
- Greita darbo sparta. Šitas lygis turi atlikti visus kreipinius į duomenų bazes ir formuoti rezultatus vartotojui. Šiam procesui paspartinti ASP.NET leidžia naudoti užklausų spartinimą (serveris po pirmos užklausos išsaugo rezultatus atmintyje ir sekantį kartą kreipiantis į jį, rezultatai grąžinami iš atminties, tuo sutaupant palyginus lėtų kreipinių į duomenų bazes kiekį per laiko tarpą)
- Galimybė realizuoti šį lygį įvairiomis programavimo kalbomis (C#, Java, VB, C++)
- Lengvai plečiamos galimybės (veikimo sparta, apdorotų užklausų kiekis) vartotojo poreikiams augant (WEB fermos)
- Nuolatinis gamintojo palaikymas.
- Geras kainos, našumo, kokybės santykis.

Klientinė programa realizuojama kaip Windows programa, nes:

- Turtinga vartotojo sąsaja.
- Greitas programos veikimas (efektyviai išnaudojami delninio kompiuterio procesoriaus resursai).
- Nebūtinas nuolatinis prisijungimas prie tinklo.
  Duomenys yra sinchronizuojami iš įrenginio su pagrindine duomenų baze tik tam tikrais momentais (pavyzdžiui: prijungus delninį kompiuterį prie tinklo).

Programinės įrangos diegimui naudojama „ClickOnce“ technologija, kuri įgalina lengvą ir patogų programinės įrangos palaikymą:

- Programa patalpinama žiniatinklio serveryje. Atnaujintos jos versijos dedamos ten pat.

- Klientų kompiuteriai patys atsinaujina programinę įrangą iš žiniatinklio serverio.

3.7. Vartotojo sąsaja

Vartotojo sąsaja turi būti kuriama siekiant sumažinti įvedimo klaidų kiekį. Gerai suprojektuota vartotojo sąsaja yra labai svarbi, nes:

- Pagal vartotojo sąsają dažniausiai sprendžiama apie programinę įrangą, o ne jos funkcionalumą
- Prastai suprojektuota vartotojo sąsaja gali įtakoti vartotojų daromų klaidų kiekio padidėjimą
- Programinės įrangos atmetimo/nenaudojimo priežastis dažnai būna prasta vartotojo sąsaja.

Dėl mažų delninių kompiuterių ekranų, vartojo sąsaja gali gautis labai sugrūsta,stengiantis sukišti kuo daugiau ir kuo arčiau šalia vienas kito įvedimo laukų, kas neduotų reikiamo efekto, o tik padidintų vartotojo daromų klaidų kiekį įvedant duomenis. Tai pat leidimas vartotojui įrašyti pačiam duomenis gali būti klaidos šaltinis. Tam, kad išvengti įvedimo klaidų, programinės įrangos vartotojo sąsaja buvo kuriama atsižvelgiant į [37]:

- Kur galima naudoti pasirinkimo laukelius.
  Galimos reikšmės pasirenkamos iš ComboBox. Tai leidžia išvengti negalimų reikšmių įvedimų. Taip pat nereikia naudotis maža ir sąlyginai nepatogia klaviatūra. Duomenis galima įvesti naudojantis pieštuku ir sąraše pasirenkant norimą duomenų variantą. Taip pat padarius pasirinkimo dėžutę pakankamai didelę, galima duomenis įvesti nesinaudojant pieštuku, o tiesiog pasirenkat ir paspaudžiant norimą reikšmę ranka.
  Galimos reikšmės gaunamos iš įvedamų duomenų specifikacijos, kurios būna surinktos reikalavimų išgavimo proceso metu.
- Minimizuoti tekstų įvedimo atvejus
  Rodomos pradinės reikšmės arba statistiškai nuspėjamos galimos naujos reikšmės (pavyzdžiui, pildant naują užsakymą, automatiškai suformuojamas sekantis užsakymo numeris). Tai taip pat jas galima išgauti iš reikalavimų įvedamiems duomenims specifikacijos.
- Išskaidyti įvedamos informacijos laukus dalimis.
  Įvedamai informacijai išskaidyti per keletą langų galima naudoti korteles (*tabs*), jie turi būti rodomo ekrano apačioje. Tuo gauname neperkrautą įvedimo langą. Žinoma reikia įvedamą informaciją sugrupuoti pagal prasmę iš išskirstyti į kortelės puslapius.

Naudojant korteles, dažnai jų būna daug ir jos netelpa ekrane, todėl šiai problemai spęsti galima vietoj užrašų rodyti tik kortelių ikonas, o puslapiui tapus aktyviam - užrašyti ir pavadinimą. Šis būdas labiau tinka, kai nereikalinga kokia nors laukų įvedimo nuosekli tvarka.

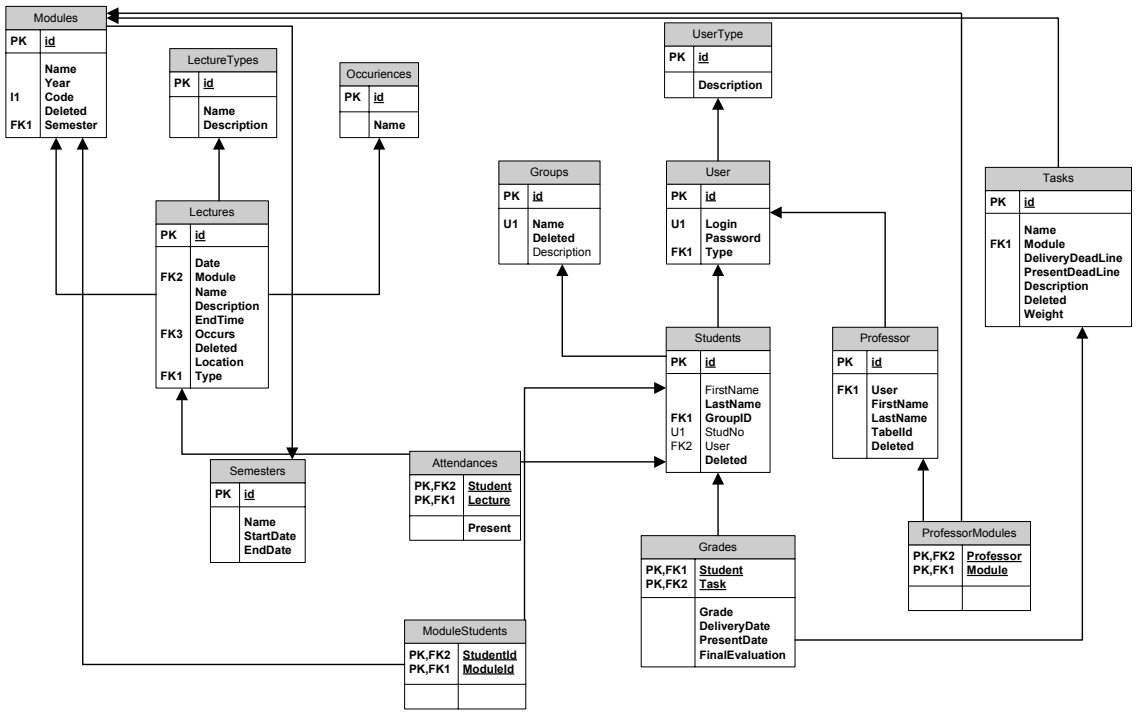
Analogiškai kaip ir kortelės atveju, įvedimo laukus galima sugrupuoti ir pateikti vedlio (*wizzard*) įvedimo dialogo lange. Ekrane rodomi įvedimo laukai ir apačioje mygtukai pirmyn, atgal, nutraukti įvedimą. Toks įvedimo būdas labiau tinka, jei informaciją reikia įvesti kokia nors nuoseklia tvarka.

3.8. Duomenų bazė

Kadangi programinė įranga realizuota atjungtoje aplinkoje, mobiliajame įrenginyje turi būti saugoma duomenų bazės kopija, su kuria būtų dirbama, kai nėra ryšio su pagrindine duomenų baze, esančia serveryje. Duomenų bazės valdymo sistema mobiliajame įrenginyje buvo pasirinkta Microsoft SQL CE Server. Ši sistema teikia patogias priemones duomenų sinchronizavimui. Vienas iš svarbesnių faktorių pasirenkant šią duomenų bazę yra jos kaina, taip pat galimybė sinchronizuoti duomenis su Microsoft SQL server desktop edition, kuri yra pateikiama nemokamai. Buvo nagrinėjamos kitos duomenų bazės, jų palyginimai pateikti priede D ir straipsnyje [52].

Kai kurie autoriai teigia, kad reikia naudoti XML duomenų bazes vietoj reliacinių [25]. Pagrindinis jų argumentas yra, kad reliacinės duomenų bazės yra skirtos sutaupyti diske vietą, o dabar vietos trūkumas nėra aktualus. Tai galioja duomenų bazių valdymo sistemoms, esančioms galinguose serveriuose, bet nelabai pritaikoma duomenų bazių valdymo sistemoms, esančioms mobiliuosiuose įrenginiuose, nes juose vis dar yra nedideli atminties resursai.

Programinės įrangos naudojama duomenų bazės schema pateikta paveikslėlyje 3.8.

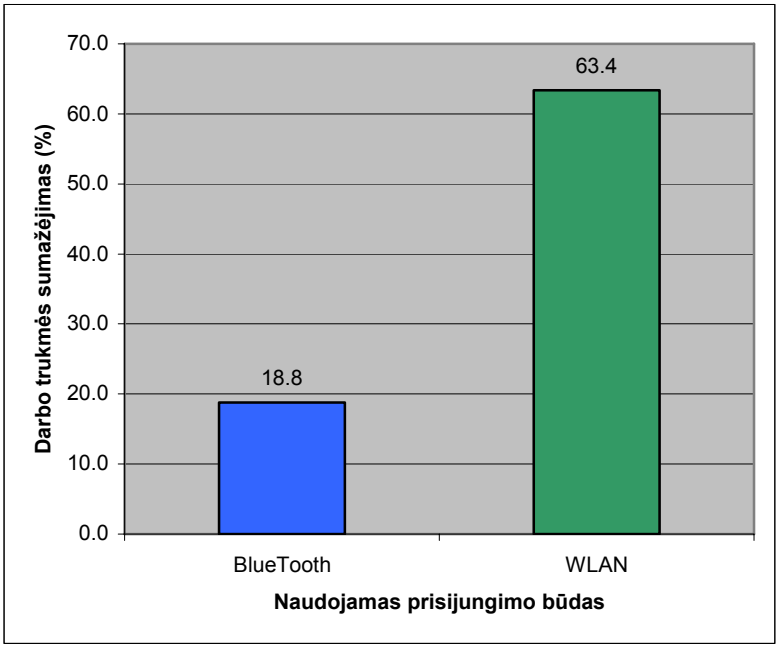


3.8 pav. Programinės įrangos duomenų bazės schema

Duomenys į mobilųjį įrenginį įkeliami sinchronizavimo metu. Sinchronizavimo metu įkeliami tik konkretaus vartotojo duomenys. Mobiliajame įrenginyje saugoma tik dalis duomenų bazės schemos, o tuo pačiu ir tik dalis duomenų, kurie reikalingi tik vienam vartotojui.

3.9. Energijos taupymas

Programinė įranga yra naudojama mobiliuosiuose įrenginiuose. Mobilieji įrenginiai paprastai naudoja akumuliatorius, kurie teikia ribotus energijos kiekius. Tad yra svarbu, kad programinė įranga esanti mobiliajame įrenginyje suvartotų kuo mažiau energijos. Daugiausia energijos suvartojama tinklo priemonių funkcionavimui. Paveikslėlyje 3.9 pateiktas mobilaus įrenginio darbo trukmės sutrumpėjimas priklausomai nuo naudojamo prisijungimo būdo.



3.9 pav. Darbo trukmės sutrumpėjimo priklausomybė nuo prisijungimo būdo

Detalūs duomenys pagal kuriuos buvo gauti rezultatai ir pateikti paveikslėlyje 3.9, aprašyti priede A. Eksperimento metu buvo matuojama kiek laiko programinė įranga gali dirbti. Darbo trukmė buvo išmatuota skaičiuojant per kiek laiko išsikrauna mobilaus įrenginio akumuliatorius dirbant su programine įranga. Testo metu buvo bandomas darbas prisijungus prie tinklo naudojant BlueTooth technologiją (tam reikalingas mobilus telefonas ir GPRS ryšys), WLAN (bevielis vietinis tinklas) ir darbas be tinklo (duomenis sinchronizuojant tik tam tikrais laiko tarpais). Technologijų palyginimas pateiktas straipsnyje [53].

Remiantis sunaudojamu energijos kiekiu, nuspręsta suprojektuoti sistemą, dirbančią atjungtoje nuo tinklo aplinkoje ir duomenis sinchronizuojančią tik tam tikrais laiko tarpais. Toks būdas padėjo sutaupyti maksimalų energijos kiekį.

3.10. Sistemos testavimas

Norint sėkmingai atlikti programinės įrangos testavimą mobiliajame įrenginyje, buvo pritaikyta modulių testavimo karkaso idėja. Pasiūlėme nuotolinį modulių testavimą, kuris modifikuoja tradicinį modulių testavimą, atskirdamas testavimo karkasą nuo programinės įrangos ne tik į atskiras bibliotekas, bet ir išskirdamas į atskirus kompiuterius. Tai leidžia vykdyti didelį modulių testų kiekį, kuris nepriklauso nuo mobilaus įrenginio ribotų resursų ir patogiai stebėti testavimo rezultatus, kurie vaizduojami ne mobiliajame įrenginyje, o kitame kompiuteryje.

Nuotolinis modulių testavimas aptartas analitinėje dalyje, jo realizacija pateikta tyrimo dalyje ir jo eksperimentinis pagrindimas, ir efektyvumo įvertinimas pateikti eksperimentinėje dalyje.

Sistemos testavimui taip pat buvo vykdomi modulių integracijos ir priėmimo testavimai.

3.11. Vartotojo dokumentacija

Vartotojo dokumentacija pateikiama kaip on-line, nuo konteksto priklausanti pagalbos sistema. Toks dokumentacijos realizavimas padeda vartotojui greitai ir patogiai rasti reikiamą informaciją, sutaupo vartotojo laiką, kuris būtų išeikvotas reikiamos informacijos paieškai dokumentacijoje.

3.12. Diegimas ir palaikymas

Kuriant programinę įrangą delniniams kompiuteriams, iškyla diegimo ir tolesnio palaikymo problemos:

- Delninių kompiuterių yra daug ir skirtingų (skirtingos operacinės sistemos, tokios kaip Microsoft Pocket PC, Palm OS, skirtingos procesorių architektūros ARM, MIPS, SH3).
- Delniniai kompiuteriai būna pas vartotojus, o ne ofisuose ir IT personalas negali lengvai ir paprastai atnaujinti programinės įrangos.

Dėl delninių kompiuterių įvairovės, tenka kurti daug tos pačios programinės įrangos versijų, pritaikytų skirtingoms operacinės sistemoms ir delninių kompiuterių procesoriams. O diegiant programinę įranga, reikia pasitelkti organizacines priemones ir suorganizuoti susitikimus su kiekvienu vartotoju, kad įdiegti programinę įrangą į jo delninį kompiuterį. Vienkartinis diegimas nėra blogai, bet situacija pablogėja, kai reikia atnaujinti programinę įrangą (pataisius klaidas, įtraukus naujų funkcijų į programinę įrangą): tektų vėl organizuoti susitikimus su kiekvienu vartotoju kaip ir diegimo metu.

Diegimo ir palaikymo problemas išspręsti galima pasirenkant programinės įrangos architektūrą ir realizuojant norimą programinę įrangą delniniams kompiuteriams vienu iš galimų būdų [36]:

- Programinė įranga vykdoma žiniatinklio naršyklės lange (WEB programa).
- Save atsinaujinanti ir automatiškai įsidiegianti programinė įranga.

Programinė įranga mobiliajame įrenginyje buvo kurta kaip save atsinaujinanti ir automatiškai įsidiegianti programinė įranga. Ji būtų diegiama ir vykdoma vartotojo delniniame kompiuteryje [28]. Įvairių operacinių sistemų ir procesorių architektūrų palaikymas būtų užtikrintas naudojant virtualią mašiną (pavyzdžiui, Sun Java VM arba Microsoft .NET). Programinės įrangos atnaujinimas būtų atliekamas automatiškai: paleidus programą arba atsiradus ryšiui su tinklu (vartotojas prijungia delninį kompiuterį prie stalinio kompiuterio).

Programos įdiegimas vyktų panašiu būdu kaip ir programų, vykdomų žiniatinklio naršyklės lange:

- Programos vykdomi ir kiti reikalingi failai būtų patalpinami reikiamame žiniatinklio serveryje.
- Sugeneruojamas pradinis puslapis su nuoroda į instaliacijos failą.
- Vartotojui perduodamas adresas.
- Vartotojas nuėjęs nurodytų adresu, įvykdo padėtą diegimo bylą.
- Instaliacinis failas įdiegia virtualią mašiną, nukopijuojami programinės įrangos bylos, įtraukiama nuoroda į delninio kompiuterio paleidimo meniu.
- Paleidžiama programinė įranga.

Programinės įrangos atnaujinimas vyktų paleidžiant programą ir esant ryšiui su tinklu arba, kai delninis kompiuteris susijungia su tinklu kokiu nors laiko momentu (prijungus delninį kompiuterį prie stalinio kompiuterio sinchronizuoti kontaktus ir panašią informaciją, būtų sinchronizuojama ir programinė įranga). Programos atnaujinimo žingsniai būtų tokie:

- Pataisius klaidas ar įdėjus naujų funkcijų, programinė įranga patalpinama tuo pačiu žinatinklio adresu.
- Vartotojas eilinį kartą bando pasinaudoti programine įranga delniniame kompiuteryje.
- Programinė įranga patikrina savo versiją ir serveryje esančią versiją.
- Radus naujesnę programinės įrangos versiją, įdiegiama nauja programinės įrangos versija iš žinatinklio serverio.
- Paleidžiama programinė įranga tolesniam darbui.

4. Tyrimo dalis

Egzistuojantiems problemos sprendimo būdams palyginti su pasiūlytu nuotoliniu modulių testavimu, atliekamas palyginimas pagal tam tikrus kriterijus.

4.1. Palyginimo metrikos

IEEE Standard Glossary of Software Engineering Terms [56] apibrėžia metriką kaip kiekybinį matą, kuris parodo kokiu laipsniu sistema, komponentas ar procesas pasižymi tam tikru atributu. Apibrėždami metrikas įvertinsime egzistuojančius problemų sprendimų būdus ir palyginsime su pasiūlytais. Problemos sprendimo būdams įvertinti naudojamos tokios metrikos:

- Ar nereikia testų kompiliavimo.

  Modulio testas paprastai būna aprašytas kaip programavimo kalba išreikštas kodas. Norint įvykdyti testą, reikia sukompiliuoti testo programinį kodą ir jį vykdyti mobiliajame įrenginyje. Kitas variantas yra testą aprašyti kaip parametrų sąrašą ir pagal juos vykdyti testuojamo objekto metodus.

- Tinklo gedimas neįtakoja.

  Modulių testai gali būti perduodami mobiliam įrenginiui tinklu. Sutrikimas tinkle gali nutraukti testų vykdymą ir įtakoti testavimo rezultatus.

- Atminties sutaupymas.

  Mobiliuosiuose įrenginiuose yra nedideli atminties resursai ir gali neužtekti atminties visiems testams saugoti. Kiekvienas metodas bando išspręsti atminties sunaudojimo problemą. Įvertinama ar metodas sumažina testų sunaudojamą atminties kiekį.

- Atminties išplėtimas.

  Mobiliuosiuose įrenginiuose yra nedideli atminties resursai ir gali neužtekti atminties visiems testams saugoti. Kiekvienas metodas bando išspręsti atminties sunaudojimo problemą. Įvertinama ar metodas bando padidinti esamos atminties kiekį.

- Sparta.

  Modulių testavime taip pat svarbus faktorius yra testų vykdymo sparta. Įvertiname kaip pakinta testų vykdymo sparta pritaikius tam tikrą metodą.

- Rezultatų stebėjimas.

  Mobilieji įrenginiai paprastai turi menką vartotoją sąsają ir joje sudėtinga butų atvaizduoti testavimo rezultatus. Įvertiname rezultatų stebėjimo patogumą.

- Atitinka mobilaus įrenginio architektūrą.

  Modulių testai gali būti perkelti vykdyti kitur nei pats mobilus įrenginys. Šiuo atveju reikia įvertinti ar nauja aplinka realiai atitinka mobilųjį įrenginį.

- Modulių testai ir programa išreikšti ne ta pačia programavimo kalba.

  Modulio testas paprastai būna aprašytas kaip programavimo kalba išreikštas kodas. Modulio testas yra rašomas ta pačia programavimo kalba kaip ir testuojamas kodas. Įvertiname ar programavimo kalba gali būti kitokia (aktualu atliekant programinės įrangos reinžineriją [48], konvertuojant programos kodą į kitą programavimo kalbą).

Metodo įvertinimo metrikos ir jų galimos reikšmės pateiktos lentelėje 4.1. Lentelėje pateikiamas metrikos pavadinimas, galimos reikšmės ir maksimalus skaitinis įvertinimas.

4.1 lentelė. Metodo metrikos ir jų reikšmės

Metrika	Reikšmės	Maksimalus įvertinimas
Nereikia testų kompiliavimo.	Taip/Ne	1
Tinklo gedimas neįtakoja	Taip/Ne	1
Atminties sutaupymas	Didelis/Vidutinis/Mažas	3
Atminties išplėtimas	Didelis/Vidutinis/Mažas	3
Sparta	Padidėja/Nepakinta/Sumažėja	2
Rezultatų stebėjimas	Patogus/Nepatogus	2
Atitinka mobilaus įrenginio architektūrą	Taip/Ne	1
Modulių testai ir programa išreikšti ne ta pačia programavimo kalba	Taip/Ne	1

Kiekviena reikšmė išreiškiama tam tikru skaitiniu įvertinimu, kuriuos vėliau galima susumuoti ir pateikti bendrą įvertinimą. Reikšmių skaitiniai įverčiai pateikti 4.2 lentelėje.

4.2 lentelė. Reikšmių skaitinis įvertinimas

Reikšmė	Skaitinis įvertinimas
Taip/Ne	Taip = 1, Ne = 0
Didelis/Vidutinis/Mažas	Didelis = 3, Vidutinis = 2, Mažas = 1
Padidėja/Nepakinta/Sumažėja	Padidėja = 2, Nepakinta = 1, Sumažėja = 0
Patogus/Nepatogus	Patogus = 2, Nepatogus = 0

4.2. Metodų palyginimas

Pagal 4.1 skyrelyje pateiktas metrikas įvertinami analizuojami metodai. Jų įvertinimas pateiktas 4.3 lentelėje.

4.3 lentelė. Metodų palyginimas

Kriterijus	Emuliatorius	Mock objektai	Perteklinių testų pašalinimas	RUT (C)[1]	RUT(U)[2]
Rezultatų stebėjimas	Patogus (2)	Nepatogus (0)	Nepatogus (0)	Patogus (2)	Patogus (2)
Atitinka mobilaus įrenginio architektūrą	Ne (0)	Taip (1)	Taip (1)	Taip (1)	Taip (1)
Nereikia testų kompiliavimo.	Ne (0)	Ne (0)	Ne (0)	Ne (0)	Taip (1)
Tinklo gedimas neįtakoja	Taip (1)	Taip (1)	Taip (1)	Ne (0)	Ne (0)
Atminties sutaupymas	Mažas (0)	Vidutinis (1)	Vidutinis (1)	Mažas (0)	Mažas (0)
Atminties išplėtimas	Didelis (2)	Mažas (0)	Mažas (0)	Didelis (2)	Didelis (2)
Sparta	Padidėja (2)	Padidėja (2)	Padidėja (2)	Sumažėja (0)	Sumažėja (0)
Rezultatų stebėjimas	Patogus (2)	Nepatogus (0)	Nepatogus (0)	Patogus (2)	Patogus (2)
Atitinka mobilaus įrenginio architektūrą	Ne (0)	Taip (1)	Taip (1)	Taip (1)	Taip (1)
Modulių testai ir programa išreikšti ne ta pačia programavimo kalba	Ne (0)	Ne (0)	Ne (0)	Ne (0)	Taip (1)
Bendras įvertinimas	9	6	6	8	10

Įvertinus metodus pagal minėtus kriterijus iš suminių rezultatų matome, kad geriausiai sprendžiamas problemas išsprendžia nuotolinis modulių testavimas su universaliu valdikliu. Taip pat neblogai problemą sprendžia emuliatoriaus sprendimo būdas. Tik nuotolinio modulio testavimo ir emuliatoriaus būdai padeda išplėsti rezultatų stebėjimo galimybes. Tinklo gedimai gali nutraukti testavimo procesą tik taikant nuotolinio modulių testavimo metodą.

[1] Nuotolinis modulių testavimas (konkretus valdiklis)

[2] Nuotolinis modulių testavimas (universalus valdiklis)

Taikant visus metodus išskyrus nuotolinį modulių testavimą padidėja testavimo sparta. Nepriklausomumas nuo programavimo kalbos gaunas tik taikant nuotolinį modulių testavimą, kuris naudoja universalaus valdiklio būdą.

Nuotolinio modulių testavimo metodas pateikia geras galimybes stebėti rezultatus, taip pat sumažina atminties sunaudojamumą mobiliajame įrenginyje, bet naudojant šį būdą sumažėja testavimo sparta.

5. Eksperimentinė dalis

5.1. Tikslas

Kaip autorius Peter J. Denning savo straipsnyje „Is Computer Science Science ?“ minėjo, kad tik 40% kompiuterių mokslų teorijų yra pagrįstos eksperimentiškai [8]. Norėdami patikrinti ar veikia nuotolinis modulių testavimas mobiliesiems įrenginiams, realizavome nuotolinį modulių testavimo karkasą ir išbandėme jo efektyvumą palygindami su kitais egzistuojančiais problemos sprendimo būdais. Matavome ir lyginome:

- Testavimo vykdymo spartą.
- Įvykdytų testų kiekį.
- Sunaudojamą atminį mobiliajame įrenginyje.
- Testavimo rezultatų stebėjimo patogumą.

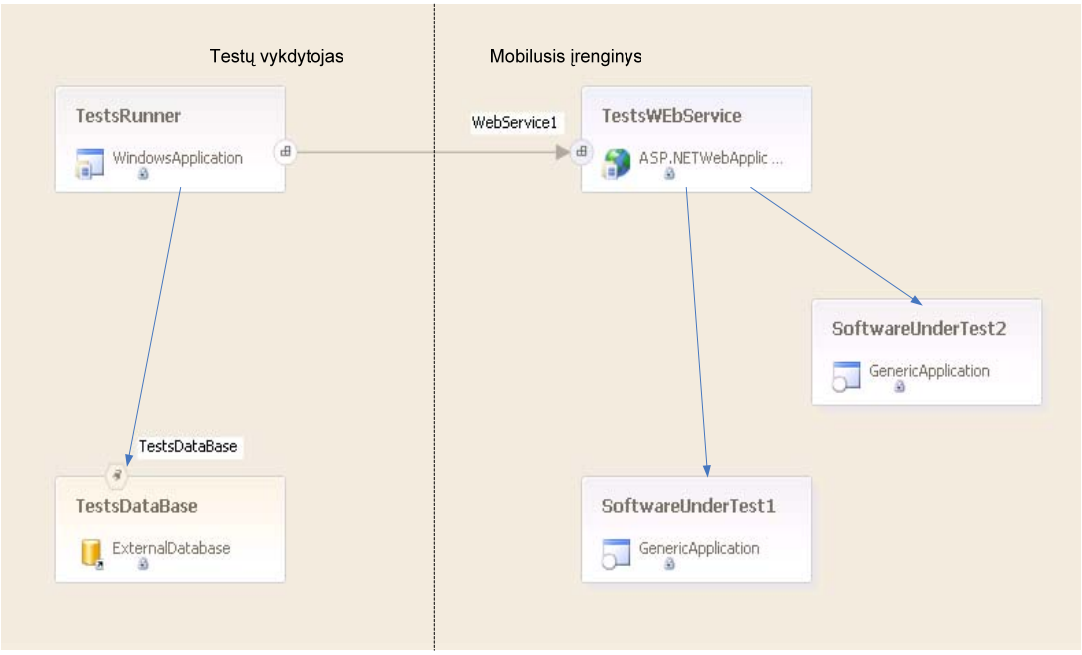
5.2. Realizacija

Nuotolinis modulių testavimas buvo realizuotas ir išbandytas kuriant mobilaus dėstytojo programinę įrangą. Programinė įranga buvo realizuota PocketPC tipo delniniame kompiuteryje, naudojant Microsoft .NET technologiją.

Siekiant sėkmingai ištestuoti šią programinę įrangą buvo sukurtas nuotolinis modulių testavimo karkasas. Jis buvo realizuotas naudojant Microsoft .NET technologiją. Jis susideda iš kelių dalių:

- Testų vykdytojas mobiliajame įrenginyje (TestsRunner).
- Testų vykdytojas valdančiajame įrenginyje (TestsWEbService).

Paveikslėlyje 5.1 pateiktas nuotolinio modulių testavimo realizacijos vaizdas.



5.1 pav. Nuotolinio modulių testavimo karkaso realizacija

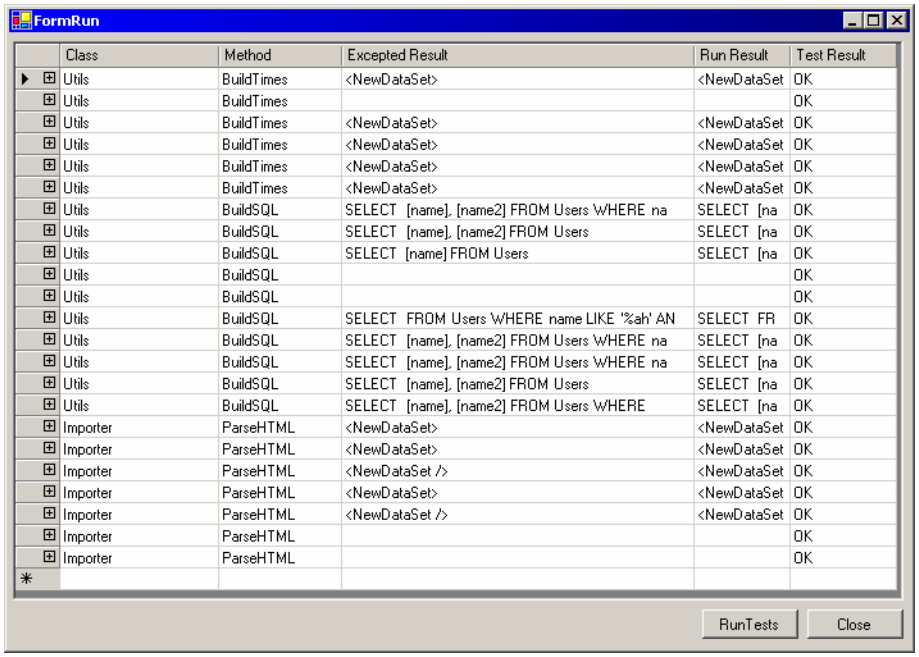
Testų vykdytojas valdančiajame įrenginyje buvo realizuotas kaip Microsoft .NET programa staliniame kompiuteryje. Jis naudoja Microsoft SQL 2000 duomenų bazę, kurioje saugo testus. Testų vykdytojas yra atsakingas už:

- testų išrinkimą,
- testų perdavimą testų vykdytojui mobiliajame įrenginyje,
- rezultatų pasiėmimą iš testų vykdytojo mobiliajame įrenginyje,
- testavimo ataskaitos vaizdavimą.

Testų vykdytojas mobiliajame įrenginyje realizuojamas kaip XML žiniatinklio paslauga. Jis teikia sąsają, per kurią galima pateikti testus vykdymui ir pasiimti testavimo rezultatus. Ji vykdo jam perduotus testų vykdytojo testus. Testų vykdytojas mobiliajame įrenginyje yra atsakingas už:

- testų priėmimą iš testų vykdytojo,
- testuojamos programinės įrangos išrinkimą,
- testų vykdymą,
- testo rezultato paėmimą,
- testo rezultato perdavimą testų vykdytojui valdančiajame įrenginyje.

Testų vykdytojo valdančiajame įrenginyje vartotojo sąsaja pateikta paveikslėlyje 5.2.



5.2 pav. Testų vykdytojo valdančiajame įrenginyje vartotojo sąsaja

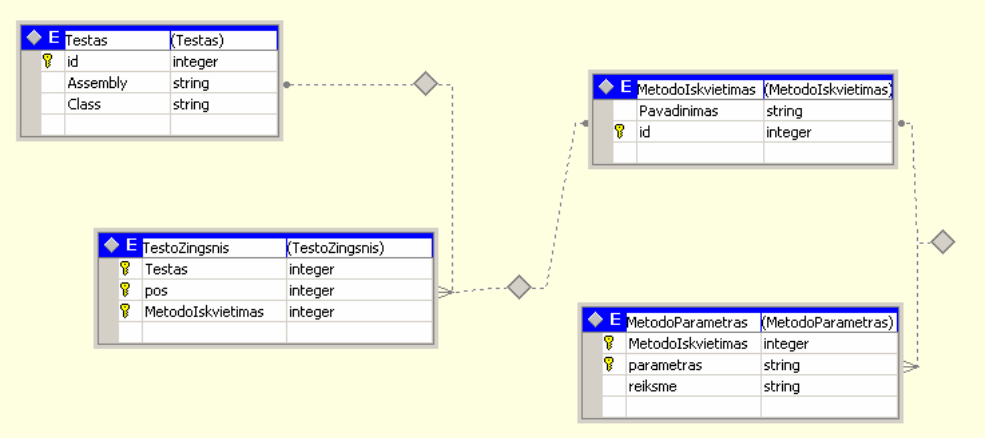
Testų vykdytojas valdančiajame įrenginyje leidžia pasirinkti testus juos vykdyti, pateikia testavimo rezultatus. Jis yra atsakingas už viso modulių testavimo vykdymo procesą, jį koordinuoja.

Norėdami detaliau paanalizuoti sukurtą metodą, realizavome nuotolinį modulių testavimo karkasą dviem būdais:

- Universalaus valdiklio
- Konkretaus valdiklio

5.3. Universalaus valdiklio realizacija

Realizuojant universalaus valdiklio būdą, modulių testai yra saugomi deklaratyvioje formoje duomenų bazėje. Duomenų bazės schema pateikta 5.3 paveikslėlyje. Testo vykdymo sekos yra išsaugomos kaip duomenys.



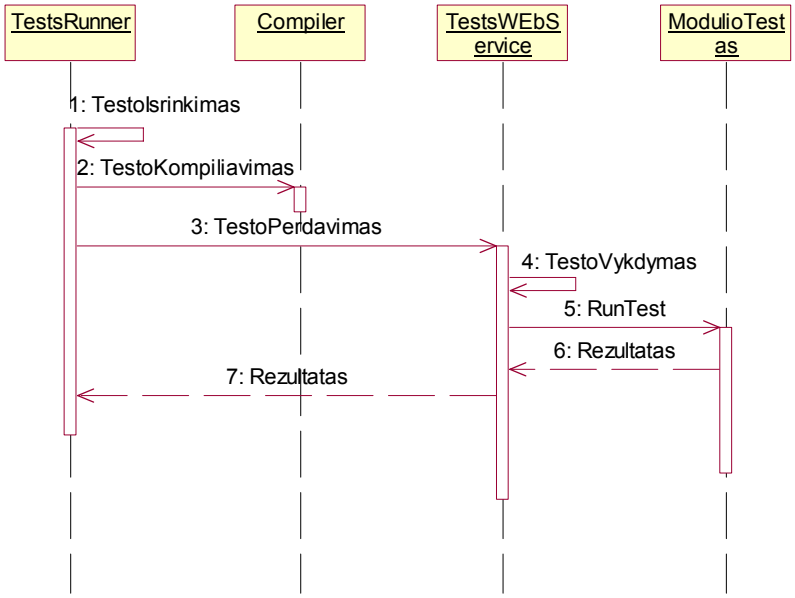
5.3 pav. Modulio testo duomenų struktūra

Vykdant modulio testą, testų vykdytojas valdančiajame įrenginyje išrenka testą iš duomenų bazės ir perduoda jo duomenis testų vykdytojui mobiliajame įrenginyje. Gavus testo duomenis, mobiliajame įrenginyje vykdoma perduotuose duomenyse apibrėžta testo vykdymo seka. Metodų iškvietimams, kurie sudaro testavimo seką, yra naudojami atspindžiai [3]. Taip gaunamas universalus valdiklis, kuris pagal pateiktus testo duomenis jį įvykdo. Įvykdžius testą, perduodami rezultatai testų vykdytojui, valdančiajame įrenginyje.

Mobilus įrenginys buvo prijungtas prie tinklo naudojant bevielį ryšį (WiFi). Testų sudarymui naudojama programinės įrangos meta data. Pagal ją išrenkamos programinės įrangos klasės, metodai ir lentelių pavidalu suvedami metodų parametrai ir laukiamos grąžinamos reikšmės. Vėliau, tokie suvesti testiniai atvejai po vieną siunčiami į mobilųjį įrenginį. Mobiliajame įrenginyje sukuriamas nurodytas objektas, iškviečiamas nurodytas metodas ir perduodami nurodyti parametrai. Fiksuojama grąžinama reikšmė ir ji perduodama į testavimo serverį, kuriame atliekamas rezultatų palyginimas su laukiamais rezultatais ir formuojama testavimo ataskaita, kuri vėliau rodoma serveryje.

5.4. *Konkretaus valdiklio realizacija*

Testai yra saugomi kaip programinis J# kalbos kodas. Prieš testus vykdant, jie yra sukompiliuojami į atskiras bylas. Jos yra perduodamos tinklu į mobilųjį įrenginį, naudojantis TestsWEbService paslaugomis. Perdavus modulio testą į mobilųjį įrenginį, testas yra paleidžiamas. Modulio testo vykdymo seka pateikta 5.4 paveikslėlyje.



5.4 pav. Modulio testo vykdymas

Kadangi testas yra vykdomas programinis kodas, tai užtenka tik sukurti testo objekto kodą ir vykdyti jo testavimo metodą, kuris griežtai apibrėžtas modulio testo sąsajos. Testų vykdytojas mobiliajame įrenginyje perima modulio testo vykdymo rezultatus ir juos grąžina testų vykdytojaui valdančiajame įrenginyje. Testo vykdymui taip pat reikalingi atspindžiai [3].

5.5. Palyginimo kriterijai

Atliekant 2.10 skyrelyje paminėtų metodų eksperimentinį tyrimą, buvo nagrinėjamos tokios jų išmatuojamos charakteristikos:

- Testavimo vykdymo sparta.

  Matuojama, kiek laiko trunka apibrėžto modulių testų kiekio vykdymo laikas. Vykdymo trukmė išreiškiama SI sistemos matavimo vienetu - sekunde. Matuojamas laiko tarpas, kuris praėjo, kol buvo įvykdyti visi testai.
- Įvykdytų testų kiekis.

  Matuojama, kiek iš sugeneruotų testų buvo įvykdyta mobiliajame įrenginyje. Į įvykdytų testų kiekį įtraukiami ir tie testai, kurie buvo atmesti metodų kaip pertekliniai. Įvertinimas pateikiamas kaip testų kiekis išreikštas vienetais.
- Sunaudojama atmintis mobiliajame įrenginyje.

  Matuojama, kiek atminties reikia, kad atlikti modulių testus mobiliajame įrenginyje. Į atminties sunaudojimą įskaičiuojama modulių testavimo karkaso dalis, kuri būna

įdiegta mobiliajame įrenginyje, modulių testai, kurie yra saugomi mobiliajame įrenginyje. Įvertinimas pateikiamas kaip atminties kiekis įvertintas baitais.

- Rezultatų stebėjimas.

  Matuojama kiek ekraninių langų reikia, kad peržiūrėti visus modulių testų vykdymo rezultatus. Ekranine forma laikoma 1024x768 ekranas, jei vaizduojama staliniame kompiuteryje ir 240x320 ekranas, jei vaizduojama mobiliajame įrenginyje.

5.6. Procesas

Eksperimentams atlikti buvo naudojamas stalinis ir delninis kompiuteriai:

- Stalinis
  Pentium IV 3GHz,
  1GB RAM
  1024x768 ekranas
- Delninis
  Intel XScale 400MHz, 128 MB RAM
  240x320 ekranas

Eksperimentų metu buvo automatiškai generuojami modulių testai 3 skyriuje aprašytai programinei įrangai. Sugeneruotų testų kiekiai ir programinės įrangos dydžiai pateikti lentelėje 5.1.

5.1 lentelė. Programinės įrangos ir modulių testų dydžiai

Parametras	Reikšmė
Klasės	23 vnt.
Metodai	287 vnt.
Modulių testai	675 vnt.
SLOC	6414 eil.
Testų SLOC	19538 eil.
Testų dydis	904 KB

Sugeneruoti testai buvo vykdomi mobiliuosiuose įrenginiuose, taikant problemų sprendimo metodus.

Emuliatoriaus sprendimo metodu, emuliatorius buvo sukurtas staliniame kompiuteryje, jo ekrano rezoliucija buvo išplėsta iki 1024x768 taškų, atminties kiekis nustatytas 1 GB, emuliuojama architektūra - x86.

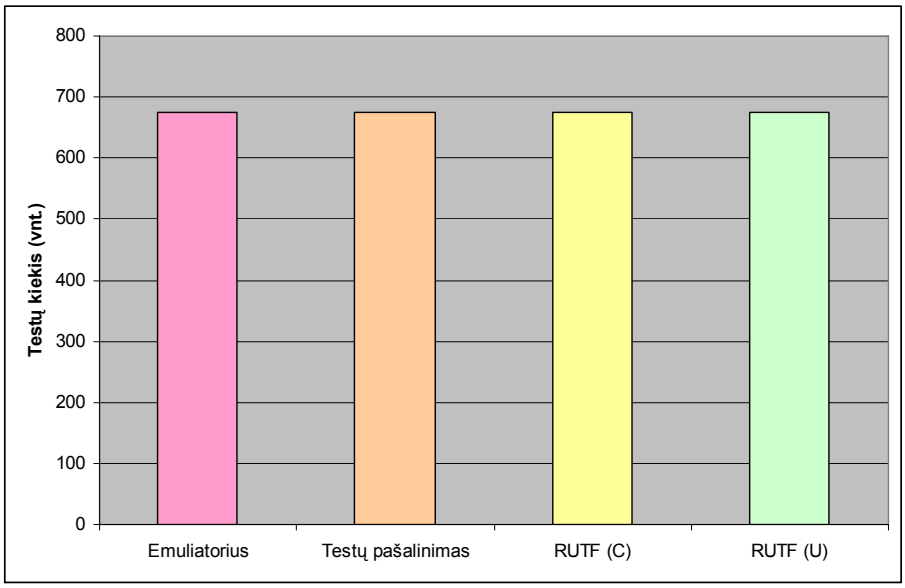
Taikant MockObjects, Rostra metodus, testavimas vyko realiame mobiliajame įrenginyje. Testų vykdymo pradžioje buvo taikomi metodai sumažinti testų kiekiui. Rezultatai taip pat stebimi mobiliajame įrenginyje. Metodų pašalinti testai kaip pertekliniai, suskaičiuojami kaip įvykdyti.

Taikant nuotolinį modulių testavimą, testai buvo sutalpinti staliniame kompiuteryje, testų vykdytojas valdančiame įrenginyje buvo paleistas staliniame kompiuteryje. Testų vykdytojas mobiliajame įrenginyje buvo paleistas delniniame kompiuteryje. Stalinis ir delninis kompiuteriai buvo sujungti USB jungtimi.

Eksperimentų metu buvo matuojamas laikas per kurį įvykdomi visi testai. Taip pat matuojama, kiek testavimo rezultatų teksto eilučių buvo atspausdinta. Atliekant laiko matavimą testai buvo pakartotinai vykdomi 100000 kartų.

5.7. Palyginimo rezultatai

Įvykdžius testus gavome rezultatus, kurie pateikti lentelėje 9.9. priede B. Paveikslėliuose 5.5, 5.6, 5.7 ir 5.8 pateikti eksperimentų rezultatai. Paveikslėliuose užrašas „Emuliatorius“ žymi testų vykdymą emuliatoriuje, užrašas „Testų pašalinimas“ žymi modulių testavimo vykdymą prieš tai apšalinus perteklinius testus, užrašas „RUTF (C)“ žymi nuotolinį modulių testavimą naudojant konkretaus valdiklio būdą ir užrašas „RUTF (U)“ žymi nuotolinį modulių testavimą naudojant universalaus valdiklio būdą.



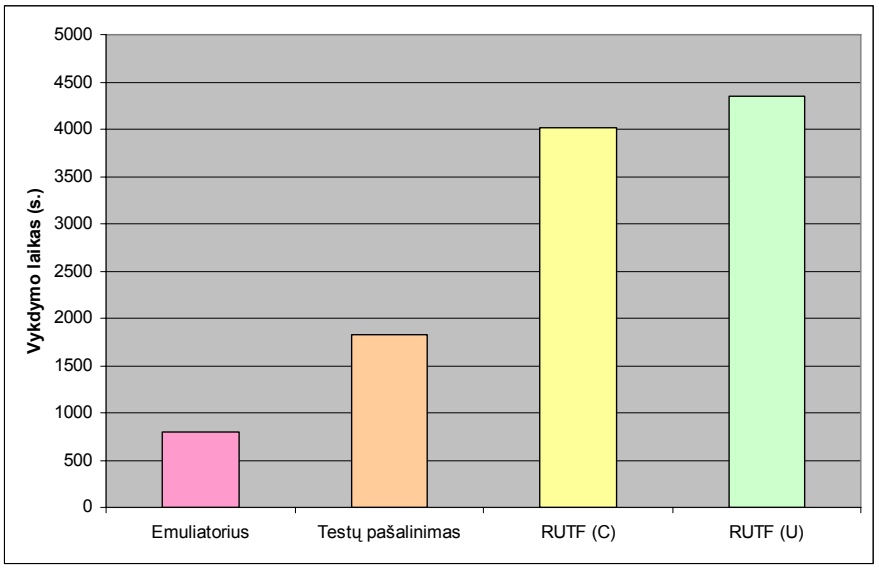
5.5 pav. Įvykdytų testų kiekis

Vykdant eksperimentus, visi automatiškai sugeneruoti modulių testai buvo įvykdyti, nes:

- Emuliatorius turėjo pakankamai atminties testams saugoti.
- Pašalinus perteklinius testus, visi modulių testai sutilpo į mobilų įrenginį.
- Naudojant nuotolinius modulių testavimas testai buvo saugomi išoriniame kompiuteryje.

Testų įvykdymo kiekiai pateikti 5.5 paveikslėlyje.

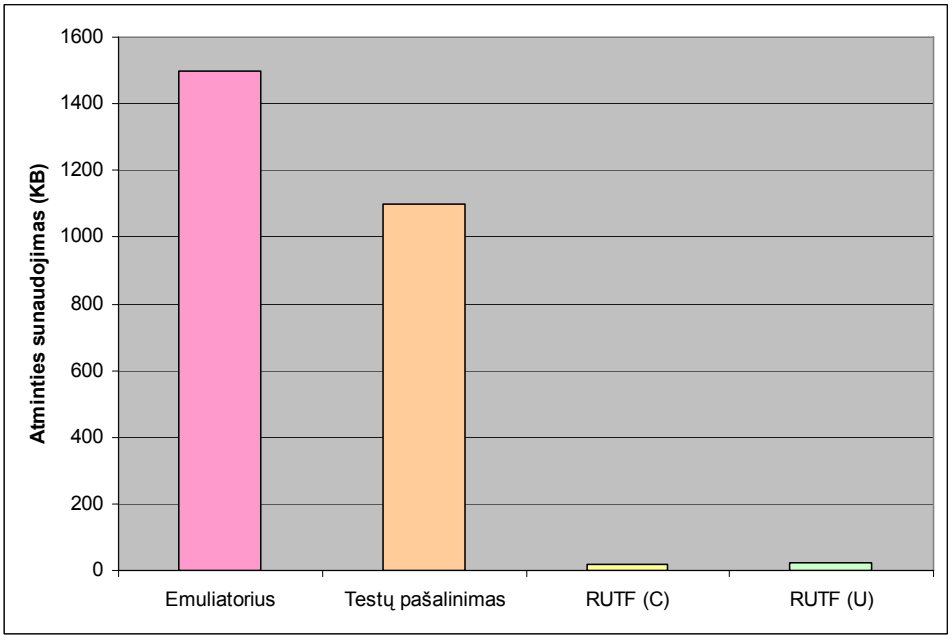
Matuojant testų įvykdymo laiką, nuotolinis modulių testavimas pasirodė prasčiausiai, nes testų vykdymo laiko didžiąją dalį sudarė testų perdavimas tinklu. Testų vykdymo laikai pateikti 5.6 paveikslėlyje.



5.6 pav. Testų vykdymo laikas

Modulių testai įvykdyti greičiausiai emuliatorije, nes mobiliojo įrenginio emuliavimui naudotas žymiai galingesnis kompiuteris, dėl kurio ir emuliuojamas įrenginys buvo žymiai greitesnis. Testai testuojantis modulius, kurie bendradarbiauja su išoriniais įrenginiais buvo įvykdyti nesėkmingai. Nuotolinis modulių testavimas vyko du kartus lėčiau dėl pastovaus testų perdavinėjimo tinklu.

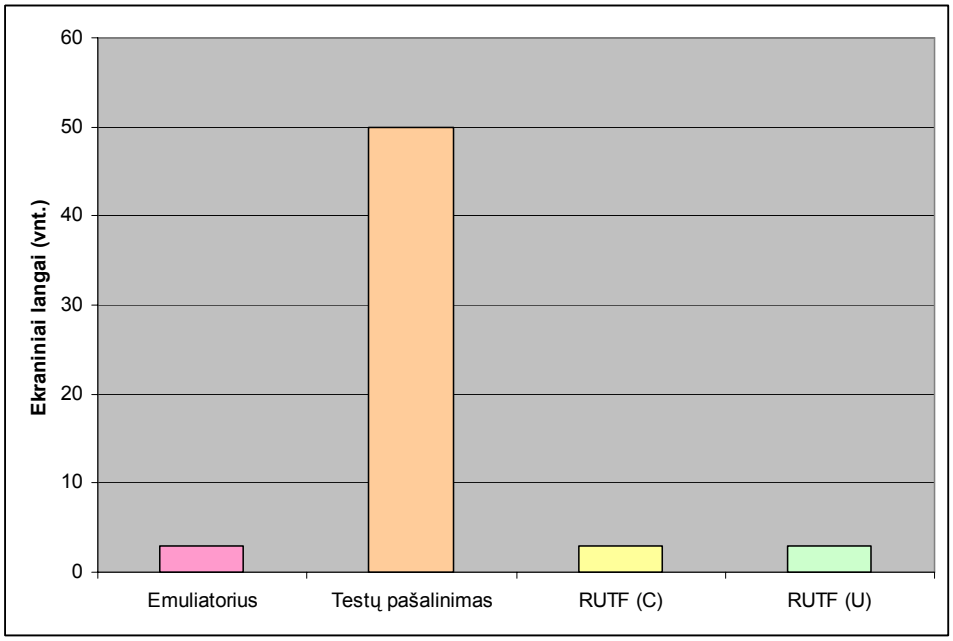
Paveikslėlyje 5.7 pateikti atminties mobiliajame įrenginyje sunaudojimo matavimų rezultatai.



5.7 pav. Atminties sunaudojimas

Emuliatoriaus atveju atminties sunaudojimo kiekis yra didžiausias, nes jame tiesiog sutalpinami visi testai. Perteklinių testų pašalinimo atveju atminties sunaudojimas mažesnis, nes vykdomi ne visi testai. Nuotolinio modulių testavimo atvejais atminties sunaudojimas yra visiškai mažas, nes naudojama atmintis tik vienam modulio testui.

Vienas iš svarbiausių eksperimentų buvo nustatyti, kaip patogu stebėti testavimo rezultatus. Patogumas išreikštas kiek reik peržiūrėti skirtingų ekraninių langų. Eksperimento rezultatai apteikti 5.8 paveikslėlyje.



5.8 pav. Ekraninių langų kiekis

Ekraninių langų kiekis yra didžiausias naudojant testų pašalinimo metodą. Didelis kiekis gaunamas todėl, kad rezultatai apteikiami mobiliame įrenginyje. Mažiausi ekraninių langų kiekiai, o tuo pačiu ir patogesnis stebėjimas, gautas naudojant nuotolinį modulių testavimą.

Eksperimentai parodė, kad nuotolinis modulių testavimas leidžia patogiau peržiūrėti testavimo rezultatus, naudoti maksimaliai mažai atminties mobiliajame įrenginyje vykdant modulių testus. Vienas nuotolinio modulių testavimo trūkumas yra tas, kad sulėtėja testų vykdymo sparta.

Remiantis eksperimentų rezultatais galima teigti, kad emulaitoriaus metodas būtų geriausias sprendimas, tačiau emuliatoriaus metodų testai nėra vykdomo realiame įrenginyje ir kai kurie testai teikia ne tokius pačius rezultatus, kaip jie teiktų juos vykdant mobiliame įrenginyje.

6. Išvados

1. Mobilios technologijos įtakoja programinės įrangos kūrimo procesą, iškeldamos problemas:
    a) vartotojo sąsajos,
    b) techninės įrangos įvairovės ir apribojimų,
    c) energijos taupymo,
    d) testavimo,
    e) diegimo ir palaikymo.

2. Projekto vykdymo metu pasirinkti ir pasiūlyti sprendimai:
    a) XML Web paslaugos (mobilūs, web klientai),
    b) virtuali mašina (įvairių mobilių įrenginių palaikymas),
    c) nuotolinis modulių testavimas (išsamus testavimas).

3. Vykdant modulių testavimą mobiliuose įrenginiuose susiduriame su vietos testiniams atvejams saugoti ir rezultatų stebėjimo problemomis.

4. Pasiūlytas nuotolinis modulių testavimas:
    a) teikiantis galimybes vykdyti neribotą modulių testų kiekį,
    b) leidžiantis vykdyti modulių testavimą, net ir tuo atveju kai mobiliajame įrenginyje nėra pakankamai vietos patalpinti testavimo programai,
    c) leidžiantis patogiau stebėti testavimo rezultatus (apie 10 kartų daugiau informacijos viename ekraniniame lange),
    d) leidžia atlikti regresinį testavimą neišjungus testuojamos sistemos.

7. Literatūra

1. Abran, A., Moore, J.W. Guide to the software engineering body of knowledge. Los Alamitos, Calif.: IEEE Computer Society, 2004. 205 p.
2. Alda, S. Component-Based Self-Adaptability in Peer-to-Peer Architectures// ICSE '04: Proceedings of the 26th International Conference on Software Engineering: tarptautinės konferencijos pranešimų medžiaga [2004 m.]. 2004, p. 33-35.
3. Ancona, M., Cazzola, W. Implementing the essence of reflection: a reflective run-time environment// SAC '04: Proceedings of the 2004 ACM symposium on Applied computing: tarptautinės konferencijos pranešimų medžiaga [2004 m.]. 2004, p. 1503-1507.
4. Beck, K. Test-Driven Development By Example. Boston: Addison Wesley, 2002. 240 p.
5. Chen, Y.F.R., Petrie, C. Guest editor's introduction - Ubiquitous mobile computing// IEEE Internet Computing. ISSN 1089-7801. 2003, Nr. 7, p. 16.
6. Cols, D.R. Business-structured client/server: an architecture for distributed applications// CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: tarptautinės konferencijos pranešimų medžiaga [1993 m.]. 1993, p. 41-53.
7. Demillo, R.A., Offutt, A.J. Experimental results from an automatic test case generator// ACM Trans. Softw. Eng. Methodol. ISSN 1049-331X. 1993, Nr. 2, p. 109-127.
8. Denning, P.J. Is computer science science?// ACM Commun. ISSN 0001-0782. 2005, Nr. 48, p. 27-31.
9. Feathers, M. Working Effectively with Legacy Code. Boston: Prentice Hall PTR, 2004. 456 p.
10. Fowler, M., Beck, K. Refactoring: improving the design of existing code. Boston: Addison-Wesley, 1999. 431 p.
11. Freeman, S., Mackinnon, T., Pryce, N., Walnes, J. Mock roles, objects// OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications: tarptautinės konferencijos pranešimų medžiaga [2004 m.]. 2004, p. 236-246.
12. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design patterns: elements of reusable object-oriented software. Boston: Addison-Wesley, 2003. 395 p.

13. Geer, D. Is It Time for Clockless Chips?// Computer. ISSN 0018-9162. 2005, Nr. 38, p. 18.
14. George, B., Williams, L. An initial investigation of test driven development in industry// SAC '03: Proceedings of the 2003 ACM symposium on Applied computing: tarptautinės konferencijos pranešimų medžiaga [2003 m.]. 2003, p. 1135-1139.
15. Hamill, P. Unit test frameworks. 1st ed. Sebastopol, CA: O'Reilly, 2005. 198 p.
16. Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., Jansen, E. The Gator Tech Smart House: A Programmable Pervasive Space// Computer. ISSN 0018-9162. 2005, Nr. 38, p. 50.
17. Highsmith, J., Cockburn, A. Agile software development: the business of innovation// Computer. ISSN 0018-9162. 2001, Nr. 34, p. 120.
18. Jain, S. Introduction to mobile computing// Crossroads. ISSN 1528-4972. 2000, Nr. 7, p. 2.
19. Jeon, T., Seung, H.W., Lee, S. Embedding built-in tests in hot spots of an object-oriented framework// SIGPLAN. ISSN 0362-1340. 2002, Nr. 37, p. 25-34.
20. Kakkad, S.V., Johnstone, M.S., Wilson, P.R. Portable run-time type description for conventional compilers// ISMM '98: Proceedings of the 1st international symposium on Memory management: tarptautinės konferencijos pranešimų medžiaga [Vancouver, British Columbia, Canada, 1998 m.]. Vancouver, British Columbia, Canada, 1998, p. 146-153.
21. Kerievsky, J. Refactoring to patterns. Boston: Addison-Wesley, 2005. 367 p.
22. Knauber, P., Succi, G. Perspectives on Software Product Lines// SIGSOFT Softw. Eng. Notes. ISSN 0163-5948. 2002, Nr. 27, p. 40-45.
23. Knowles, R. Automatic testing: systems and applications. London; New York: McGraw-Hill, 1976. 246 p.
24. Kochhar, V. OakUT - C++ unit test framework// Ubiquity. ISSN 2004, Nr. 5, p. 2.
25. Kroenke, D.M. Beyond the Relational Database Model// Computer. ISSN 0018-9162. 2005, Nr. 38, p. 89-90.
26. Kruchten, P. The rational unified process: an introduction. 3rd ed. Boston: Addison-Wesley, 2003. 336 p.
27. Lima, E.F.A., Fl, C.D.L.M., Figueiredo, A.R.S., Patr, J.C.A. An approach to modelling and applying mobile agent design patterns// SIGSOFT Softw. Eng. Notes. ISSN 0163-5948. 2004, Nr. 29, p. 1-8.
28. Mackenzie, D. Essential ClickOnce. Boston: Addison-Wesley Pub Co, 2004. 395 p.

29. Mackinnon, T., Freeman, S., Craig, P. Endo-testing: unit testing with mock objects. Extreme Programming Examined. Boston: Addison-Wesley, 2001. 301 p.
30. Matinlassi, M. Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA// ICSE '04: Proceedings of the 26th International Conference on Software Engineering: tarptautinės konferencijos pranešimų medžiaga [2004 m.]. 2004, p. 127-136.
31. Medvidovic, N., Mikic-Rakic, M., Mehta, N.R., Malek, S. Software Architectural Support for Handheld Computing// Computer. ISSN 0018-9162. 2003, Nr. 36, p. 66.
32. Milojicic, D.S., Douglis, F., Paindaveine, Y., Wheeler, R., Zhou, S. Process migration// ACM Comput. Surv. ISSN 0360-0300. 2000, Nr. 32, p. 241-299.
33. Munoz, M.A., Rodriguez, M., Favela, J., Martinez-Garcia, A.I., Gonzalez, V.M. Context-aware mobile communication in hospitals// Computer. ISSN 0018-9162. 2003, Nr. 36, p. 38.
34. Myers, B.A., Beigl, M. Handheld computing// Computer. ISSN 0018-9162. 2003, Nr. 36, p. 27.
35. Noble, J., Marshall, S., Marshall, S., Biddle, R. Less Extreme Programming// CRPIT '30: Proceedings of the sixth conference on Australian computing education: tarptautinės konferencijos pranešimų medžiaga [Dunedin, New Zealand, 2004 m.]. Dunedin, New Zealand, 2004, p. 217-226.
36. Packevičius, Š. Mobilių programų diegimas ir palaikymas// Informacinės technologijos - 2004: konferencijos pranešimų medžiaga [Kaunas, 2004 m. vasario 28 d.]. Kaunas, 2004, p. 20-22.
37. Packevičius, Š., Ušaniov, A. Vartotojo sąsajos delniniams kompiuteriams kūrimo principai// Informacinė Visuomenė ir Universitetinės Studijos: konferencijos pranešimų medžiaga [Kaunas, 2004 m. balandžio 15 d.]. Kaunas, 2004, p. 323-327.
38. Packevičius, Š., Ušaniov, A., Bareiša, E. Programinės įrangos testavimas mobiliuose įrenginiuose naudojant nuotolinį modulių testavimą// Informacinės Technologijos: konferencijos pranešimų medžiaga [Kaunas, 2005 m. balandžio 29 d.]. Kaunas, 2005, p. 177-180.
39. Pham, T.-L., Schneider, G., Goose, S. A situated computing framework for mobile and ubiquitous multimedia access using small screen and composite devices// MULTIMEDIA '00: Proceedings of the eighth ACM international conference on Multimedia: tarptautinės konferencijos pranešimų medžiaga [Marina del Rey, California, United States, 2000 m.]. Marina del Rey, California, United States, 2000, p. 323-331.

40. Qualitylogic. Mobile Device Testing Services. [žiūrėta 2005-04-22] prieiga internete http://www.qualitylogic.com/mobile_test.html.
41. Raatikainen, K., Christensen, H.B., Nakajima, T. Application requirements for middleware for mobile and pervasive systems// SIGMOBILE Mob. Comput. Commun. Rev. ISSN 2002, Nr. 6, p. 16-24.
42. Raghunath, M., Narayanaswami, C., Pinhanez, C. Fostering a symbiotic handheld environment// Computer. ISSN 0018-9162. 2003, Nr. 36, p. 56.
43. Richard N. Taylor and Nenad Medvidovic and Kenneth M. Anderson and E. James Whitehead, J.A.J.E.R. A component- and message-based architectural style for GUI software// ICSE '95: Proceedings of the 17th international conference on Software engineering: tarptautinės konferencijos pranešimų medžiaga [1995 m.]. 1995, p. 295-304.
44. Roman, G.-C., Picco, G.P., Murphy, A.L. Software engineering for mobility: a roadmap// ICSE '00: Proceedings of the Conference on The Future of Software Engineering: tarptautinės konferencijos pranešimų medžiaga [Limerick, Ireland, 2000 m.]. Limerick, Ireland, 2000, p. 241-258.
45. Saha, D., Mukherjee, A. Pervasive computing: a paradigm for the 21st century// Computer. ISSN 0018-9162. 2003, Nr. 36, p. 25.
46. Satoh, I. A testing framework for mobile computing software// IEEE Transactions on Software Engineering. ISSN 0098-5589. 2003, Nr. 29, p. 1112.
47. Sobering, G., Cook, L., Anderson, S. Pseudo-classes: very simple and lightweight mockObject-like classes for unit-testing// j-SIGPLAN. ISSN 0362-1340. 2004, Nr. 39.
48. Sommerville, I. Software engineering. 7th ed. Boston: Addison-Wesley, 2004. 759 p.
49. Stan, M.R., Skadron, K. Power-aware computing// Computer. ISSN 0018-9162. 2003, Nr. 36, p. 35.
50. Stobie, K. Too darned big to test// Queue. ISSN 1542-7730. 2005, Nr. 3, p. 30-37.
51. Tao, X., Notkin, D., Marinov, D. Rostra: a framework for detecting redundant object-oriented unit tests// ASE 2004: IEEE International Conference on Automated Software Engineering: tarptautinės konferencijos pranešimų medžiaga [Linz, Austria, 2004 m.]. Linz, Austria, 2004, p. 196.
52. Ušaniov, A., Packevičius, Š., Bareiša, E. Duomenų saugyklų taikymas mobiliose sistemose// Informacinės technologijos - 2005: konferencijos pranešimų medžiaga [Kaunas, 2005 m. sausio 29 d.]. Kaunas, 2005, p. 508-513.

53. Ušaniov, A., Packevičius, Š., Bareiša, E. Tinklo ryšio priemonės mobiliuose įrenginiuose// Informacinės technologijos - 2005: konferencijos pranešimų medžiaga [Kaunas, 2005 m. vasario 28]. Kaunas, 2005, p. 767-770.
54. Vainio, A.M., Tuunanen, T., Abrahamsson, P. Developing Software Products for Mobile Markets: Need for Rethinking Development Models and Practices// Hawaii International Conference on System Sciences - 2005: tarptautinės konferencijos pranešimų medžiaga [Hawaii, 2005 m.]. Hawaii, 2005, p. 189.
55. Vigna, G. Mobile agents: ten reasons for failure// MDM'04: IEEE International Conference on Mobile Data Management: tarptautinės konferencijos pranešimų medžiaga [2004 m.]. 2004, p. 298.
56. Zelenty, V.E. IEEE standard glossary of software engineering terminology// IEEE Std 610.12-1990. ISSN 0-7381-0336-5. 1990.
57. Zhu, H., Hall, P.A.V., May, J.H.R. Software unit test coverage and adequacy// ACM Comput. Surv. ISSN 0360-0300. 1997, Nr. 29, p. 366-427.

8. Terminų ir santrumpų žodynas

Automatinis testų generavimas

Automatinis modulių testų generavimas turint programinį kodą pagal tam tikrą kriterijų (pvz.: visų šakų padengimas).

Atspindžiai

(*Reflections*). Priemonės leidžiančios iš sukompiliuotos programos, jos bibliotekų gauti kodo meta duomenis.

COPA

(*A Component-Oriented Platform Architecting Method* [30]). Metodas skirtas kurti produktų šeimas. Jis yra paremtas komponentiniu programų kūrimu ir orientuotas į architektūros kūrimą.

Emuliatorius

Emuliatorius dubliuoja vienos sistemos funkcijas naudojantis kita sistema, tuo padarant, kad antros sistemos elgsena būtų tokia pati kaip ir pirmosios.

Emuliavimas

Žiūrėti Emuliatorius.

FAST

(*Family-Oriented Abstraction, Specification and Translation* [30]). Programinės įrangos kūrimo procesas, kuris pritaikytas kurti produktų šeimoms.

FORM

(*A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures* [30]). Metodas skirtas, kurti produktų linijoms (*product lines*). FORM yra FODA metodo išplėtimas. Dalykinės srities analizė ir jos panaudojimas kuriant prisitaikančius ir pakartotinai panaudojamus dalykines srities daiktus yra FORM pagrindas. FORM yra galimybėmis paremtas būdas produktų linijų architektūrų kūrimas.

Integracinis testavimas

(*Integration Testing*) testavimas, kuris tikrina programinės įrangos komponentų bendradarbiavimą.

KobrA

(*Komponentenbasierte Anwendungsentwicklung* [30]). Metodas skirtas produktų linijoms kurti, remiantis komponentinių projektavimu ir UML.

Kodo meta duomenys

Duomenys aprašantys sukompiliuotą programinę įrangą (klases, metodus, metodų parametrus). Naudojami aprašant pakartotinai naudoti skirtų bibliotekų sąsajas.

Mobilusis įrenginys

(*Mobile Device*). mobilus telefonas, protingas telefonas (*Smartphone*), delninis kompiuteris.

Modulis

(*Unit*). Mažiausias programinės įrangos komponentas, kurį galima atskirai ištestuoti (klasė, funkcija, metodas).

Modulio testas

(*Unit Test*) testas, kuris testuoja programinės įrangos dalių funkcionalumą izoliuotoje aplinkoje.

Modulių testavimo karkasas

(*Unit Test Framework*). Programinės priemonės atliekančios testų vykdymą, rezultatų pateikimą vartotojui.

Produktų linija

(*Product line*). Keli panašūs produktai, kuriami tos pačios organizacijos.

Produktų šeima

(*Product family*). Keli panašūs produktai, kuriami tos pačios organizacijos.

Regresinis testavimas

Selektyvus pakartotinis programų sistemos ar programinės įrangos komponentų, siekiant patikrinti, kad pakeitimai programinėje įrangoje nesukėlė netikėtų efektų.

RTTI

(*Run time type identification*). Programos vykdymo metų meta duomenų apie jos objektus identifikavimas.

Sistemos testavimas

Testavimas, kuris testoja visos sistemos elgseną.

Testų valdiklis

(*Test Driver*). Programa ar testavimo įrankis, naudojamas įvykdyti modulio testą. Jis susideda iš pradinių reikšmių nustatymo funkcijų, įėjimo reikšmių nustatymo funkcijų, kvietimų testuojamo objektų metodų, gautų rezultatų fiksavimo ir palyginimo su tikimomis gauti reikšmėmis.

Testavimo orakulas

(*Test Oracle*). Priemonės leidžiančios nustatyti ar reikiamas testas pateikė laukiamus rezultatus, ir nurodo testo įvykdymo sėkmę arba nesėkmę.

Testavimo ataskaita

Ataskaita, kuri pateikia testavimo rezultatus, kuriuose parodoma, kurie testai pavyko, kurie ne, kokie gauti rezultatai, kokius rezultatus buvo tikimasi gauti.

Testinis atvejis

(*Test Case*) Aibė testinių duomenų, vykdymo sąlygų ir laukiami rezultatai, kurie sukirti siekiant tam tikro tikslo (pvz.: patikrinti tam tikrą programos kelią ar patikrinti atitikimą reikalavimams).

Testavimo duomenys

(*Test Data*). Duomenys, kurie naudojami testuoti programinius modulius.

Testavimo serveris

Kompiuteris kontroliuojantis testų vykdymą mobiliame įrenginyje ir vaizduojantis testavimo rezultatus.

Testavimo rezultatai

Ataskaita, kuri pateikia testavimo rezultatus, kuriuose parodoma, kurie testai pavyko, kurie ne, kokie gauti rezultatai, kokius rezultatus buvo tikimasi gauti.

UML

(*Unified Modeling Language*). Unifikuotam modeliavimo kalba, leidžianti vizualiai modeliuoti programinę įranga.

Virtuali mašina

Programinė įranga, kuri izoliuoja programa nuo kompiuterio. Dėl to, kad virtualios mašinos yra pritaikomos įvairiems kompiuteriams, bet kuri programa parašyta virtualiai mašinai gali būti vykdoma tuose kompiuteriuose. Tos programos nereikia pritaikyti konkrečiam kompiuteriui ar jo operacinei sistemai.

QADA

(*Quality-driven architecture design and quality analysis method* [30]). Produktų linijos architektūros projektavimo metodas, kuris teikia priemones sekti produkto kokybę ir ją įvertinti net ir projektavimo metu.

XML Web paslauga

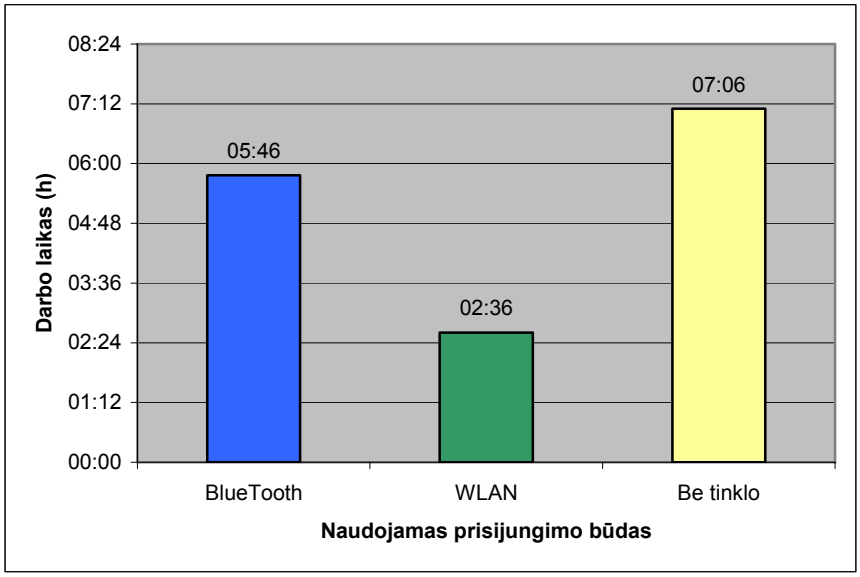
Protokolų ir standartų aibė, kuri naudojama duomenų pasikeitimui tarp programų. Programoms parašytos įvairiomis kalbomis ir vykdomos skirtingose platformose, kompiuteriuose, gali naudoti web paslaugas duomenų pasikeitimui kompiuterių tinklais, tokiais kaip internetas.

9. Priedai

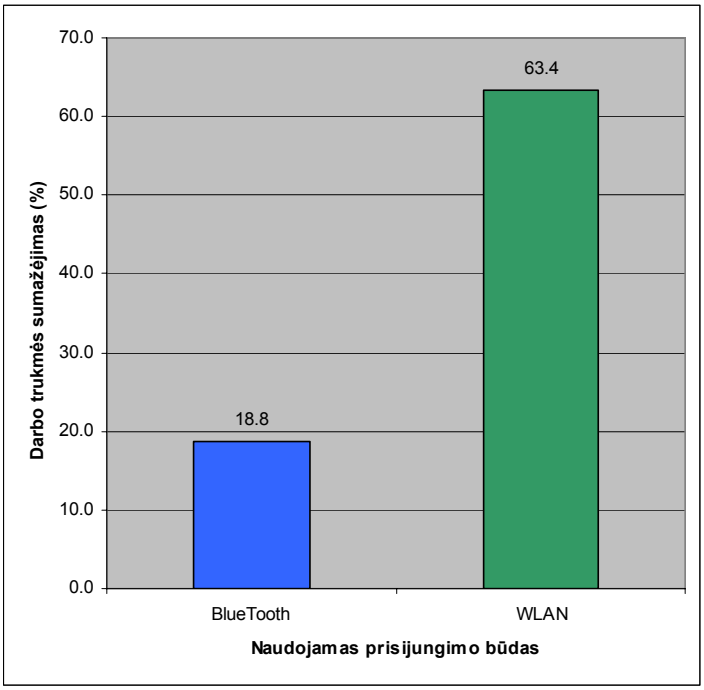
Priedas A: Energijos suvartojimas

9.1 lentelė. Darbo trukmės priklausomybė nuo prisijungimo būdo

Prisijungimo būdas	Darbo trukmė (h)	Sutrumpėjimas (%)
BlueTooth	05:46	18.8
WLAN	02:36	63.4
Be tinklo	07:06	0.0



9.1 pav. Darbo trukmės priklausomybė nuo prisijungimo būdo



9.2 pav. Darbo trukmės sutrumpėjimo priklausomybė nuo prisijungimo būdo

Priedas B: Eksperimentų rezultatų duomenys

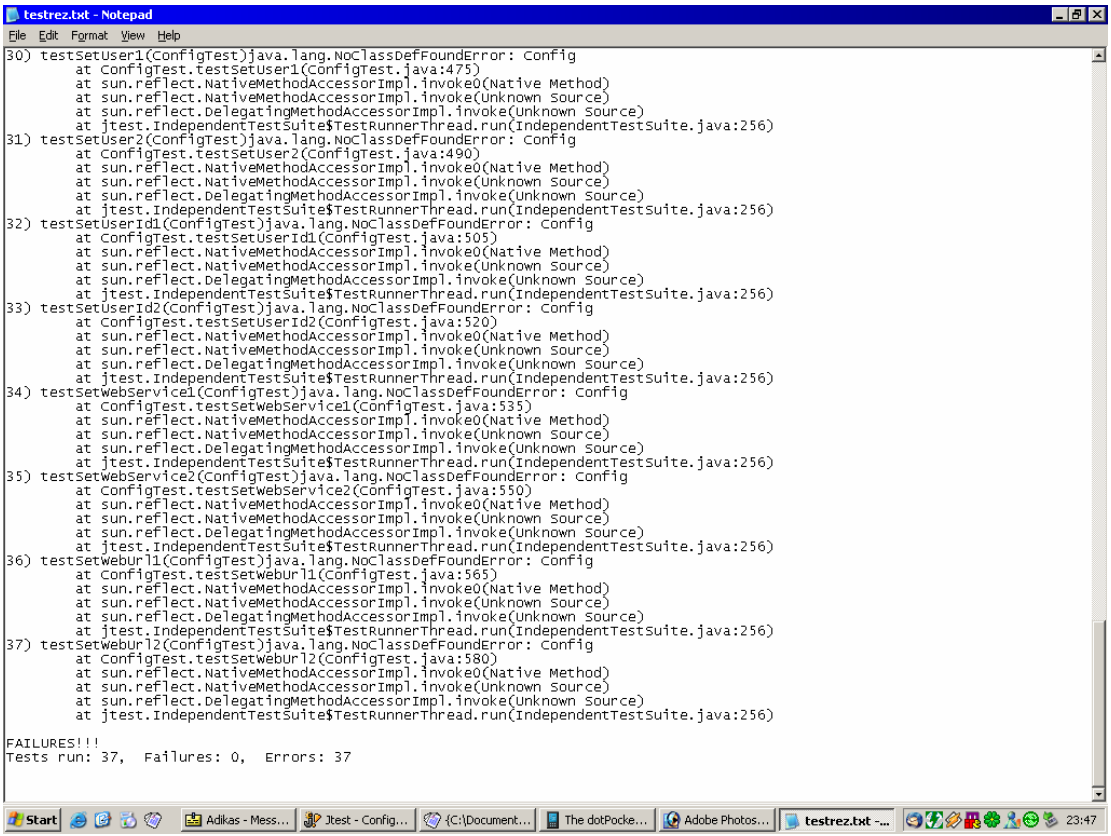
9.2 lentelė. Eksperimentų rezultatų duomenys

	Emuliatorius	Testų pašalinimas	RUTF (C)	RUTF (U)
Pradžia	1:32:33	8:15:18	13:05:13	18:03:28
Pabaiga	1:45:44	8:45:44	14:12:11	19:15:57
Trukmė	0:13:11	0:30:26	1:06:58	1:12:29
Vykdymo laikas (s.)	791	1826	4018	4349
Testų kiekis	675	675	675	675
Atmintis (KB)	1500	1100	18	21
Ekraninės formos	5	49	5	5

Priedas C: Testavimo rezultatų stebėjimo vartojo sąsajos



9.3 pav. Testavimo rezultatų stebėjimas mobiliajame įrenginyje



9.4 pav. Testavimo rezultatų stebėjimas staliniame kompiuteryje

Priedas D: Pranešimai konferencijose magistrinio darbo tema

1. Ušaniov, A., Packevičius, Š., Bareiša, E. Duomenų saugyklų taikymas mobiliose sistemose// Informacinės technologijos - 2005: konferencijos pranešimų medžiaga [Kaunas, 2005 m. sausio 29 d.]. Kaunas, 2005, p. 508-513.

2. Packevičius, Š., Ušaniov, A., Bareiša, E. Programinės įrangos testavimas mobiliuose įrenginiuose naudojant nuotolinį modulių testavimą// Informacinės Technologijos: konferencijos pranešimų medžiaga [Kaunas, 2005 m. balandžio 29 d.]. Kaunas, 2005, p. 177-180.

3. Packevičius, Š., Ušaniov, A. Vartotojo sąsajos delniniams kompiuteriams kūrimo principai// Informacinė Visuomėnė ir Universitetinės Studijos: konferencijos pranešimų medžiaga [Kaunas, 2004 m. balandžio 15 d.]. Kaunas, 2004, p. 323-327.

4. Packevičius, Š. Mobilių programų diegimas ir palaikymas// Informacinės technologijos - 2004: konferencijos pranešimų medžiaga [Kaunas, 2004 m. vasario 28 d.]. Kaunas, 2004, p. 20-22.

DUOMENŲ SAUGYKLŲ TAIKYMAS MOBILIOSE SISTEMOSE

Andrej Ušaniov, Šarûnas Packevičius , Eduardas Bareiša

Kauno Technologijos universitetas, Informatikos fakultetas, Programų Inžinerijos Katedra
Studentų g. 50, LT - 3031 Kaunas

Mobiliųjų įrenginių platus taikymas bei vartojimas informaciniuose sistemose buvo beprasmis dėl pačių įrenginių ribotų galimybių bei mobiliųjų duomenų saugyklų nebuvimo. Informacinės sistemos su mobiliais įrenginiais negalėdavo suteikti savo vartotojams pridėtinės vertės atitinkančios informacinės sistemos kainos. Tačiau situacija pasikeitė, atsiradus priemonėm, leidžiančios organizuoti mobiliąsias duomenų saugyklas. Šiuo metu mobilios informacinės sistemos kūrėjai susiduria su jų poreikius atitinkančios mobilios Duomenų Bazių Valdymo Sistemos pasirinkimo problema.

Pranešime apžvelgiamos duomenų saugyklų mobiliuose įrenginiuose organizavimo būdai bei technologijos: Java Record Management System, Microsoft Pocket PC platformos duomenų bazės, Palm OS platformos duomenų bazės, Išorinių duomenų bazių panaudojimas

1. Įžanga

Mobiliųjų įrenginių platus taikymas bei vartojimas informaciniuose sistemose buvo beprasmis dėl pačių įrenginių ribotų galimybių bei mobiliųjų duomenų saugyklų nebuvimo. Informacinės sistemos su mobiliais įrenginiais negalėdavo suteikti savo vartotojams pridėtinės vertės atitinkančios informacinės sistemos kainos. Tačiau situacija pasikeitė, atsiradus priemonėm, leidžiančios organizuoti mobiliąsias duomenų saugyklas. Šiuo metu mobilios informacinės sistemos kūrėjai susiduria su jų poreikius atitinkančios mobilios Duomenų Bazių Valdymo Sistemos pasirinkimo problema.

Kuriant programinė įranga mobiliems įrenginiams, duomenų bazė mobiliame įrenginyje numatoma norint tiekti vartotojui priėjimą prie jam reikiamų duomenų, greitą duomenų pasiimimą net ir dirbant atsijungus nuo tinklo. Tokio sprendimo atveju būna reikalinga r duomenų sinchronizacijos su pagrindine duomenų baze mechanizmai. Projektuojant tokia programinę įrangą tenka pasirinkti duomenų bazės valdymo sistemą mobiliam įrenginiui. Mobilaus įrenginio techniniai parametrai ir jo programinė įranga apriboja projektuotojo galimybes pasirinkti tinkamą duomenų bazės valdymo sistemą mobiliam įrenginiui.

2. Metrikos

Norint įvertinti duomenų bazių valdymo sistemų mobiliems įrenginiams galimybes ir jų tinkamumą kuriant programinę įrangą mobiliems įrenginiams buvo vertinami tokie parametrai, jie pateikti sekančioje lentelėje.

3. Duomenų bazės mobiliuose įrenginiuose

Analizuojamos tokios duomenų bazių valdymo sistemos mobiliems įrenginiams:

- Java Record Management System
- Microsoft SQL CE Server
- Oracle Database Lite
- DB2 Everyplace

3.1. Java Record Management System

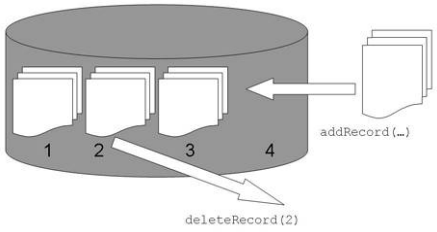
Kaip ir daugeliui desktop-based programų, MIDletams (Java mobiliosios programos) reikalinga pastovi duomenų saugojimo priemonė. Skirsime duomenys į dvi grupes: vartotojo duomenys ir programos duomenys. Vartotojo duomenų saugojimas yra labai svarbus. Tačiau mobiliųjų įrenginių galimybės, lyginant su desktop sistemomis, yra labai ribotos. Dėl šių apribojimų mobilus įrenginiai neturi įprastos failų sistemos koncepcijos. J2ME (Java 2 Micro Edition) bazinės klasės skirtos darbui su duomenų saugykla, neatvaizduoja duomenų paskirstymo į vartotojo ir programos grupes.

1 lentelė. Duomenų bazių valdymo sistemų mobiliems įrenginiams palyginimo kriterijai.

Kriterijus	Aprašymas
Platforma	Kokiuose delniniuose įrenginiuose galima naudoti. (PocketPC, PalmOS, Windows CE ir pan.)
Atminties sunaudojimas	Kiek reikia atminties mobiliame įrenginyje, kad suktųsi duomenų bazės valdymo sistema.
Duomenų bazės dydis	Kokio dydžio galima sukurti duomenų bazę mobiliame įrenginyje.
Duomenų bazės tipas	Reliacinė, failų sistema paremta.
Ryšys su kitomis DB	Ar yra galimybė sinchronizuoti duomenis su duomenų bazėmis esančiomis serveriuose.
Sinchronizavimo protokolas	Koks protokolas naudojamas sinchronizuoti duomenų bazes tarp mobilių įrenginių ir serverių. SyncML, ActiveSync ir pan.
Sinchronizavimo konfliktų sprendimas programiškai	Ar yra galimybės programiškai išspręsti sinchronizavimo metu iškilusias problemas. T.y. ar yra galimybė rašyti kokius plug-ins sinchronizavimo protokolui.
Sinchronizavimo konfliktų sprendimo vieta.	Serveryje, mobiliame įrenginyje. Programuojama atskirai.
Duomenų bazių kiekis	Kiek duomenų bazių galiam sukurti mobiliame įrenginyje.
Lentelių kiekis	Kiek galima sukurti lentelių duomenų bazėje mobiliame įrenginyje.
Laukų kiekis	Kiek galima sukurti lentelėje laukų.
Stored Procedūrų palaikymas	Ar galima naudoti Stored procedūras duomenų bazėje mobiliame įrenginyje.
SQL palaikymas	Kuria SQL versiją palaiko duomenų bazė mobiliame įrenginyje.
Palaikomi duomenų tipai.	Kokius duomenų tipus palaiko mobili duomenų bazė.
Duomenų bazės valdymo įrankiai	Ar yra duomenų bazės valdymo įrankiai pasiekiami mobiliame įrenginyje.
Programavimo priemonės.	Ar yra bibliotekos skirtos ADO.NET, ADO, ODBC, JDBC, native biblioteka.
Index palaikymas	Ar palaiko duomenų bazė indeksavimą.
Foreighn key palaikymas	Ar palaiko duomenų bazė Foreighn key.
Primary Key Palaikymas	Ar palaiko duomenų bazė Primary key.
Sinchronizavimas iš kelių serverių.	Ar galiam sinchronizuoti duomenis iš kelių serverių į ta pačia duomenų bazę mobiliame įrenginyje.
Sinchronizavimo stebėjimas	Ar yra programinės priemonės stebėti sinchronizavimo progresą.
Sinchronizavimas laukų.	Ar galima sinchronizuoti tik kelis laukus lentelėje vietoj visos eilutės.
Programų kiekis	Ar gali kelios programos mobiliame įrenginyje pasiekti tą pačią duomenų bazę.
Saugumas	Ar yra priemonės apsaugoti duomenų bazę mobiliame įrenginyje (šifravimas, autentifikavimas, autorizavimas)
Kaina	Kieka kainuoja licenzija vienam mobiliam įrenginiui.

Record Management System (RMS) tai įrašų pagrindu veikianti duomenų bazė. Įrašas – tai susijusios informacijos rinkinys apie esybę/objektą. Kiekvienas įrašas turi vienodą aibę laukų, kurių ilgiai yra fiksuoto dydžio.

Įrašų skaitymas/rašymas atliekamas per unikalų įrašų identifikatorių recordId, kuris yra pirminis raktas. RMS sistema yra atsakinga už recordId valdymą: naujų išskirimą, unikalumo palaikymą. Įrašų skaitymas vyksta ne baitų lygmenys, o įrašų lygmenyje, t.y. per vieną kartą galima įrašyti arba nuskaityti tik vieną įrašą. Įrašo turinys nėra svarbus RMS sistemai, ji mato įrašus kaip baitų masyvą. Vienintelis dalykas kuris rupi RMS tai recordId. Įrašų koncepcija RMS sistemoje pavaizduota 1 paveikslėlyje.



1 pav. Įrašų koncepcija RMS sistemoje

MIDletai yra pristatomi MIDletų rinkiniuose. Viename MIDletų rinkinyje yra mažiausiai vienas MIDletas. MIDletui sukūrus įrašų saugyklą, ji yra prieinama visiems MEDletų rinkinio MIDletams. Įrašų saugyklos pavadinimas turi būti unikalus MIDletų rinkinyje. MIDletas gali pasiekti įrašų saugyklą tik savo MIDletų rinkinyje.

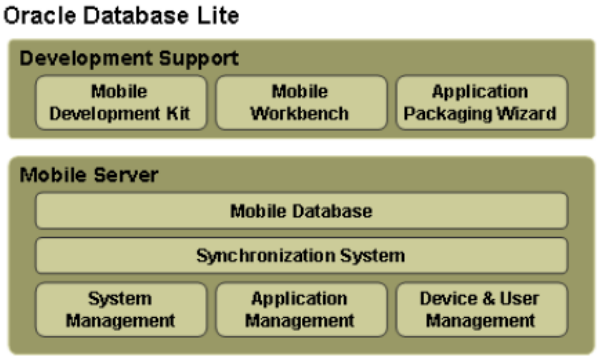
Programos kūrėjai tūrėtų skirti dėmesio RMS veikimo greitaveikos analizei. Kadangi priklausomai nuo J2ME platformos realizacijos, skirtingos RMS funkcijos veikia skirtingu greičiu. Pvz. Kai kuriose sistemose dirbant su dideliais duomenų kiekais ir norint atlinkti tam tikrų įrašų modifikacija yra efektyviau perskaityti visą saugyklos (RecordStore) turinį, ištrinti saugyklą ir sukurti naują, kurioje užsaugoti modifikuotus įrašus vietoj to, kad atnaujinti(update) įrašus skirtus modifikavimui.

3.2. Microsoft SQL CE Server

SQL Server CE yra maža duomenų bazė skirt avis dažniau kuriamoms programoms kurios išplečia duomenų valdymo galimybes iki mobilių įrenginių. SQL Server CE yra pilnateisis narys tarp SQL Server 2000 šeimos produktų. Turi įrankius skirtus DB valdymui, programavimo sąsajas (API), ir SQL sintakse, kuri yra pažystama visiems programuotojams dirbantiems su SQL Server produktais. SQL CE yra vienintelis produktas iš SQL Server 2000 šeimos, kuris teikia reliacinės duomenų bazės galimybes Windows CE paremtiems įrenginiams. SQL CE turi:

- optimizuojantį užklausų procesorių
- Transakcijų palaikymą.
- Duomenų tipų palaikymą.
- Mažai naudoja sistemos resursų.
- Remote Data Access ir Merge Repclication per HHTP protokolą.

3.3. Oracle Database Lite



2 pav. Oracle Database Lite sistemos sudėtis

Oracle Database Lite yra priedas prie Oracle duomenų bazės naudojamas dažnam kūrimui ir diegimui didelės svarbos, mission-critical programoms skirtoms mobiliems įrenginiams. Oracle Database Lite naudoja duomenų sinchronizavimą, kad patikimai ir saugiau apsikeistų duomenimis tarp Oracle Database ir nutolusios aplinkos. Darbuotojai gali naudotis kompanijos informacija ir atlikti jiems riekiamas funkcijas būdami atsijungę nuo kompanijos duomenų bazės. Kompanijos naudojamos Oracle Database Lite gali apdidinti darbuotojų produktyvumą,

sumažinti darbų sąnaudas, ar padidinti klientų pasitenkinimą. Sekančiame paveikslėlyje pateikta Oracle Database Lite sistemos vaizdas.

3.4. DB2 Everyplace

IBM DB2 Everyplace yra IBM produktas teikiantis duomenų bazės paslaugas mobiliems įrenginiams. DB2 Everyplace susideda iš trijų pagrindinių komponentų, tokių kaip duomenų bazės variklis, kuris sukasi mobiliame įrenginyje, sinchronizavimo serveris ir pagrindinė duomenų bazė, kuri sukasi desktop PC arba enterprise serveryje.

DB2 Everyplace leidžia vartotojams įsidėti fragmentus pagrindinės duomenų bazės į savo mobilius įrenginius ir sinchronizuoti juos su pagrindine duomenų baze. Duomenų bazės variklis mobiliame įrenginyje yra gana nereikus resursams jam užtenka net 200KB atminties.

Prie DB2 Everyplace teikiamas DB2 Everyplace's MAB įrankis leidžiantis sukurti programas, naudojančias DB2 Everyplace duomenų bazę nerašant kodo. Šis įrankis sugeneruoja programą kuri gali būti pritaikyta konkrečiam mobiliam įrankiui arba tiesiog J2SE kodas, kuris gali būti vykdomas betuokiame įrenginyje.

Duomenų sinchronizavimams vyksta naudojantis HTTP protokolu, sinchronizavimo serveryje yra java servletas, kursi gali būti patalpintas betuokiame servletus palaikančiame serveryje (Tomcat, WebSphere ir pan.).

DB2 Everyplace teikia įvairias priemones dirbti su duomenų baze, tokais kaip C/C++, Java, Visual Basic, .NET bibliotekas.

4. Palyginimo rezultatai

Išanalizavimus duomenų bazių valdymo sistemas skirtas mobiliems įrenginiams palyginimo rezultatai yra pateikti žemiau esančioje 2 lentelėje.

2 lentelė. Duomenų bazių valdymo sistemų mobiliems įrenginiams palyginimo kriterijai

DBMS / Kriterijus	Java Record Management System	Microsoft SQL CE Server	Oracle Database Lite	DB2 Everyplace
Platforma	PocketPC, PalmOS, Symbian, kitos	Windows CE, Pocket PC, Windows Mobile	Windows CE, Pocket PC, Windows Mobile, Windows NT/XP/2000/98, PalmOS, Linux	Linux, Neutrino, PalmOS, Symbian, Windows, Windows CE, Windows Pocket PC
Atminties sunaudojimas.	Priklauso nuo JVM realizacijos	Nedidelis	150KB – 1MB	137 KB
Duomenų bazės dydis	Ribojamas mobilaus įrenginio laisvos atminties kiekiu	Iki 2 GB	Iki 4 GB	Ribojamas mobilaus įrenginio laisvos atminties kiekiu
Duomenų bazės tipas	Įrašų pagrindu	Reliacinė.	Reliacinė.	Reliacinė
Ryšys su kitomis DB	Nėra	Tik Microsoft SQL Server.	Oracle	Per "DB2 Everplace's synchronization server" su **DB2**, Oracle, Microsoft SQL server, Domino and Exchange
Sinchronizavimo protokolas	Atskirai programuojamas	Vidinis (RDA, Merge Repclication).	Publish/Subscription modelis. Push modelis.	SyncML

Sinchronizavimo konfliktų sprendimas programiškai	Atskirai programuojamas	Yra	Yra	Yra
Sinchronizavimo konfliktų sprendimo vieta.	Programos projektuotojas sprendžia	Mobiliame įrenginyje, programiškai.	Sisteminė ir programuojama.	Sisteminė ir programuojama.
Duomenų bazių kiekis	Ribojamas mobilaus įrenginio laisvos atminties kiekiu	Neribotas.	Neribotas.	Neribotas
Lentelių kiekis	Ribojamas mobilaus įrenginio laisvos atminties kiekiu	Neribotas.	Neribotas.	Neribotas
Laukų kiekis	Ribojamas mobilaus įrenginio laisvos atminties kiekiu	Neribotas.	Neribotas.	Neribotas
Stored Procedūrų palaikymas	Nėra	Nėra	Yra.	Yra
SQL palaikymas	Nėra	Yra.	Yra. (SQL-92)	Yra. (SQL-99)
Palaikomi duomenų tipai.	Byte	INT, FLOAT, CHAR, VARCHAR.	Oracle DB tipai.	IBM DB2 tipai
Duomenų bazės valdymo įrankiai	Nėra	Yra (SQLCE Query)	Yra.	Yra
Programavimo priemonės.	J2ME API	ADOCE, ADO.NET, ADO.	JDBC, ADOCE, ADO, ADO.NET.	JDBC, ADO.NET
Index palaikymas	Nėra	Nėra.	Yra	Yra
Foreighn key palaikymas	Nėra	Yra.	Yra	Yra
Primary Key Palaikymas	Yra	Yra.	Yra	Yra
Sinchronizavimas iš kelių serverių.	Programuojamas atskirai	Nėra.	Nėra.	Nėra.
Sinchronizavimo stebėjimas	Nėra	Nėra.	Nėra.	Nėra.
Sinchronizavimas laukų.	Programuojamas atskirai	Nėra. Tik eilutės.	Nėra.	Yra
Programų kiekis	Neribotas	1 programa prie 1 duomenų bazės.	Neribotas.	Neribotas.
Saugumas	Programuojamas atskirai	Duomeny bazės šifravimas. Autentifikavimas.	Autentifikavimas, Autorizavimas, šifravimas.	Autentifikavimas, Autorizavimas, šifravimas.
Kaina	Nemokama	Nemokamai SQL Server 2000 Developer Edition vartotojams.	Nemokama.	Nemokama.

5. Išvados

SQL CE tinka jei programinės įrangos pagrindinės duomenų bazės yra realizuotos SQL Server 2000 duomenų bazėse. Jei aplinkoje naudojamos Oracle ar kitokios duomenų bazės SQL CE nebeturi prasmės, nes negali su jomis sinchronizuoti duomenų.

Nepaisant ribotų mobiliųjų įrenginių galimybių, RMS suteikia patogią ir lengvai naudojamą infrastruktūrą ilgalaikiam pastoviam duomenų kaupimui.

Oracle Database Lite yra gana puikus sprendimas, jei yra poreikis naudoti įvairaus tipo mobiliuose įrenginiuose. Galimybėmis aplenkia Microsoft SQL CE Server. Vienas minusas, kad sinchronizuojasi tik su Oracle duomenų bazėmis.

IBM DB2 Everyplace panašiai kaip Oracle Database Lite veikia įvairiose platformose teikia puikias sinchronizavimo galimybes. Bet taip pat kaip ir Oracle sinchronizuojasi tik su DB2 duomenų baze.

Literatūros sąrašas

[1] Access data anywhere with Everyplace. [žiūrėta 2004-12-15], prieiga internete http://www.infoworld.com/DB2_Everyplace_Enterprise_8.1.4/product_46523.html?view=1&curNodeId=0

[2] Java Database Review Places PointBase At The Top. [žiūrėta 2004-11-09], prieiga internete http://wirelessdev.weblogsinc.com/entry/4868643664242743/

[3] Mobile Memories: The MIDP Record Management System. [žiūrėta 2004-11-14], prieiga internete http://today.java.net/pub/a/today/2004/11/16/J2ME-3.html

[4] Oracle Database Lite Overview. [žiūrėta 2004-11-10], prieiga internete http://www.oracle.com/technology/products/lite/lite_datasheet_10g.pdf

[5] Palm OS Database Applications. [žiūrėta 2004-12-13], prieiga internete http://www.the-gadgeteer.com/databases-review.html

[6] SQL Server 2000 Product Overview. [žiūrėta 2004-11-16], prieiga internete http://www.microsoft.com/sql/evaluation/overview/default.asp

Data Stores Usage in Mobile Systems

Wide usage of mobile devices was quite useless because of the lack of resources in these devices and the lack of data storages for them. Information systems with mobile devices were unable to add additional value to theirs user for a given price. Though, situation has changed with increased resources and processing power on mobile devices. There are a lot of products of data storage systems in market. And developers of information systems encounter problems such as choosing a best Data Base Management System for mobile device.

In this article are described and analyzed data storages for mobile devices (Java Record Management System, DBMSD for Microsoft Pocket PC platform and Palm OS platform), and possibilities of using remote data bases.

PROGRAMINĖS ĮRANGOS TESTAVIMAS MOBILIUOSE ĮRENGINIUOSE NAUDOJANT NUOTOLINĮ MODULIŲ TESTAVIMĄ

Šarūnas Packevičius, Andrej Ušaniov, Eduardas Bareiša

Kauno Technologijos universitetas, Informatikos fakultetas, Programų Inžinerijos Katedra, Studentų g. 50, LT - 3031 Kaunas

Pasirodžius mobiliems įrenginiams, vis daugiau kuriama programinės įrangos mobiliems įrenginiams (delniniams kompiuteriams, mobiliems telefonams). Programinės įrangos kūrimas šiems įrenginiams iškelia naujų problemų jos kūrimo procese, ypač testavimo fazėje.

Šiame pranešime aptariamos testavimo fazėje, atliekant programinės įrangos mobiliuose įrenginiuose testavimą, iškylančios problemos, pateikiami testavimo metodai budingi testuojant programinę įrangą mobiliems įrenginiams, klasikinių testavimo metodų pritaikymo galimybės programinės įrangos testavimui mobiliuose įrenginiuose.

1 Įžanga

Patobulėjus mobilioms technologijoms, vis daugiau kuriama programinės įrangos mobiliems įrenginiams, tokiems kaip delniniai kompiuteriai, protingi mobilūs telefonai. Programinė įranga mobiliems įrenginiams dažniausiai yra skiriama plačiam vartotojų ratui. Kuriant programinę įrangą šiems mobiliems įrenginiams susiduriama su daugeliu problemų jos kūrimo procese. Problemos yra įtakotos specifikos susijusios su mobiliais įrenginiais. Esminiai apribojimai yra riboti resursai, tokie kaip sąlyginai lėti procesoriai, nedili atminties kiekiai, taip pat ribota vartotojo sąsaja[4].

Programinės įrangos projektavimo etape reikia ypatingą dėmesį atkreipti į šiuos apribojimus, taip pat svarbu atkreipti dėmesį ir į mobilių įrenginių įvairovę. Egzistuoja įvairiausių konfigūracijų, skirtingų techninių galimybių, architektūrų mobilių įrenginių. Visa tai turi būti įvertinta projektuojant programinę įrangą mobiliems įrenginiams, ypač tuo atveju kai programinė įranga skirta plačiam vartojimui.

Taip pat mobilios technologijos įneša ir savo specifiką programinės įrangos testavimo etape. Dėl techninės įrangos apribojimų sudėtinga ne tik sukurti programinę įrangą mobiliems įrenginiams bet ir taip pat sudėtinga ją testuoti.

Likusioje šio straipsnio dalyje aptariamos problemos iškylančios testuojant programinė įrangą mobiliems įrenginiams, pateikiami siūlomi būdai joms spręsti, pateikiamos technikos naudojamos testuojant programinę įrangą mobiliems įrenginiams, pateikiami konkretūs problemų sprendimai taikyti kuriant mobilaus dėstytojo programinę įrangą.

2 Testavimo problemos, uždaviniai

Mobilūs įrenginiai su savo specifika sukuria tokias problemas programinės įrangos testavimo procese:

- Ribotas atminties kiekis mobiliame įrenginyje neleidžia lokaliai jame saugoti testavimo atvejus. Atliekant automatinį testų generavimą yra pagaminami dideli kiekiai testų[5] ir jiems reikia saugoti nemažų atminties resursų, kurių mobiliuose įrenginiuose yra trūkumas.
- Nedideli ekranai mobiliuose įrenginiuose ir ribotos vartojo sąsajos galimybės apsunkina testavimo rezultatų stebėjimą, programinės įrangos derinimą(debug) ar kūrimą.
- Mobilių įrenginių įvairovė reikalauja ištestuoti programinę įrangą daugelyje įrenginių.
- Apribotos vartotojo sąsajos galimybės reikalauja atkreipti daugiau dėmesio į programinės įrangos panaudojamumo testavimą (usability testing).
- Kadangi programinė įranga mobiliuose įrenginiuose dažnai neegzistuoja izoliuota, o bendrauja su įvairiais aplinkiniais servisais, kita programine įrangą, svarbu atlikti išsamų bendradarbiavimo testavimą.

3 Testavimas

Dabartiniu metu gana sparčiai populiarėja testavimu paremtas programų kūrimas[2] (Test Driven Development). Jo principas yra, kad kiekvienam programinės įrangos komponento (objektinio programavimo atveju klasei), sukuriami testai, kurie ištestuoja komponento korektišką funkcionavimą. Testai turi būti sukuriami prieš programuojant komponentą ir toliau pastoviai vykdomi modifikavus komponentą, tuo užtikrinant, kad įvesti pakeitimai nesugriovė komponento. Tokie testai, kurie testuoja programinės įrangos komponentus, vadinami modulių testais (Unit tests).

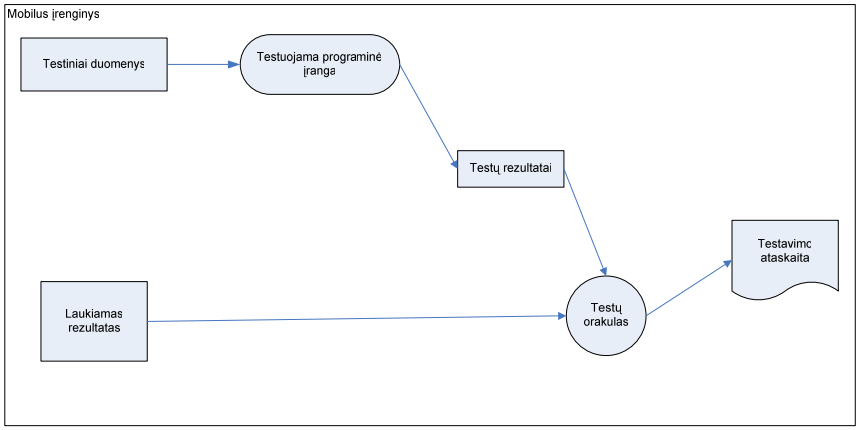
Modulio testas paprastai sukuria testuojamos klasės objektą, kviečia jos metodus su atitinkamais parametrais, tikrina gražinamų parametrų reikšmes, lygina jas su laukiamais rezultatais. Taip pat patikrina objekto būseną, ar ji perėjo į tokia kokią tikimasi.

4 Modulių testavimo karkasas

Įvedus dar vieną elementą vadinamą testų vykdytoju (Test runner), kuris surenka visus pagamintus modulių testus ir juos vykdo gauname modulių testavimo karkasus (Unit Test Framework), kurių pagalba galima automatizuoti testavimo procesą. Modulių testavimo karkasai yra esminiai elementai populiariose programinės įrangos kūrimo metodologijose, tokiose kaip extreme programming (XP)[3] ir Agile Development. Modulių testavimas jau viršijo extreme programming metodologijos ribas ir dabar dažnai sutinkamas įvairiose kituose programinės įrangos kūrimo metodologijose. Modulių testų pagalba galima užtikrinti žemiausio lygio programinės įrangos kodo teisingumą, sumažinti programinės įrangos kūrimo ciklo trukmę, padidinti produktyvumą ir leistu sukurti labiau patikimą programinę įrangą.

4.1 Standartinis

Paprastai modulių testai ir programinę įranga yra sudedama į vieną programą (testai įkompiliuojami) ir atliekant testavimą vykdoma ta pati programa tik kviečiamas ne jos vykdymas, o testai esantys joje. Tokio testavimo atveju visas modulių testavimo karkasas turi būti patalpintas į mobilų įrenginį ir vykdomas jame. Mobilaus įrenginio modulių testavimo karkasas pateiktas sekančiame paveikslėlyje.



1 pav. Modulių testavimo karkasas mobiliame įrenginyje.

Testiniai duomenys yra paruošiami, pagal juos sukuriamas programinis kodas vykdantis tuos testus (automatiškai arba rankiniu būdų), testai vėliau įkompiliuojami į programinę įrangą, patalpinami mobiliame įrenginyje. Atliekama testų vykdymas ir rezultatų (Testavimo ataskaitos) formavimas, stebėjimas mobiliame įrenginyje.

Pagrindinis trūkumas, kad viskas turi būti sutalpinta į mobilų įrenginį. Deja mobiliuose įrenginiuose atminties resursai yra riboti ir pakankamo testų kiekio neišeina sutalpinti. Taip pat dėl ribotų vartotojo sąsajos galimybių testų rezultatų stebėjimas yra problematiškas. Šiom problemoms spręsti siūlomas nuotolinis modulių testavimo karkasas.

4.2 Egzistuojantys sprendimai

Buvo pasiūlyti keli variantai efektyviai atlikti testavimą mobiliuose įrenginiuose išvengiant jų apribojimų. Vienas iš jų yra siūlymas naudoti mobilių įrenginių emulatorius[4]. Šis variantas leidžia sukurti mobilų įrenginį paprastame kompiuteryje įraukiant kiek reikia resursų. Tokiu būdų galima turėti didelį kiekį atminties, taip pat patogias vartotojo sąsajos galimybes. Tačiau šio sprendimo trūkumas yra tas, kad emulatorius bus tos pačios architektūros kaip ir paprastas kompiuteris, kuri daugeliu atvejų skirsis nuo mobiliame įrenginyje esančios. Taip pat resursų padidinimas neatspindės realios situacijos mobiliuose įrenginiuose ir neleis pilnai ištestuoti (pavyzdžiui atminties trūkumo situacijos).

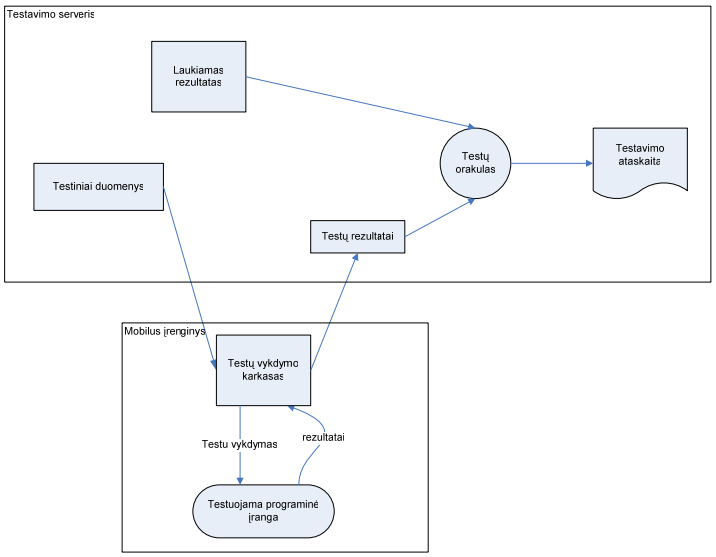
Taip pat siūlomas variantas naudoti automatinį perteklinių testų pašalinimą[5]. Naudojant automatinius testų generavimo įrankius sugeneruojami dideli kiekiai testų, iš kurių daugelis yra pertekliniai, tad siūloma juos aptikt ir pašalinti. Šis sprendimas būtų gana efektyvus, bes jis visiškai nenaudingas jei testų kiekis yra didelis ir perteklinių nėra.

4.3 Nuotolinis modulių testavimo karkasas

Mes siūlome naudoti nuotolinį modulių testavimo karkasą testuojant programinę įrangą mobiliems įrenginiams. Jis skirtas išspręsti problemoms kylančioms dėl ribotų atminties resursų ir ribotos vartotojo sąsajos mobiliuose įrenginiuose.

Šio nuotolinio modulių testavimo karkaso idėja yra ta, kad atskiriame daug vietos užimančius testus ir juos patalpiname ne mobiliame įrenginyje, o išorėje. Išore laikomas kitas kompiuteris turintis pakankamai resursų saugoti testavimo duomenis ir sujungtas su mobiliu įrenginiu kokiu nors tinklo ryšiu.

Siūloma nuotolinio modulių testavimo karkaso sandara pateikta sekančiame paveikslėlyje.



1 pav. Nuotolinis vietų testavimo karkasas.

Mobiliame įrenginyje būti paliktas tik testų vykdomo karkasas, kurio paskirtis būtų gauti reikiamą testą, jį įvykdyti ir gražinti rezultatus. Paprasčiausiu atveju tai būti tiesiog funkcija, kuri per parametrus gauna testą, testuojamo objekto vardą, metodą, paduoda testus testuojamo objekto metodui, pasiima vykdymo rezultatus ir gražina į testavimo serverį. Visi testavimo duomenys, laukiami rezultatai, rezultatų formavimas, atvaizdavimas būtų atliekami išorėje tuo nenaudojant mobilaus įrenginio resursų.

Toks variantas reikalauti žymiau mažiau atminties resursų mobiliame įrenginyje.

Testavimo procesą vykdyti šiuo būdų:

1. Iš testinių duomenų sąrašo imamas vienas testinis atvejis (vienas testas, su laukiamu rezultatu).
2. Jis siunčiamas į testų vykdymo karkasą, mobiliame įrenginyje.
3. Mobiliame įrenginyje įvykdomas testas.
4. Gražinamas testo vykdymo rezultatas iš mobilaus įrenginio
5. Sulyginamas gaunas rezultatas su laukiamu rezultatu.
6. Imamas sekantis testinis atvejis.
7. Kartojamas 2-6 žingsniai, kol nesibaigė testiniai atvejai.
8. Formuojama testavimo ataskaita.

Testinio atvejo atvaizdavimas gali būti realizuotas kelias būdais:

- Perduodant objekto vardą, testuojamo metodo vardą ir parametrus metodui.
  Šiuo būdų Testų vykdymo karkasas mobiliame įrenginyje turėtu mokėti sukurti testuojamos programinės įrangos nurodytą objektą ir iškviesti nurodytą metodą. Šiai galimybei realizuoti būtu reikalinga galimybė matyti programinės įrangos meta duomenis ir aplinka turėtu teikti atspindžių (Refelctions) [1] galimybes.
- Generuojat testavimo kodą vienam testiniam atvejui serverio pusėję ir perduodant testų vykdymo karkasui mobiliame įrenginyje.
  Šiuo būdų testavimo karkaso dalis serverio pusėje turėtu sukompiliuoti programos vieneto testinio atvejo kodą, kuris turėtų būti pritaikytas prie konkrečios procesoriaus architektūros esančios mobiliame įrenginyje. Paruošus kompiliuotą, kodą jis turėtų būti persiunčiamas į mobilų įrenginį ir vykdomas.

Nuotolinis modulių testavimo būdas padeda išspręsti problemas susijusias su resursų apribojimais, nes testai saugomi serverio pusėje, rezultatų atvaizdavimas taip pat atliekamas serverio pusėje. Tačiau vienas iš šio būdo minusų yra, kad platforma turi palaikyti atspindžius (Reflections)) [1], ir tinko gedimai gali įtakoti testavimo procesą. Tinklo gedimams aptikti reiktų realizuoti tam tikras priemones, kurios aptiktų tinklo gedimą ir perleistų testavimo atvejį iš naujo.

5 Realizacija

Nuotolinis modulių testavimas buvo realizuotas ir išbandytas kuriant mobilaus dėstytojo programinę įrangą. Programinė įranga buvo realizuota PocketPC tipo delniniame kompiuteryje, naudojant Microsoft .NET technologiją.

Siekianti sėkmingai ištestuoti šia programinę įrangą buvo sukurtas nuotolinis modulių testavimo karkasas. Jis buvo realizuotas naudojant Microsoft .NET technologiją. Jis susideda iš kelių dalių:

- Testų vykdytojas mobiliame įrenginyje
  Realizuotas kaip XML web servisas.
- Testų vykdytojas valdančiajame įrenginyje (serveryje)
  Realizuota kaip Microsoft .NET programa staliniame kompiuteryje.

Testuojama programinė įranga buvo realizuota Microsoft .NET platformoje, kuri leido naudoti atspindžius (Reflections) [1]. Todėl buvo pasirinktas pirmas realizavimo būdas.

Mobilus įrenginys buvo prijunktas prie tinklo naudojant bevielį ryšį (WiFi). Testų sudarymui naudojama programinės įrangos meta data. Pagal ją išrenkamos programinės įrangos klasės, metodai ir lentelių pavidalu suvedami metodų parametrai ir laukiamos grąžinamos reikšmės. Vėliau tokie suvesti testiniai atvejai po viena siunčiami į mobilų įrenginį. Mobiliame įrenginyje sukuriamas nurodytas objektas, iškviečiamas nurodytas metodas ir perduodami nurodyti parametrai. Fiksuojama grąžinama reikšmė ir ji perduodama į testavimo serverį, kuriame, atliekami rezultatų palyginimas su laukiamas rezultatais ir formuojama testavimo ataskaita, kuri vėliau rodoma serveryje.

6 Išvados

- Nuotolinis modulių testavimas leis išspręsti testavimo problemas susijusias su mobilių įrenginių resursų apribojimais.
- Nuotolinis modulių testavimas leis testuojant programinė įranga mobiliame įrenginyje įvykdyti didesnį unit testų kiekį.
- Nuotolinis modulių testavimas suteiks patogias galimybės testavimo rezultatų stebėjimui.
- Testavimo nesėkme galia įtakoti tinklo gedimai, bet juos galima fiksuoti.

Literatūros sąrašas

[1] **M. Ancona, W. Cazzola**, Implementing the essence of reflection: a reflective run-time environment, *Symposium on Applied Computing archive. Proceedings of the 2004 ACM symposium on Applied computing*, 2004, 1503-1507 p.

[2] **B. George, L. Williams**, Software engineering: An initial investigation of test driven development in industry, *Proceedings of the 2003 ACM symposium on Applied computing*, 2003.

[3] **J. Noble, S. Marshall, S. Marshall, R. Biddle**. Less Extreme Programming. *Proceedings of the sixth conference on Australian computing education*, 2004, Volume 30.

[4] **I. Satoh**. A Testing Framework for Mobile Computing Software. *IEEE Transactions on Software Engineering*, 2003.

[5] **T. Xie, D. Marinov, D. Notkin**. Rostra: A Framework for Detecting Redundant Object-Oriented Unit Tests. *Automated Software Engineering, 19th International Conference on (ASE'04)*, 2004, 196-205p.

Remote Unit Testing for Mobile Devices

Software for mobile devices is becoming increasingly popular. The limited resources and limited user interface brings new issues in software development process for mobile devices. These limitations should be considered especially in design and testing phases of software development process.
This paper presents problems encountered in software testing process for mobile devices. Suggests the new approach for performing unit testing for mobile devices, which does not suffer for these limitations of mobile devices.

VARTOTOJO SĄSAJOS DELNINIAMS KOMPIUTERIAMS KŪRIMO PRINCIPAI

Šarūnas Packevičius, Andrej Ušaniov

Kauno Technologijos universitetas, Informatikos fakultetas, Studentų g. 50, LT - 3031 Kaunas

Kuriant programinę įrangą delniniams kompiuteriams svarbi dalis yra vartotojo sąsaja. Delninių kompiuterių ekranų dydžiai yra gana maži, tai pat vartotojo bendravimo su delniniu kompiuterių būdai yra gana riboti. Tai neleidžia taikyti tuos pačius principus vartotojo sąsajos kūrimui kaip ir staliniams kompiuteriams.

Pranešime pateikiami vartotojo sąsajos kūrimo principai delniniams kompiuteriams. Pavyzdys kaip buvo principai pritaikyti kuriant mobilaus dėstytojo programinę įrangą delniniam kompiuteriui.

1 Įžanga

Pasirodžius mobilioms technologijos, tarp jų ir delniniams kompiuteriams tai suteikia naujų galimybių kuriant programinę įrangą. Vartotojai naudodamiesi delniniais kompiuteriais gali pasiekti jiems reikiamą informaciją bet kuriuo metų. Tai gali duoti didele naudą verslui. Pavyzdžiui galimybės peržiūrėti turimų prekių sąrašus delniniuose kompiuteriuose, užsakymų pildymas ir pan. Delniniai kompiuteriai yra puikūs įrenginiai norimai informacijai išvesti ekrane, taip pat leidžia įvesti reikiamą informacija greitai ir esant bet kurioje vietoje. Tačiau programinė įranga delniniuose kompiuteriuose turi ir trukumų. Dėl delniniu kompiuterių mažų dydžių iškyla problemų susijusių su duomenų atvaizdavimu juose, dėl įvedimo būdų nedidelio patogumų padidėja vartotojo duomenų įvedimo klaidų kiekis.

2 Delninio kompiuterio vartotojo sąsaja

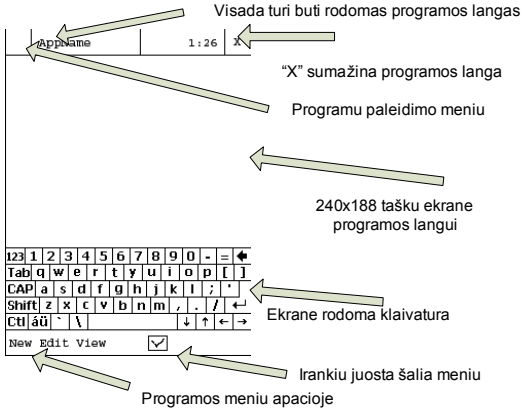
Mobilieji įtaisai leidžia susipažinti su įvairia medžiaga bet kuriuo metų (būnant darbe, lauke, pietaujant ir t.t.). Tačiau mažas ekranas neleidžia greitai peržvelgti pateikiamos informacijos (perskaitant tiesiog antraštes), o tenka skaityti visą tekstą po kelis sakinius (tiek jų telpa ekrane).

Kadangi delniniu kompiuterių ekrano plotas yra gana nedidelis kyla problemų kaip sutalpinti reikiamą informaciją juose. Ekranai dažnai būna ne didesni kaip 240x320 taškų skiriamosios gebos, taip pat dalį ekrano užima sistemos elementai, tokie kaip viršutinė programų juosta, apatinis programų meniu. Taip pat delniniuose kompiuteriuose būna naudojami riboto kiekio spalvų ekranai (16 bit, 12-bit, nespalvoti). Dar taip pat didėlę ekrano dalį atima iš programos, ekrane rodomi duomenų įvedimo priemonės, pavyzdžiui klaviatūra.

Delninių kompiuterių vartotojo sąsaja yra labai skirtinga nuo stalinių kompiuterių, pagrindinai skirtumai neskaitant mažo ekrano dydžio, yra tai, kad programos meniu yra dedamas ekrano apačioje, visai priešingai, taip pat meniu nėra „File“, „Save“ ar „Exit“ mygtukų/meniu punktų, programos turi automatiškai išsaugoti pakeitimus tik juos padarius, arba operacinei sistemai paprašius programai baigti darbą.

Paleistos programos langas delniniame kompiuteryje visada būna per visą ekraną. Dalį ekrano užima tik iššokantys pranešimų langai, o jei reikia įvesti kokią informaciją įvedimo dialogo langas turėtų būti per visą ekraną, tuo leidžiant lengviau įvesti duomenis.

Programos langas uždaromas paspaudus „X“ mygtuką kairėje viršuje. Bet šis mygtuko paspaudimas tik panaikina programos langas, jis greičiau atlieka programos lango sumažinimo funkciją, o pačios programos uždarymo nėra iš viso. Operacinė sistema automatiškai valdo atmintį, ir atsiradus atminties stygiui automatiškai uždaro rečiausiai naudojamą paleistą programą. O norint suaktyvinti paslėptą programą („X“ mygtuką paspaudus), reikia bandyti paleisti programą iš naujo, o iš to išplaukai, kad tos pačios programos negali būti paleistos kelios kopijos, paleidžiant programą ji turi pasitikrinti kad tik viena jos kopija paleista, o jei jau yra ji paleista tai tik suaktyvinti jos langą. Tipinis delninio kompiuterio programos langas pateiktas paveikslėlyje 1.



1pav. Delninio kompiuterio vartotojo sąsaja

Pelę atstoja pieštukas. Juo naudojantis nematomas ekrane žymeklis. Taip pat dešinės pelės mygtukas imituojamas funkcija pavadinta „Tap-and-held“, norint iškviesti meniu reikia paspausti ir laikyti pieštuką prispaudus prie ekrano, kol išlenda norimas meniu. Su šia funkcija vartotojas būna supažindinamas pirmą kartą įjungus delninį kompiuterį. Šia funkcija yra gana nepatogu naudotis jei tenka delniniu kompiuteriu naudotis viena ranką.

Pagrindinės problemos yra kaip išdėlioti įvairius įvedimo laukus mažame ekrane, kad tuo pačiu vartotojas galėtų juos lengvai rasti ir juose įvesti duomenis, ir taip pat nepakraunant įvedimo lango didelių kieku tankiau suspaustų įvedimo komponentų.

3 Duomenų įvedimas

Vartotojams tenka įvesti informaciją naudojantis delniniu kompiuteriu. Delninis kompiuteris turi daug skirtingų būdų įvesti informacijai, bet tuo pačiu kiekvienas iš jų yra ribotas. Pagrindinai įvedimo būdai būtų:

- Klaviatūra (rodoma ekrane, ar klavišai ant įrenginio)
  Įvedant duomenis klaviatūra, tenka naudotis delniniame kompiuteryje esančia klaviatūra, kur yra pakankamai maža (maži klavišai, klaviatūros pavyzdys pateiktas paveikslėlyje 1), tuo padidinant tikimybę atsirasti įvedimo klaidoms.
- Rašant ranka pieštuku ekrane
  Žmogui patogu būtų tekstą įvedinėti tiesiog jį užrašant ranką. Bet daugelio įrenginių procesoriai dar per lėti, kad galėtų atpažinti tekstą. Tekstas gali būti atpažįstamas neteisingai. Taip pat teksto įvedimas rašant ranka gali reikalauti naudoti specialią abėcėlę (tekstas užrašomas gestais).
- Diktuoti žodžiu
  Įvedami duomenys gali būti diktuojami žodžių, bet padiktuoto teksto atpažinimas reikalauja nemažų delninio kompiuterio skaičiavimo resursų, kurių nedaugelis dar delninių kompiuterių turi. Taip pat padiktuoto teksto atpažinimas gali prastai veikti triukšmingose aplinkose.

Įvedant duomenis pagrindinės problemos kyla dėl klaidų atsiradusiu įvedimo metu (paspaustas ne tas klavišas, įvestas ne tas simbolis, per daug reikia įvesti pasikartojančios informacijos ir pan., per daug apkrautos vartotojo sąsajos.)

4 Sąsajos projektavimas

Vartotojo sąsaja turi būti kuriama siekiant sumažinti įvedimo klaidų kiekį. Taip gerai suprojektuota vartotojo sąsaja yra labai svarbi, nes:

- Pagal vartotojo sąsaja dažniausiai sprendžiama apie programinę įranga, o ne jos funkcionalumą
- Prastai suprojektuota vartotojo sąsaja gali įtakoti vartotojų daromų klaidų kiekio padidėjimą
- Programinės įrangos atmetimo/nenaudojimo priežastis dažnai būna prasta vartotojo sąsaja.

Dėl mažų delninių kompiuterių ekranų, vartojo sąsaja gali gautis labai sugrūsta stengiantis sukišti kuo daugiau ir kuo arčiau šalia vienas kito įvedimo laukų, kas neduotų reikiamo efekto, o tik padidintu vartotojo daromų klaidų kiekį įvedant duomenis. Tai pat leidimas vartotojui įrašyti pačiam duomenis, gali būti klaidos šaltinis. Tad, kad išvengt įvedimo problemų reikti PĮ kurti:

- Kur galima naudoti pasirinkimo laukelis
- Minimizuoti tekstų įvedimo atvejus
- Išskaidyti įvedamos informacijos laukus dalimis.

5 Pasirinkimo laukelių naudojimas

Galimos reikšmės pasirenkamos iš comboBox. Tai leidžia išvengti negalimų reikšmių įvedimą. Taip pat nereikia naudotis maža ir sąlyginai nepatogia klaviatūrą. Duomenis galim įvesti naudojantis pieštuku ir sąraše pasirenkant norimą duomenų variantą. Taip pat padarius pasirinkimo dėžutė pakankamai didelę galima duomenis įvesti nesinaudojant pieštuku, o tiesiog pasirenkat ir paspaudžiant norimą reikšmę ranka.

Galimos reikšmės gaunamais iš įvedamų duomenų specifikacijos, kurios būna surinktos reikalavimų išgavimo proceso metu.

6 Įvedimo atvejų minimizavimas

Rodomos pradinės reikšmės, arba statistiškai nuspėjamos galimos naujos reikšmės (pavyzdžiui pildant naują užsakymą automatiškai suformuojamas sekantis užsakymo numeris). Tai taip pat galiam išgauti iš reikalavimų įvedamiems duomenims specifikacijos.

7 Išskaidyti įvedamos informacijos laukus dalimis

Informacija išskaidyti dalimis galiama naudoti:

- Kortelės (Tabs)
- Vedlys (Wizzard)

8 Kortelės (Tabs)

Įvedamai informacijai išskaidyti per keltą langų galima naudoti tab`s, jie turi būti rodomo ekrano apačioje. Tuo gauname nepakrautą įvedimo langą. Žinoma reikia įvedamą informaciją sugrupuoti pagal prasmę iš išskirstyti į tab`o puslapius. Naudojant tab`s dažnai jų būna daug ir jie netelpa ekrane, šiai problemai spęsti galima rodyti vietoj užrašų tik tab`u ikonas, o puslapiui tapus aktyviam užrašyti ir pavadinimą. Šis būdas labiau tinka kai nereikalinga kokia nors laukų įvedimo nuosekli tvarka.

9 Vedlys (Wizzard)

Analogiškai kaip ir tab`s atveju įvedimo laukus galima sugrupuoti ir pateikti wizzard įvedimo dialogo lange. Ekrane rodomi įvedimo laukai ir apačioje mygtukai pirmyn, atgal, nutraukti įvedimą. Toks įvedimo būdas labiau tinka jei informaciją reikia įvesti kokia nors nuoseklia tvarka.

10 Darbas be pieštuko

Su pieštuku galiam tiksliai pataikyti į norimą ekrano vietą. Tuo remiantis galima sukurti vartotojo sąsają su daug smulkių elementų ir sukišti vos ne viską į vieną formą. Tačiau taip daryti yra nerekomenduojama, nes tai tik suklaidins vartotoją ir vis tiek bus prarastas efektas. Kartais vartotojams tenka greitai įvesti informaciją ar iškviesti reikiamas funkcijas (pvz. kokias nors ataskaitas) ir vartotojas neturi laiko kada naudotis pieštuku, tad programinė įranga turėtų būti pritaikyta ir darbui kai funkcijos pasirenkamas ar įvedama ekraną spaudant pirštais. Kad tai apsiekti reikia:

- Vaizduoti didesnius grafinius elementus
- Palikti daugiau laisvos vietos šalia grafinio elemento ir išplėsti jo paspaudimui jaurią sritį.

Tai leis vartotojui lengviau pataikyti š norimą ekrano vietą ir iškviesti reikiamą funkciją.

11 Šriftų naudojimas

Siūloma naudoti tik keletą to pačio šrifto dydžių, vienoje formoje, norint ką nors pabrėžti naudoti pastorinta šriftą. Taip pat rekomenduojama naudoti operacinės sistemos teikiamus šriftus.

12 Spalvų naudojimas

Kadangi jau daugelis delninių kompiuterių turi jau spalvotus ekranus, tai rekomenduojama naudoti spalvotus vaizdus. Taip pat siūloma ekrane naudoti per daug slavų, gerai būtu tik kokios 2-3 spalvos iš viso. Nes didesnis spalvų naudojimas greičiau supainioja vartotoją negu jam padeda. Tai pat siūloma laikytis operacinės sistemos spalvų tipo, kad išlaikyti programinės įrangos sąsajos vientisumą visos programinės įrangos kompiuteryje atžvilgiu.

13 Vartotoją sąsaja paįvairinti grafika

Kad vartotojo sąsaja nebūtu tokia nubodi reikia pridėti kažkiek grafikos. Pavyzdžiui ikonų. Tai taip pat padėtų vartotojams greičiau įsisavinti vartoją sąsają, prisiminti kur koks langas. Žinoma neriekia dėti jos per daug. Siūloma viename lange tik koks vienas nedidelis paveikslėlis (ikona).

14 Duomenų laukų įvedimo būdo pasirinkimas

Jei reik pasirinkti iš trijų ar mažiau elementų sąrašo, rekomenduojama rodyti radio mygtukus. Jei pasirenkamų variantų nereikia iškarto matyti ekrane galima naudoti comboBox. Jei pasirenkama informacija netelpa ekrane galiama naudoti listbox, taip pat būtų gerai kad įvedus pirmas raides sąraše butų nušokama prie reikiamo lauko.

15 Patvirtinimo langai

Ištrinant įrašus arba užrašant ant viršaus ką nors, reikia rodyti paklausimą ar vartotojas nori atlikti tą veiksmą. Tai labai sumažins skambučių į techninės pagalbos skyrių su klausimų „Kaip atitrinti duomenis?“.

16 Numatyti galimybės programinę įrangą vaizduoti įvairiomis kalbomis

Programinė įranga vaizduojanti informaciją įvairiomis kalbomis gali pritraukti didesnį kiekį vartotojų.

17 Operacinės sistemos suderinamumas ir jos keliami reikalavimai

Taip pat kuriant programinė įranga pritaikyta Pocket PC arba Palm OS siūloma laikytis operacinei sistemai būdingų vartotojos sąsajos projektavimo rekomendacijų. Remiantis tomis rekomendacijomis išlaikome vieningą sąsaja visų programų delniniame kompiuteryje, taip pat sumažiname laiko tarpą, per kurį vartotojas gali išmokti dirbti su programine įranga.

Microsft PocketPC operacinėms sistemos siūlomos rekomendacijos pateikiamos kaip reikalavimai, kad programinė įranga būtų sertifikuota su Designed for Windows Mobile™ ženklų. Šios rekomendacijos siūlo.

Diegimo metu įdedamos programos nuorodos į Programų katalogą. Įtraukiamos nuorodos į My Pocket PC\Windows\Start Menu\Programs katalogą. Tame kataloge yra visos nuorodos į visas delniniame kompiuteryje esančias programas.

Programos pavadinimas turi būti rodomas navigacijos laukelyje. Navigacijos laukelyje visada rodomas programos pavadinimas, nepriklausomai koks programos langas atidarytas. Pavadinimas negali būti nukirptas. Turi tilpti į ekraną. Navigacijos laukelyje negalima nieko kito rodyti.

„Naujas“ ir „SIP“ mygtukai turi būti rodomi tik standartinėse vietose „Naujas“ mygtukas turi būti rodomas ekrano apačioje, kairėje pusėje. „SIP“ mygtukas turi būti rodomas ekrano apačioje, dešinėje pusėje. Taip pat SIP mygtukas turi būti rodomos tik tada, kai yra įmanomas duomenų įvedimas, kitais atvejais turi būti paslepiamas.

Meniu turi butų kairėje pusėje apačioje. Mygtukai turi būti kairėje apačioje į dešine nuo meniu. Programoje neturi būti „File“ meniu. Dažnai pasitaikantys meniu punktai turi būti išrikiuoti specialia tvarka. Jei programoje yra meniu punktai tai jei turėtų būti išdėstyti tokia kaire iš kairės į dešinę : „Edit“, „View“, „Insert“, „Format“, „Tools“. Jei programoje yra dažnai programose pasitaikantys mygtukai jei turėtų būti išdėstyti tokia tvarka iš kairės į dešinė: „“New“, „Open“, „Save“, „Print“

ToolTips ant visų įrankių juostos mygtukų. Įrankių juostoje mygtukai turi turėti tooltips. Kad vartotojas galėtų susipažinti greitai su mygtukais.

Nėra vartotojui pasirenkamų funkcijų programos uždarymui. Operacinė sistema pati valdo kada reikia uždaryti programą. Vartotojui leidžiama tik iškviesti lango paslėpimo funkciją.

Pranešimai rodomo standartinėmis priemonėmis Pranešimų langams rodyti naudojamos standartinės priemonės. Negalima paišyti savo pranešimo langų.

Kalendoriaus valdymas, kontaktų, užduočių valdymai nekuriami iš naujo, o programa turi naudoti operacinės sistemos teikiamas funkcijas

Programa turi reaguoti į įvedimo lango atsiradimą, paslėpimą. Reikiant įvesti informaciją į programos laukus, būna rodoma klaviatūra ekrano apačioje, kuri uždengia 80 taškų ekrano nuo apačios, tad programa turi užtikrinti, kad rodant klaviatūra nebus paslėpti įvedimo laukai.

Programa turi naudoti operacinėje sistemoje nustatytus parametrus: spalvas, skaičių, valiutų rodymo formatus ir pan.

Programai užsidarant neturi būti rodomo jokie pranešimai ar dialogo langai su užklausimais. Tai reikalinga todėl, kad operacinė sistema automatiškai valdo atmintį ir prireikus uždaro retai naudojamas programas. Programos būsenos išsaugojimas. Programa išeidama turi išsaugoti savo būseną, o paleidus iš naujo ją atstatyti.

Naudojami sisteminiai failų atidarymo, išsaugojimo dialogo langai.

18 Praktinis panaudojimas

Pateikti vartotojo sąsajos kūrimo principai delniniams kompiuteriams buvo pritaikyti kuriant mobilaus dėstytojo programinę įrangą. Programinė įranga leidžia dėstytojui naudojantis delniniu kompiuteriu žymėti studentų lankomumą, žymėti studentų darbų vertinimus, taip pat juos publikuoti juos internete, kad studentai galėtų peržiūrėti savo darbų įvertinimus. Taip pat programinė įranga leidžia sudaryti darbų, atsiskaitymų, paskaitų tvarkaraštį ir jį publikuoti studentams. Programinė įranga leidžia spausdinti įvairias ataskaitas iš delninio kompiuterio spausdintuvu.

Programinės įrangos vartotojo sąsaja buvo kuriame remiantis Designed for Windows Mobile™ rekomendacijomis. Daugelis įvedamos informacijos laukų nenaudoja įvedimo iš klaviatūros. Pagal reikalavimų specifikaciją ir įvedamų duomenų specifiką buvo nustatytos galimos įvedamos reikšmės ir jas galima pasirinkti iš sąrašų. Tuo minimizuojant įvedimo klaidas ir palengvinant patį įvedimo procesą.

Literatūros sąrašas

[1] **R. Bellamy, C. Swart, W. Kellogg, J. Richards, J. Brezin**. Designing an E-Grocery Application for a Palm Computer: Usability and Interface Issues. *IEEE Personal Communications*, 2001 August, 60 - 64.

[2] **D. Mitchel**. The Challenges of Pocket PC Development. *Pocket PC Magazine*, 2003 November.

[3] **B. Shadish**. Handheld User Interface Ten Commandments. *Mobile Coders*, 2003.

[4] **I. Sommerville**. User interface design. *Software Engineering Addison-Wesley Pub Co,* 2000.

User interface design principles for handheld computers

User interface is important part of software development for handheld computers. Handheld computers has small screens, also user computer interaction is quite complicated. For these reasons user interface design principles for desktop applications can not be used for handheld computers software.

This article describes some design principles for handheld computers user interface. Also article describes how these practices were used while developing software for mobile lecturer project.

MOBILIŲ PROGRAMŲ DIEGIMAS IR PALAIKYMAS

Šarūnas Packevičius

Kauno Technologijos universitetas, Informatikos fakultetas
Studentų g. 50, LT - 3031 Kaunas

Atsiradus mobilioms technologijoms kyla naujų galimybių atlikti norimus darbus greičiau ir patogiau, tačiau tai taip pat iškelia nauju problemų, tokiu kaip programinės įrangos diegimas ir palaikymas mobiliuose įranginiuose.

Šiame pranešime trumpai aptariamas dažniausiai pasirenkamas šių problemų sprendimo būdas, naudoti programinę įrangą realizuota žiniatinklio naršyklės lange, kuris turi nemažai trūkumų.

Pranešime pateikiamas kitas sprendimas, naudojantis automatiškai įsidiegiančią ir save atnaujinančią programinę įrangą mobiliuose įrenginiuose, kuris neturi programinei įrangai realizuotai tinklo naršyklės lange trūkumų ir suteikia naujų galimybių mobilioms programoms

1. Įžanga

Pasirodžius mobilioms technologijos, tarp jų ir delniniams kompiuteriams tai suteikia naujų galimybių kuriant programinę įrangą. Žmonės naudodamiesi delniniais kompiuteriais gali pasiekti jiems reikiamą informaciją bet kuriuo metų. Tai gali duoti didele naudą verslui. Pavyzdžiui galimybės peržiūrėti turimų prekių sąrašus delniniuose kompiuteriuose, užsakymų pildymas ir pan.

Kuriant programinę įrangą delniniams kompiuteriams iškyla diegimo ir tolesnio palaikymo problemos:

- Delninių kompiuterių yra daug ir skirtingų (skirtingos operacinės sistemos, tokios kaip Microsoft Pocket PC, Palm OS, skirtingos procesorių architektūros ARM, MIPS, SH3)
- Delniniai kompiuteriai būna pas vartotojus, o ne ofisuose, ir IT personalas negali lengvai ir paprastai atnaujinti programinės įrangos.

Dėl delninių kompiuterių įvairovės tenka kurti daug tos pačios programinės įrangos versijų pritaikytų skirtingoms operacinės sistemoms ir delninių kompiuterių procesoriams. O diegiant programinę įranga reikia pasitelkti organizacines priemones ir suorganizuoti susitikimus su kiekvienu vartotoju, kad įdiegti programinė įrangą į jo delninį kompiuterį. Vienkartinis diegimas nėra blogai, bet situacija pablogėja kai reikia atnaujinti programinę įrangą (pataisius klaidas, įtraukus naujų funkcijų į programinę įrangą), tektų vėl organizuoti susitikimus su kiekvienu vartotojų kaip ir diegimo metu.

2. Galimi sprendimai

Diegimo ir palaikymo problemas išspręsti galima pasirenkant programinės įrangos architektūrą ir realizuojant norimą programinę įrangą delniniams kompiuteriams vienu iš galimų būdų:

- Programinė įranga vykdoma žiniatinklio naršyklės lange (WEB programa).
- Save atsinaujinanti ir automatiškai įsidiegianti programinė įranga.

3. Programinė įranga vykdoma žiniatinklio naršyklės lange (WEB programa)

Šios problemos dažniausiai išsprendžiamos kuriant programinė įrangą, kuri yra vykdoma žiniatinklio naršyklės lange. Visa programinė įranga būna įdiegta pagrindiniame žiniatinklio serveryje, o vartotojams duodama tik tinklo adresas kuriuo nuėjus reikia užsikrauti tinklo puslapį ir naudotis. Realizavus programinę įrangą kaip programą vykdomą naršyklės lange teikia šiuos privalumus:

- Nereikalingas programinės įrangos diegimas vartotojo delniniame kompiuteryje.
- Diegiama tik viename kompiuteryje
- Atnaujinama, taisoma programinė įranga tik viename kompiuteryje

Tai žymiai sumažina diegimo ir palaikymo kaštus.

Tačiau realizavus programinę įrangą kaip vykdoma tinklo naršyklės lange susiduriame su kitomis problemomis:

- Reikalingas nuolatinis delninių kompiuterių prisijungimas prie tinklo

Visas programinės įrangos vykdymas vyksta pagrindiniame serveryje, o pas vartotoja atliekamas tik atvaizdavimas, tad reikalingas nuolatinis ryšys su serveriu. Be to tik keletas delninių kompiuterių modelių turi

galimybes būti nuolatos prisijungus prie tinko. Arba galimybė būti nuolat prisijungus yra gana brangi (naudojant GSM ryšį).

- Apkraunamas tinkas
  Kiekviena operacija programoje reikalauja duomenų siuntimo pirmyn ir atgal.
- Ribotos vartotojo sąsajos kūrimo galimybės
  Vartotojo sąsaja apribota tik tiek kiek leidžia HTML kalba ją aprašyti, taip pat dėl delninių kompiuterių įvairovės ir skirtingos HTML kalbos atvaizdavimo priemonių
- Apkraunamas pagrindinis serveris
  Visi skaičiavimai atliekami pagrindiniame kompiuteryje. Didėjant vartotojų kiekiui reikia vis daugiau kompiuterio resursų juos aptarnauti.
- Neišnaudojamos delninio kompiuterio procesoriaus galimybės
  Kadangi visi skaičiavimai atliekami pagrindiniame serveryje, neišnaudojami delniniuose kompiuteriuose esantys spartūs procesoriai (pvz.: 400 MHz)
- Nevienodos naršyklių galimybės delniniuose kompiuteriuose [1]
  Skirtingi delniniai kompiuteriai turi skirtingas HTML kalbos vaizdavimo priemones, taip pat įvairių tik jiem būdingų išplėtimų leidžiančiu kurti turtingesnę vartotojo sąsają)
- Nežinomybė klaidos atvejų

Tarkim serveris „nusmigo“ ir klientui rodomas neaiškus klaidos kodas (pvz. 403) ekrane ir būna neaišku ar buvo įvykdyta vartotojo komanda ar ne.

Programinė įranga realizuota kaip vykdoma žiniatinklio naršyklės lange netinka delniniams kompiuteriams dėl ribotos vartotojo sąsajos galimybių kūrimo (maži ekranai delniniuose kompiuteriuose) ir dėl poreikio turėti visada nuolatinį ryšį su tinklų (ne visi delniniai kompiuteriai turi bevielio tinklo galimybes arba tai per daug brangu).

4. Save atsinaujinanti ir automatiškai įsidiegianti programinė įranga

Kitas būdas būtų kurti save atsinaujinančią ir automatiškai įsidiegiančią programinę įrangą. Pasirinkus šį būda išsprendžiamos programinės įrangos realizuotas žiniatinklio naršyklės lange trūkumus ir taip pat užtikrinant lengvą programinės įrangos diegimą ir palaikymą.

Programinė įranga būtų diegiama ir vykdoma vartotojo delniniame kompiuteryje[2]. Įvairių operacinių sistemų ir procesorių architektūrų palaikymas būtų užtikrintas naudojant virtualią mašiną (pavyzdžiui Sun Java VM arba Microsoft .NET). Programinės įrangos atnaujinimas būtų atliekamas automatiškai: paleidus programą arba atsiradus ryšiui su tinklu (vartotojas prijungia delninį kompiuterį prie stalinio kompiuterio).

Programos įdiegimas vyktų panašiu būdų kaip ir programų vykdomų žiniatinklio naršyklės lange:

- Programos vykdomi ir kiti reikalingi failai būtų patalpinami reikiamame žiniatinklio serveryje.
- Sugeneruojamas pradinis puslapis su nuoroda į instaliacijos failą.
- Vartotojui perduodamas adresas.
- Vartotojas nuėjės nurodytų adresu įvykdo padėtą instaliacinį failą.
- Instaliacinis failas įdiegia virtualią mašiną, nukopijuojami programinės įrangos failai, įtraukiama nuoroda į delninių kompiuterio paleidimo menių.
- Paleidžiama programinė įranga

Programinės įrangos atnaujinimas vyktų paleidžiant programą ir esant ryšiui su tinklu, arba kai delninis kompiuteris susijungia su tinklu kokiu nors laiko momentu (prijungus delninį kompiuterį prie stalinio kompiuterio, sinchronizuoti kontaktus ir panašia informaciją būtų sinchronizuojama ir programinė įranga). Programos atnaujinimo žingsniai būtų tokie:

- Pataisius klaidas ar įdėjus naujų funkcijų, programinė įranga patalpinama tuo pačiu web adresu.
- Vartotojas eilinį karta bando pasinaudoti programine įranga delniniame kompiuteryje.
- Programinė įranga patikrina savo versiją ir serveryje esančią versiją.
- Radus naujesnė programinės įrangos versija įdiegiama nauja programinės įrangos versija iš web serverio.
- Paleidžiama programinė įranga tolesniam darbui.

Programinė įranga būtų paleidžiama kaip ir bet kokia kita programinė įranga esanti delniniame kompiuteryje, paleidžiant ją per programų meniu. Taip pat programą būtų galima paleisti ir nuėjus tinklo adresu iš kurio ji buvo įdiegta. Toks paleidimas vyktų taip:

- Vartotojas nueina tuo pačiu nurodytų adresu ir paleidžia instaliacinį failą.
- Paleistas failas patikrina ar yra įdiegta programinė įranga, jei jos nėra vykdomas diegimas.

- Radus įdiegtą programinę įrangą jį paleidžiama.

Lieka viena neišspręsta problema: kaip vartotojui įdiegti reikiamą virtualią mašiną. Tai galima išspręsti taip:

- Vartotojui kreipiantis į web serverį nustatoma jo operacinė sistema ir procesoriaus architektūra.
- Paduodama atitinkama nuoroda į reikiamą virtualią mašiną diegimui.

Delninio kompiuterio operacinės sistemos ir procesoriaus architektūros nustatymui galima pasinaudoti tuo, kad žiniatinklio naršyklės kreipiamosios perduoda savo versiją, operacinės sistemos versija, pavadinimą ir procesoriaus tipą. Pasinaudojus šias įrodymais ir sukaupta duomenų baze kuri atitinka virtuali mašina -> jai būdingi požymiai, nustatome reikiamą virtualios mašinos tipą. Pavyzdžiui duomenų bazė atrodytų taip kaip pateikta lentelėje 1.

Lentelė 1. Virtualios mašinos versijos ir įrodymų lentelė

Virtuali Mašinos diegimo failas	Operacinė sistema	Procesorius
VM.ARM.CAB	Pocket PC 2003	-
VM.ARM.CAB	Pocket PC 2002	ARM
VM.MIPS.CAB	Pocket PC 2002	MIPS
...	...	...

5. Išvados

Realizuojant programinę įrangą delniniams kompiuteriams kaip save atsinaujinančią ir automatiškai įsidiegiančią, turime visus žiniatinklyje realizuotos programinės įrangos varianto privalumus, ir priedo naujų galimybių:

- Turtinga vartotojo sąsaja
  Kuriama vartotojo sąsaja išnaudoja dauguma operacinės sistemos leidžiamų galimybių.
- Greitas programos veikimas
  Išnaudojami delniniame kompiuteryje esantys resursai, nenaudojamas kreipimasis į tinklą.
- Nebūtinas nuolatinis prisijungimas prie tinko
  Duomenys yra sinchronizuojami iš įrenginio su pagrindine duomenų baze tik tam tikrais momentais (pavyzdžiui: prijungus delninį kompiuterį prie tinklo, ar prie stalinio kompiuterio)

Programinės įrangos vykdomos žiniatinklio karšyklėje ir automatiškai įsidiegiančios ir save atsinaujinančios programinės įrangos realizavimo būdų palyginimas pateiktas lentelėje 2.

Lentelė 2. Būdų palyginimas

Būdas / Kriterijus	WEB programa	Automatiškai įsidiegianti ir save atsinaujinanti programinė įranga
Nuolatinis prisijungimas prie tinklo	Reikia	Nereikia
Delninių kompiuterių resursų išnaudojimas	Neišnaudojami	Išnaudojami
Vartotojos sąsajos galimybės	Minimalios	Išnaudojamos dauguma įrenginio siūlomų galimybių
Darbas be tinklo	Ne	Taip
Klaidų valdymas	Beveik nekontroliuojamas	Programinės įrangos kontroliuojamas
Lengvas diegimas	Taip	Taip
Lengvas palaikymas	Taip	Taip

Literatūros sąrašas

[1] **T. Goh, Kinshuk**. A discussion on mobile agent based mobile web-based ITS. *International Workshop on Wireless and Mobile Technologies in Education (WMTE`02)*, 2002, 137 – 138.

[2] **D. Mackenzie.** Introducing Client Application Deployment with "ClickOnce", *Essential Click Once™: A First Look at Deploying Windows Forms Applications with Click Once™. Addison Wesley Professional,* 2004, 130 - 148.

[3] **J. Waycott**. An Evaluation of the Use of PDAs for Reading Course Materials, *IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE`02)*, 2002, 177 - 178.