VILNIAUS UNIVERSITETAS

Žilvinas Vaira

ASPEKTINIO PROJEKTAVIMO ŠABLONŲ TYRIMAS, TOBULINIMAS IR KŪRIMAS

Daktaro disertacijos santrauka
Technologijos mokslai, informatikos inžinerija (07 T)

Vilnius, 2012

Disertacija rengta 2007–2011 m. Vilniaus universiteto Matematikos ir informatikos institute.

Mokslinis vadovas

prof. dr. Albertas Čaplinskas (Vilniaus universitetas, technologijos mokslai, informatikos inžinerija – 07 T).

Disertacija ginama Vilniaus universiteto Matematikos ir informatikos instituto Informatikos inžinerijos mokslo krypties taryboje:

Pirmininkas

prof. habil. dr. Gintautas Dzemyda (Vilniaus universitetas, technologijos mokslai, informatikos inžinerija – 07 T)

Nariai:

prof. dr. Romas Baronas (Vilniaus universitetas, fiziniai mokslai, informatika – 09 P)

prof. habil. dr. Romualdas Baušys (Vilniaus Gedimino technikos universitetas, technologijos mokslai, informatikos inžinerija – 07 T)

dr. Virginijus Marcinkevičius (Vilniaus universitetas, technologijos mokslai, informatikos inžinerija – 07 T)

prof. habil. dr. Rimantas Šeinauskas (Kauno technologijos universitetas, technologijos mokslai, informatikos inžinerija – 07 T)

Oponentai:

prof. dr. Rimantas Butleris (Kauno technologijos universitetas, technologijos mokslai, informatikos inžinerija – 07 T)

prof. dr. Saulius Gudas (Vilniaus universitetas, technologijos mokslai, informatikos inžinerija – 07 T)

Disertacija bus ginama viešame Informatikos inžinerijos mokslo krypties tarybos posėdyje 2012 m. balandžio 6 d. 13:00 val. 203a.
Adresas: Akademijos g. 4, LT-08663, Vilnius, Lietuva

Disertacijos santrauka išsiuntinėta 2012 m. kovo 5 d.
Disertaciją galima peržiūrėti Vilniaus universiteto bibliotekoje

VILNIUS UNIVERSITY

Žilvinas Vaira

INVESTIGATION, IMPROVEMENT AND DEVELOPMENT OF ASPECT-ORIENTED DESIGN PATTERNS

Summary of Doctoral Dissertation
Technological sciences, informatics engineering (07 T)

Vilnius, 2012

Doctoral dissertation was prepared at Institute of Mathematics and Informatics of Vilnius University in 2007-2011.

Scientific Supervisor

Prof Dr Albertas Čaplinskas (Vilnius University, Technological Sciences, Informatics Engineering – 07 T).

The dissertation will be defended at the Council of Science of Informatics Engineering at the Institute of Mathematics and Informatics of Vilnius University:

Chairman

Prof Dr Habil Gintautas Dzemyda (Vilnius University, Technological Sciences, Informatics Engineering – 07 T)

Members:

Prof Dr Romas Baronas (Vilnius University, Physical Sciences, Informatics – 09 P)

Prof Dr Habil Romualdas Baušys (Vilnius Gediminas Technical University, Technological Sciences, Informatics Engineering – 07 T)

Dr Virginijus Marcinkevičius (Vilnius University, Technological Sciences, Informatics Engineering – 07 T)

Prof Dr Habil Rimantas Šeinauskas (Kaunas University of Technology, Technological Sciences, Informatics Engineering – 07 T)

Opponents:

Prof Dr Rimantas Butleris (Kaunas University of Technology, Technological Sciences, Informatics Engineering – 07 T)

Prof Dr Saulius Gudas (Vilnius University, Technological Sciences, Informatics Engineering – 07 T)

The dissertation will be defended at the public meeting of the Council of Science of Informatics Engineering in the auditorium number 203 at the Institute of Mathematics and Informatics of Vilnius University, at 1 p. m. on 6 April 2012.
Address: Akademijos g. 4, LT – 08663, Vilnius, Lithuania.

The summary of the doctoral dissertation was distributed on 5 March 2012.
A copy of the doctoral dissertation is available for review at the Library of the Vilnius University.

Įvadas

Mokslo problemos aktualumas

Programų sistemos yra dažnai keičiamos, nes jas reikia pritaikyti prie pasikeitusių reikalavimų ir dėl nuolat kintančių technologijų. Modulinė sistemos architektūra įgalina nepriklausomus dalykinius turinius realizuoti nepriklausomais ar nedaug vienas nuo kito priklausomais moduliais arba, kitaip tariant, sudaro projektuotojui prielaidas atlikti sistemoje turinių atskyrimą. Kadangi tokius modulius galima keisti nepriklausomai vienas nuo kito, šitaip yra palengvinami visos sistemos atnaujinimai (Bertrand Meyer, 1997). Objektinėje programų sistemų inžinerijos paradigmoje sukurta nemažai veiksmingų metodų ir technologijų, skirtų modulinėms programų sistemoms projektuoti ir įgyvendinti. Tačiau objektinės paradigmos priemonėmis galima atskirti ne visus turinius. Kai kurie dalykiniai turiniai yra susipynę tarpusavyje ir objektinės architektūros sistemose jų negalima realizuoti savarankiškais moduliais. Šiai problemai spręsti buvo pasiūlyta nauja programų sistemų inžinerijos paradigma – aspektinė paradigma (Kiczales, et al., 1997). Greta kitų programų sistemų inžinerijos problemų, pavyzdžiui, keleto subjektyvių požiūrių inkapsuliavimo atskiruose objektuose problemos (Harrison, Ossher, 1993), aspektinėje paradigmoje išspręsta ir susipynusių turinių atskyrimo problema. Tačiau sistemų projektavimo technologija joje kol kas dar nėra pakankamai brandi. Be kita ko, nėra žinoma, kuriuos Gamma ir jo kolegų (Gamma et el., 1994) pasiūlytus objektinio projektavimo šablonus, vadinamuosius *GoF šablonus*, galima panaudoti aspektinėje paradigmoje ir kaip juos transformuoti, keliant iš vienos paradigmos į kitą. *GoF* yra angliškų žodžių *Gang of Four* santrumpa. Šitaip anglų kalba yra vadinami minėto darbo (Gamma et el., 1994) autoriai. Kadangi tokių projektavimo šablonų yra 23, tolimesniame tekste jie yra vadinami *GoF23* arba tiesiog *GoF šablonais*.

GoF šablonai buvo sukurti analizuojant ir apibendrinant ilgametę objektinių programų sistemų projektavimo patirtį. Iki tol projektavimo šablonai buvo netyrinėti, nebuvo netgi ir paties termino „projektavimo šablonas“. GoF šablonų įvedimas į objektinių programų sistemų projektavimo priemonių arsenalą padarė ir toliau tebedaro didelį poveikį tiek objektinio projektavimo teorijai bei praktikai, tiek ir visai programų sistemų inžinerijos sričiai. Vis dėlto tiek GoF23, tiek ir kiti objektiniai šablonai pakankamai išnagrinėti tik objektinės paradigmos kontekste, o jų taikymas kitose programų sistemų inžinerijos paradigmose yra beveik netyrinėtas. Nors keliuose darbuose (Hannemann, Kiczales, 2002; Hachani, Bardou, 2002; Hirschfeld et al., 2003) nagrinėjama, kaip GoF23 šablonų realizacijas perrašyti aspektinėmis programavimo kalbomis, nė viename iš jų nebuvo sistemiškai nagrinėti objektinių projektavimo šablonų transformavimo į analogiškus projektavimo šablonus kitose programų sistemų inžinerijos paradigmose klausimai. Tai reiškia, kad vis dar nėra žinoma, kurie projektavimo šablonai skirti nuo konkrečios paradigmos nepriklausomoms projektavimo problemoms spręsti, o kuriuos galima panaudoti tik sprendžiant konkrečios paradigmos specifines projektavimo problemas. Ypač svarbu yra atsakyti į šį klausimą bent jau apimant objektinę ir aspektinę paradigmas. Jei kai kuriuos objektinius projektavimo šablonus galima pritaikyti aspektinės paradigmos specifikai, tai būtų nelogiška bandyti juos kurti šioje paradigmoje iš naujo, nes toks procesas būtų ilgas ir brangiai kainuojantis. Akivaizdu, jog aktualu pasinaudoti objektinėje paradigmoje sukaupta patirtimi ir pradėti reikia nuo išsamiai išnagrinėtų objektinės paradigmos projektavimo šablonų, visų pirma – nuo GoF23 šablonų.

Šioje disertacijoje nagrinėjami grynieji aspektinio projektavimo šablonai ir jų taikymas aspektiniams dalykiniams karkasams projektuoti. *Grynaisiais aspektinio projektavimo šablonais* (trumpiau – *grynaisiais aspektiniais šablonais*) yra vadinami tokie projektavimo šablonai, kuriuos realizuojant yra naudojamos tik aspektinių programavimo kalbų konstrukcijos, t. y. apsieinama be klasių, objektų bei kitų specifinių objektinių programavimo kalbų konstrukcijų. Pavyzdžiui, mišrieji projektavimo šablonai yra realizuojami panaudojant ir aspektus, ir objektus bei klases. Mišriuosiuose šablonuose aspektai dažniausiai vaidina antraeilį vaidmenį ir daugiausia yra naudojami turinių susipynimui šabloną realizuojančiame kode panaikinti.
Disertacijoje yra siekiama nustatyti, kurie GoF23 projektavimo šablonai sprendžia nuo objektinės paradigmos nepriklausomas projektavimo problemas, pasiūlyti, kaip tokius šablonus transformuoti į aspektinius projektavimo šablonus, ir ištirti aspektinių dalykinių karkasų, sukurtų panaudojant gautus projektavimo šablonus, savybes.

Darbo tikslas ir uždaviniai

Darbo tikslas yra išskirti tokią objektinių projektavimo šablonų grupę, kuriai priklausančius šablonus galima būtų transformuoti į grynuosius aspektinius šablonus, pateikti sistemingą tokios transformacijos procedūrą bei ištirti gautų grynųjų aspektinių šablonų taikymo aspektiniams dalykiniams karkasams projektuoti galimybes. Siekiant šio tikslo yra sprendžiami tokie uždaviniai:

- įvertinti dabartinę situaciją, palyginti esamus aspektinių projektavimo šablonų kūrimo būdus ir įvertinti tų būdų privalumus bei trūkumus;
- ištirti, kurios GoF23 projektavimo šablonais sprendžiamos projektavimo problemos yra aktualios ir aspektinėje paradigmoje, ir pasiūlyti, kaip tokius šablonus transformuoti į aspektinius projektavimo šablonus;
- ištirti tokių projektavimo šablonų taikymo galimybes projektuojant aspektinius dalykinius karkasus ir jų panaudojimo poveikį gaunamo kodo sudėtingumui, našumui ir kitoms vykdymo meto charakteristikoms.

Tyrimo klausimai ir hipotezės

Disertacijoje siekiama atsakyti į šiuos klausimus:

- Kokiu mastu jau yra išplėtota aspektinė programų sistemų inžinerijos paradigma?
- Kokie būdai gali būti naudojami aspektiniams projektavimo šablonams kurti, kokie yra šių būdų privalumai ir trūkumai? Ar aspektinė paradigma generuoja naujus, tik šiai paradigmai būdingus, projektavimo šablonus?
- Kokie gali būti ir kuo tarpusavyje skiriasi projektavimo šablonų, sprendžiančių nuo konkrečios paradigmos nepriklausančias projektavimo problemas ir projektavimo problemas, būdingas tik aspektinei ar objektinei paradigmai, realizavimo būdai?
- Ar įmanoma realizuoti bent dalį GoF23 projektavimo šablonų panaudojant tik aspektinių programavimo kalbų konstrukcijas? Jei įmanoma, tai kokius ir kaip? Ar tokia realizacija yra kuo nors geresnė nei objektinė? Kaip tai pamatuoti?
- Kuo, projektavimo šablonų atžvilgiu, aspektai skiriasi nuo klasių ir kaip šie skirtumai gali paveikti grynųjų aspektinių projektavimo šablonų struktūrą ir kitas savybes?
- Kokią naudą duoda grynųjų aspektinių projektavimo šablonų panaudojimas kuriant programų sistemas apskritai ir projektuojant aspektinius dalykinius karkasus konkrečiai?

- Kaip grynųjų aspektinių projektavimo šablonų naudojimas aspektiniuose dalykiniuose karkasuose paveikia turinių susipynimą, kodo realizavimo sudėtingumą ir sukurtos programos našumą?

Ieškant atsakymų į suformuluotus klausimus buvo iškeltos tokios hipotezės:

- kai kurios GoF23 šablonais sprendžiamos projektavimo problemos, bent jau objektinėje ir aspektinėje programų sistemų inžinerijos paradigmose, nepriklauso nuo konkrečios programų sistemų inžinerijos paradigmos;
- nors tiesioginis aspektų egzempliorių kūrimas aspektinėse programavimo kalbose nėra galimas, tų kalbų konstrukcijų pakanka realizuoti tuos GoF23 šablonus, kuriais sprendžiamos nuo konkrečios programų sistemų inžinerijos paradigmos nepriklausomos projektavimo problemos;
- projektuojamos programų sistemos yra efektyvesnės (angl. *efficiency of design*) tada, kai grynieji aspektiniai šablonai yra naudojami kartu su objektiniais šablonais;
- skaidriosios dėžės tipo aspektiniuose dalykiniuose karkasuose grynųjų aspektinio projektavimo šablonų panaudojimas duoda galimybę projektuoti naujo tipo užpildymo taškus (t. y. užpildymo taškus suprojektuotus, panaudojant abstrakčiuosius aspektus);
- aspektiniuose dalykiniuose karkasuose grynųjų aspektinių projektavimo šablonų panaudojimas sumažina tuose karkasuose turinių susipynimą;
- aspektinių dalykinių karkasų projektavimas, naudojant grynuosius aspektinio projektavimo šablonus, nedaro poveikio programų sistemų, sukurtų panaudojant tuos karkasus, našumui.

Tyrimų metodika

Disertacijoje atliekamas paieškomasis (angl. *exploratory*) tyrimas. Aspektinė programų sistemų inžinerijos paradigma dar yra gana nauja, o dauguma šioje srityje atliekamų tyrimų yra tik ankstyvosios stadijos tyrimai. Tai reiškia, kad siekiant suformuluoti tikslią problemos struktūrą ir norint geriau suvokti aplinką, kurioje yra keliama problema, reikia atlikti nuodugnią literatūros šaltinių analizę.

Kadangi šis tyrimas yra ne tik paieškomasis tyrimas, bet kartu ir inžinerinis nagrinėjamosios srities tyrimas, sprendžiant svarstomą problemą turi būti naudojami inžineriniai tyrimo metodai. Tokiame kontekste priimtiniausias yra tyrimo konstravimu (angl. *constructive research*) metodas.

Galiausiai, anot Cooper'io ir Schindler'io (Cooper, Schindler, 1998), paieškomasis tyrimas iš prigimties yra kokybinis tyrimas. Tokiame tyrime gautų rezultatų nėra galimybės vertinti kiekybiškai, atliekant matavimus, nes kokybiniai faktoriai iš principo negali būti išmatuojami. Be to, bet koks disertacijos tyrimas yra nedidelės apimties tyrimas, žvelgiant tiek iš finansinės, tiek iš laiko sąnaudų perspektyvos. Tai reiškia, kad tokiame tyrime yra per brangu ir praktiškai neįmanoma užtikrinti aukštą statistinį patikimumą ir aukštą kiekybinių matavimų statistinį reikšmingumą tais atvejais, kai matavimai gali būti atliekami. Todėl, nepaisant galimų paklaidų, atskiro atvejo analizė (angl. *case study*) yra vienintelis priimtinas tyrimo metodas, siekiant eksperimentiškai patvirtinti šio tyrimo rezultatus.

Atsižvelgiant į išdėstytus motyvus, tyrimo projektas yra išskaidomas į tris etapus: susijusių darbų koncepcinė analizė (Laurence, Margolis, 2003), objektinių GoF projektavimo šablonų transformavimo į aspektinius būdui sukurti taikomas tyrimas

konstravimu ir grynųjų aspektinio projektavimo šablonų taikymui projektuojant aspektinius dalykinius karkasus vertinti naudojamas empirinis tyrimas.
Koncepcinė analizė yra konceptų, terminų, kintamųjų, konstrukcijų, apibrėžimų, teiginių, hipotezių ir teorijų analizė. Koncepcinėje analizėje visi šie išvardyti dalykai yra nagrinėjami siekiant aiškumo ir rišlumo, kritiškai įvertinant jų loginius ryšius ir nustatant prielaidas bei implikacijas (Machado, Silva, 2007). Koncepcinės analizės tikslas yra padidinti nagrinėjamosios srities koncepcinį aiškumą. Pagrindinė koncepcinės analizės teikiama nauda yra tiriamosios srities būsenos įvertinimas tam, kad būtų galima strategiškai planuoti tolimesnį darbą (Penrod, Hupcey, 2004). Koncepcinė susijusių darbų analizė buvo atlikta, siekiant suformuluoti svarbias teorines konstrukcijas ir sudaryti teorinį pagrindą tolimesniems tyrimams, taip pat, kad būtų galima išvengti pakartotinių, kitų tyrėjų jau atliktų, tyrimų vykdymo. Pagrindinė koncepcinės analizės nagrinėjama sritis šioje disertacijoje apima objektinius bei aspektinius programų sistemų projektavimo šablonus. Koncepcinė analizė leidžia atsakyti į šiuos klausimus: kokiu mastu šiuo metu yra išplėtota aspektinė programų sistemų inžinerijos paradigma, kuo aspektai skiriasi nuo klasių, projektavimo šablonų atžvilgiu, ir kokia yra šių skirtumų įtaka grynųjų aspektinių šablonų struktūrai ir kitoms savybėms?
Svarbi koncepcinės analizės dalis yra konceptų kategorizavimas. Kategorizavimas yra naudojamas kaip pagrindas, apibrėžiant projektavimo šablonų, kurie gali būti transformuojami į grynuosius aspektinius šablonus, klasę.
Tyrimo konstravimu metodas yra tyrimo procedūra, generuojanti naujoviškas konstrukcijas, skirtas spręsti realaus pasaulio problemoms ir padaryti tam tikrą įnašą, taikant jas nagrinėjamos disciplinos teorijoje (Lukka, 2003; Crnkovic, 2010). Pagrindinė šio metodo notacija, nauja konstrukcija, yra abstrakti notacija, galinti turėti daug potencialių realizacijų. Modeliai, diagramos, metodai, algoritmai ir daugelis kitų artefaktų yra suprantami kaip konstrukcijos. Tai reiškia, kad jie yra išrasti arba sukurti, o ne atrasti. Matematiniai algoritmai ir naujos matematinės esybės yra teorinių konstrukcijų pavyzdžiai. Tyrimo konstravimu metodas remiasi įsitikinimu, kad nuodugniai analizuojant tai, kas veikia (arba neveikia) praktikoje, galima padaryti svarbų mokslinį įnašą į teoriją. Šioje disertacijoje tyrimo konstravimu metodas yra naudojamas, kuriant GoF projektavimo šablonų transformavimo į grynuosius aspektinius analogus, GoF_{AO} projektavimo šablonus, taisykles. Tikėtina, kad ne visi GoF23 projektavimo šablonai turi grynuosius aspektinius analogus, t. y. GoF_{AO} apima mažiau nei 23 šablonus. Analizuojant tiriamąją problemą buvo nustatyta, kad aspektai yra panašūs į vadinamąsias *klases singletonus*. Tai perša mintį, kad tokios klasės objektiniuose projektavimo šablonuose galėtų būti pakeistos aspektais. Tokio transformavimo galimybės turi būti išnagrinėtos kiekvienam šablonui atskirai, o rezultatai turi būti apibendrinti taip, kad galima būtų sukurti transformavimo taisykles, galiojančias visiems GoF projektavimo šablonams. Klasių ir aspektų panašumai taip pat rodo, kad objektiniai projektavimo šablonai, kurie negali būti realizuoti panaudojant tik klases singletonus, negali būti transformuojami į GoF_{AO} šablonus ir dėl šios priežasties sprendžia objektams būdingas projektavimo problemas. Dalis GoF projektavimo šablonų yra skirti su objektų kūrimu susijusioms projektavimo problemoms spręsti. Iš pirmo žvilgsnio tokių šablonų panaudojimas aspektinėje paradigmoje yra labai abejotinas ir turėtų būti ištirtas kaip atskiras atvejis, net jeigu pavyktų transformuoti juos į GoF_{AO} šablonus. Šio tipo projektavimo šablonai toliau bus žymimi GoF_{*AO}.

Tyrimo konstravimu metodas taip pat yra naudojamas ir darbinėms hipotezėms, laikinai suformuluotoms šiai disertacijai, testuoti. Vienas iš šio tyrimo metodo privalumų yra tai, kad jis suteikia galimybę ne tik testuoti ir analizuoti naujos konstrukcijos savybes, bet ir tyrinėti patį konstravimo procesą. Iš kitos pusės, tyrimas konstravimu, lygiagrečiai su kitais eksperimentinio tyrimo metodais, gali būti suprantamas kaip tam tikra atskiro atvejo analizės tyrimo metodo rūšis. Laikantis tradicinio požiūrio, atskiro atvejo analizė turi būti taikoma tik hipotezėms falsifikuoti. Atskiro atvejo analizė negali būti naudojama hipotezėms įrodyti ir turi būti priskiriama hipotetiniam dedukciniam aiškinamajam modeliui. Vis dėlto atskiro atvejo analizės atitiktis realaus pasaulio situacijoms ir jos detalumas rodo, kad toks požiūris yra tik iš dalies teisingas (Flyvbjerg, 2004). Remiantis šiuo argumentu ir tuo faktu, kad disertacijos tyrimas yra nedidelės apimties tyrimas, žvelgiant tiek iš finansinės, tiek iš laiko sąnaudų perspektyvos, atskiro atvejo analizės tyrimo metodas buvo pasirinktas kaip pagrindinis hipotezių testavimo metodas. Apskritai atskiro atvejo analizė yra empirinis tyrimo metodas, naudojamas, kai reikia ištirti tam tikrą fenomeną ir jo kontekstą (Runeson, Höst, 2009). Šios disertacijos tikslas yra ištirti grynųjų aspektinių šablonų taikymo įtaką aspektinių dalykinių (skaidriosios dėžės tipo) karkasų projektavimui.

Anot Ragin'o (Ragin, 1992), atskiro atvejo analizė gali būti sustiprinta strategiškai parenkant atvejus – kritinį arba tipinį. Kritinis atvejis dar gali būti suprantamas kaip ekstremalus atvejis, naudojamas testuoti hipotezėms kritinėse situacijose. Aspektinio dalykinio karkaso atvejis, kai karkasas yra projektuojamas, naudojant bent vieną GoF_{*AO} projektavimo šabloną, buvo pasirinktas kaip kritinis atvejis. Be to, buvo parinkti dar du tipiniai atvejai: jau sukurto objektinio dalykinio karkaso pertvarkymas į aspektinį dalykinį karkasą, naudojant GoF_{AO} šablonus ir naujo aspektinio dalykinio karkaso projektavimas, naudojant GoF_{AO} šablonus.

Pirmasis tipinis atvejis yra ribojamas esamo objektinio karkaso struktūros ir leidžia analizuoti pertvarkymo pasekmes, kai dalis objektinio karkaso yra pakeičiama atitinkamais, grynaisiais aspektiniais šablonais. Tik tos karkaso dalys, kurios buvo paveiktos turinių susipynimo, yra pertvarkomos. Antrasis tipinis atvejis neturi jokių išankstinių ribojimų ir leidžia pasirinkti bet kokią karkaso struktūrą, kuri yra labiausiai tinkama projektuoti aspektams. Iš viso disertacijoje yra nagrinėjami trys skirtingi atvejai.

Apskritai tiek kiekybiniai, tiek kokybiniai duomenų rinkimo metodai gali būti naudojami atskiro atvejo analizės rezultatams vertinti, nors paprastai atskiro atvejo analizės tyrimuose yra naudojama kokybinių duomenų analizė. Kokybiniai duomenys šioje disertacijoje susideda iš teksto, diagramų ir paveikslų, kurie yra analizuojami naudojant kategorizavimą ir rūšiavimą. Kiekybinio ir kokybinio metodų taikymas kartu leidžia pateikti svaresnius argumentus, vertinant hipotezes atskiro atvejo analizės tyrimuose (Runeson, Höst, 2009). Dėl šios priežasties abu duomenų rinkimo metodai yra taikomi disertacijoje.

Rezultatai

Disertacinio tyrimo rezultatai yra šie:

- Patvirtinta hipotezė, teigianti, kad kai kurios GoF23 šablonais sprendžiamos projektavimo problemos, bent jau objektinėje ir aspektinėje programų sistemų inžinerijos paradigmose, nepriklauso nuo konkrečios programų sistemų inžinerijos paradigmos.
- Nustatyta 20 objektinių GoF projektavimo šablonų, kurie sprendžia nuo konkrečios programų sistemų inžinerijos paradigmos nepriklausomas

projektavimo problemas ir gali būti transformuojami į grynuosius aspektinio projektavimo (GoF_{AO}) šablonus.
- Patvirtinta hipotezė, teigianti, kad nors tiesioginis aspektų egzempliorių kūrimas aspektinėse programavimo kalbose nėra galimas, tačiau tų kalbų konstrukcijų pakanka realizuoti tuos GoF23 šablonus, kuriais sprendžiamos nuo konkrečios programų sistemų inžinerijos paradigmos nepriklausomos projektavimo problemos, atsižvelgiant į tai, kad 5-ose iš jų pasireiškia ribotas panaudojamumas.
- Pasiūlytos taisyklės, nusakančios, kaip transformuoti 20 objektinių GoF projektavimo šablonų į GoF_{AO} projektavimo šablonus.
- Patvirtinta hipotezė, teigianti, kad projektuojamos programų sistemos yra efektyvesnės tada, kai grynieji aspektiniai šablonai yra naudojami kartu su objektiniais šablonais.
- Patvirtinta hipotezė, teigianti, kad skaidriosios dėžės tipo aspektiniuose dalykiniuose karkasuose grynųjų aspektinio projektavimo šablonų panaudojimas suteikia galimybę projektuoti naujo tipo užpildymo taškus (t. y. užpildymo taškus suprojektuotus, panaudojant abstrakčiuosius aspektus).
- Patvirtinta hipotezė, teigianti, kad aspektiniuose dalykiniuose karkasuose grynųjų aspektinių projektavimo šablonų panaudojimas sumažina tuose karkasuose turinių susipynimą.
- Paneigta hipotezė, teigianti, kad aspektinių dalykinių karkasų projektavimas, naudojant grynuosius aspektinio projektavimo šablonus, nedaro poveikio programų sistemų, sukurtų panaudojant tuos karkasus, našumui.

Praktinė vertė

Pagrindinis disertacijos praktinis įnašas yra 20 grynųjų aspektinio projektavimo šablonų, kurie buvo sukurti disertacijos tyrimo metu ir kurie gali būti naudojami projektuojant aspektinius dalykinius karkasus bei kitas aspektines dalykines programas. Disertacijoje taip pat yra parodyta, kad abstraktieji aspektai, pasitelkiant šiuos 20 grynųjų aspektinio projektavimo šablonų, gali būti projektuojami kaip užpildymo taškai aspektiniuose dalykiniuose karkasuose. Be to empirinio tyrimo metu yra apibrėžiami tokių karkasų projektavimo žingsniai.

Mokslinis naujumas

Ši disertacija yra vienas iš pirmųjų tyrimų, siekiančių ištirti grynuosius aspektinius projektavimo šablonus ir tokių šablonų taikymą, projektuojant aspektinius dalykinius karkasus. Ir nors pasitaiko pavienių bandymų (Arpaia, et al, 2008; Santos et al., 2007; Kulesza et al., 2006) projektuoti abstrakčiuosius, modifikuojamus aspektus karkasuose, nė vienas iš jų nenagrinėja grynųjų aspektinių šablonų panaudojimo projektuoti aspektams, kaip karkasų užpildymo taškams, ir nė vienas iš jų taip detaliai nenagrinėja pačių aspektinių karkasų projektavimo. Tai taip pat yra pirmasis darbas, kuris kelia klausimą apie projektavimo problemų, bendrų visoms arba bent jau kelioms programų sistemų inžinerijos paradigmoms, egzistavimą. Galiausiai, šioje disertacijoje taikomas atskiro atvejo analizės tyrimo metodas sustiprina nuostatą, kad programų sistemų inžinerijoje vykdomuose empiriniuose tyrimuose konstruktyvaus tyrimo ir atskiro atvejo analizės tyrimo metodai gali būti naudojami hipotezėms vertinti.

Aprobavimas ir publikacijos

Pagrindiniai disertacijos rezultatai buvo pristatyti ir aprobuoti šiose konferencijose:

- 3-oji tarptautinė konferencija „Pervasive Patterns and Applications“, PATTERNS 2011, 2011 m. rugsėjo 25–30 d., Roma, Italija.

- 15-oji mokslinė Lietuvos kompiuterininkų draugijos konferencija „Kompiuterininkų dienos-2011“, 2011 m. rugsėjo 22–24 d., Klaipėda, Lietuva.
- 50-oji Lietuvos matematikų draugijos konferencija, 2009 m. birželio 18–19 d., Vilnius, Lietuva.
- 12-oji Klaipėdos universiteto Gamtos ir matematikos mokslų fakulteto studentų mokslinės draugijos konferencija „Fundamentiniai tyrimai ir inovacijos mokslų sandūroje“, 2009, Klaipėda, Lietuva.

Darbo apimtis

Disertaciją sudaro įvadas, 5 skyriai, išvados, literatūros sąrašas, disertanto paskelbtų publikacijų sąrašas ir priedai. Bendra disertacijos apimtis – 161 puslapiai, 55 paveikslai, 5 lentelės, 7 pavyzdžiai. Kiekvienas skyrius yra pradedamas skyriaus santrauka ir baigiamas skyriaus išvadomis (išskyrus I skyrių).

Įvade aprašomas tyrimo kontekstas ir iššūkiai, pateikiama problemos formuluotė, aptariama tyrimo motyvacija, tikslai ir uždaviniai, apibrėžiami tyrimo klausimai ir hipotezės, aprašoma tyrimo metodika, rezultatai bei disertacijos praktinė vertė ir naujumas, galiausiai yra pateikiamas disertacijos rezultatų aprobavimas.

I skyriuje pateikiama projektavimo šablonų, aspektinės programų inžinerijos paradigmos ir programinių karkasų analitinė apžvalga.

II skyriuje aprašomi kritinės literatūros analizės rezultatai.

III skyriuje formuluojami ir aptariami pagrindiniai teoriniai tyrimo rezultatai. Šiame skyriuje patvirtinama hipotezė, teigianti, kad egzistuoja nuo paradigmos nepriklausomos problemos, bent jau objektinės ir aspektinės programų inžinerijos paradigmų kontekste, yra identifikuojamas 23 objektinių GoF projektavimo šablonų, sprendžiančių nuo konkrečios programų sistemų inžinerijos paradigmos nepriklausomas projektavimo problemas ir galinčių būti transformuojamais į grynuosius aspektinius (GoF_{AO}) šablonus, poaibis (20 GoF šablonų). Taip pat patvirtinama hipotezė, teigianti, kad aspektinių konstrukcijų pakanka realizuoti 20 GoF_{AO} projektavimo šablonų, atsižvelgiant į tai, kad 5-uose iš jų pasireiškia ribotas panaudojamumas, bei pateikiamos taisyklės, nusakančios, kaip reikia transformuoti 20 objektinių GoF projektavimo šablonų į aspektinius GoF_{AO} projektavimo šablonus.

IV skyriuje detaliai aprašomi empiriniai atskirų atvejų analizės tyrimai, nagrinėjantys transformuotų projektavimo šablonų taikymą aspektiniams dalykiniams karkasams projektuoti ir įvertinantys iškeltas hipotezes. Skyriuje yra patvirtinamos hipotezės, teigiančios: 1) kad GoF_{AO} projektavimo šablonų naudojimas (kartu su 23 GoF projektavimo šablonais) pagerina dalykinių karkasų modelių efektyvumą; 2) kad GoF_{AO} projektavimo šablonai leidžia projektuoti naujo tipo užpildymo taškus skaidriosios dėžės aspektiniuose dalykiniuose karkasuose (t. y. užpildymo taškams projektuoti yra naudojami abstraktieji aspektai); 3) kad GoF_{AO} projektavimo šablonai sumažina turinių susipynimą aspektiniuose dalykiniuose karkasuose; bei paneigta hipotezė, kad aspektinių dalykinių karkasų projektavimas, naudojant grynuosius aspektinio projektavimo šablonus, neturi įtakos programų, sukurtų, naudojant šiuos karkasus, našumui.

V skyriuje aptariami kai kurie svarbūs, bet tiesiogiai su disertacijoje atlikto tyrimo tikslu nesusiję ir dėl to joje nenagrinėti klausimai, kurie galėtų būti tolimesnių tyrimų objektu.

Išvadose pateikiamos darbo išvados.

Prieduose pateikti trumpas *AspectJ* programavimo kalbos aprašas (1 priedas), grynųjų aspektinio projektavimo šablonų modeliai ir jų aprašai (2 priedas) bei keletas papildomai detalizuotų diagramų (3-5 priedai).

1. Analitinė apžvalga

Šioje disertacijos dalyje yra pateikiamas glaustas disertacijoje naudojamų terminų ir susijusių konceptų aprašas. Analitinėje apžvalgoje yra aptariamos trys programų sistemų inžinerijos sritys: projektavimo šablonai, aspektinė programų sistemų inžinerijos paradigma ir programiniai karkasai.

Programų sistemų inžinerijoje, arba tiksliau – objektinėje programų sistemų inžinerijos paradigmoje, projektavimo šablonai buvo pasiūlyti Gamma'os ir bendraautorių (Gamma, et al. 1994). Terminas *projektavimo šablonas* (angl. *design pattern*) yra pasiskolintas iš architektūros terminijos, kurioje jį suformulavo žinomas architektas Alexander'is (Alexander, et al., 1977). Gamma ir jo kolegos savo knygoje pritaikė Alexander'io suformuluotą šablonų apibrėžtį programų sistemų inžinerijai ir pasiūlė objektinius projektavimo šablonus aprašyti naudojant 4 esminius elementus: pavadinimą, problemą, sprendimą ir pasekmes. Projektavimo šablonus galima apibūdinti kaip abstrakčią schemą, taip apibendrinančią anksčiau spręstų projektavimo problemų sprendimo būdus, kad tą schemą būtų galima panaudoti naujoms panašaus pobūdžio projektavimo problemoms spręsti (MacDonald et al., 2002). Projektavimo šablonai aprašo abstrakčią projektavimo sprendimo idėją, nusakančią, kaip projektuoti žemesnio lygmens architektūros sudedamąsias dalis – posistemes ir komponentus, tačiau nėra žemiausio lygmens šablonai. Žemiausio lygmens projektavimo šablonai yra vadinami *idiomomis*. Gamma ir jo kolegos ne tik pasiūlė būdą, kaip aprašyti šablonus, bet ir sudarė išsamų 23 objektinių šablonų katalogą (Gamma, et al. 1994). Nors projektavimo šablonai atsirado objektinėje paradigmoje, jie išplito ir į kitas programų sistemų inžinerijos paradigmas, pavyzdžiui, į aspektinę paradigmą.

Pats terminas *paradigma* programų sistemų inžinerijoje pirmą kartą buvo pavartotas 1978 m. Robert'o W. Floyd'o. Jis teigė, kad programavimo (arba programų sistemų inžinerijos) paradigma yra susijusi su bendra taisykle panašioms problemoms spręsti, su tuo, kad kuriasi jos naudotojų bendruomenės ir kad remiantis ja yra kuriamos programavimo kalbos (Floyd, 1979). Viena iš naujausių, besiformuojančių programų sistemų inžinerijos paradigmų yra aspektinis programavimas, pasiūlytas kai kurių tyrėjų (Kiczales et al, 1997; Lopes, 2005). Aspektinis programavimas sprendžia objektinėje paradigmoje kylančią turinių susipynimo problemą. Aspektinė paradigma siūlo būdą, leidžiantį panaikinti ar bent sumažinti turinių susipynimą objektinėje programų sistemoje. Nors ir pasiekta pažangos, tačiau aspektinėje paradigmoje vis dar nėra pakankamai ištobulinti aspektinėms sistemoms projektuoti skirti projektavimo šablonai. Pirmoji aspektinio programavimo kalba *AspectJ* buvo pasiūlyta pačių aspektinės paradigmos autorių (Kiczales et al., 2001). Ji ir išlieka viena populiariausių ir labiausiai išvystytų aspektinių programavimo kalbų. Pagrindinė aspektinės paradigmos idėja yra ta, kad turinių susipynimą galima perkelti taisyklių pavidalu į aspektus, o jų išbarstymą sistemoje atlikti automatiškai, naudojant vadinamąjį *aspektų supyniklį* (angl. *aspect weaver*). Pagrindiniai supynimui aprašyti skirti aspekto elementai yra šie: *pjūvis* (angl. *pointcut*), aprašantis keletą to paties turinio supynimo su kitais turiniais taškų; *įterpiamasis kodas* (angl. *advice*), t. y. programinis kodas, kuris tuose taškuose yra įterpiamas į minėtąjį turinį; *tarptipiniai aprašai* (angl. *intertype declarations*), t. y. *aspektuose pateikiamos* apibrėžtys, aprašančios interfeiso, klasės arba aspekto tipą. Tarptipiniai aprašai gali būti naudojami, pavyzdžiui: tipų hierarchijoms modifikuoti, naujiems metodams į esamas klases pridėti ir kitiems uždaviniams (reikalingiems statiškai atskirti susipynusius turinius vieną nuo kito) realizuoti.

Programiniai karkasai, kaip ir prieš tai aptarti projektavimo šablonai bei programų sistemų inžinerijos paradigmos, bendru atveju taip pat gali būti įvardyti kaip vienas iš programų sistemų inžinerijos metodų, siekiančių pakartotinai panaudoti pasiteisinusias programines konstrukcijas, tačiau pats būdas šiam tikslui pasiekti skiriasi. Programinis karkasas yra suprantamas kaip pakartotinai panaudojama, iš dalies sukurta, programinė konstrukcija, kuri gali būti užbaigiama, specializuojama ir pasirinktinai modifikuojama karkaso naudotojų, kuriančių galutinius programinius produktus. Tokie karkasai gali būti skirstomi į: taikomuosius, dalykinius ir pagalbinius karkasus. *Objektiniu karkasu* yra vadinamas toks karkasas, kurį panaudojant yra kuriamos programos, susidedančios iš tarpusavyje sąveikaujančių objektų rinkinio. Anot Johnson'o ir Foot'o, karkasas yra klasių rinkinys, įgyvendinantis abstraktų susijusių problemų sprendimų modelį (Johnson, Foote, 1988). Pagrindinė tokio karkaso struktūra apima fiksuotąją ir kintamąją dalis (Froehlich et al, 1998). Fiksuotoji dalis apima klasių bibliotekas ir kodo skeletą (Kirk, 2005). Kintamoji dalis yra karkaso užpildas, kuris yra kuriamas karkaso naudotojų. Taip karkasas paverčiamas užbaigta programų sistema. Taškai, kuriuose galima „sujungti" karkaso kintamąją ir fiksuotąją dalis, yra vadinami karkaso *užpildymo taškais* (anlg. *hot spots*). Kodo skeleto arba klasių bibliotekos dalys, kurios niekuomet nesikeičia, yra vadinamos *fiksuotais taškais* (angl. *frozen spots*) (Froehlich et al, 1998). Užpildymo taškuose yra paliekami vadinamieji karkaso *užpildymo metodai* (angl. *hooks*) (dažniausiai – abstrakčiosios operacijos). Karkaso užpildymas gali būti atliekamas naudojant paveldėjimo arba komponavimo mechanizmus trūkstamam klasių funkcionalumui realizuoti. Šioje disertacijoje yra nagrinėjami taip pat ir aspektiniai dalykiniai karkasai. Tai yra tokie dalykiniai karkasai, kuriuose šalia tradicinių objektinių karkaso užpildymo mechanizmų yra naudojami ir aspektiniai karkaso užpildymo taškai.

2. Susijusių darbų analizė

Dažniausiai naudojamas GoF objektinių projektavimo šablonų transformavimo į aspektinius šablonus būdas yra tiesioginis objektinius šablonus realizuojančio objektinio kodo perrašymas į aspektinį kodą (papildymas aspektais) (Hannemann, Kiczales 2002). Pirmieji tokio pobūdžio bandymai transformuoti objektinius projektavimo šablonus į aspektinius (Hannemann, Kiczales, 2002) buvo atlikti dar 2002 m., o rezultatai patobulinti 2004 m. Darbe pasiūlyta, kaip objektinius šablonus realizuoti *AspectJ* programavimo kalba. Hannemann'o ir Kiczales'o straipsnis iki šiol lieka savotišku modeliniu pavyzdžiu, savo darbuose juo remiasi daugelis autorių. Tačiau aspektinio kodo realizacijos jame buvo gautos tiesiogiai perrašant *Java* programavimo kalba realizuotus objektinius projektavimo šablonus. Daugelyje panašių tyrimų bandoma parodyti, kad aspektinis programavimas pagreitina patį programavimo procesą ir palengvina sukurtų programų sistemų priežiūrą (Papapetrou, Papadopoulos. 2004). Vienas iš detalesnių tyrimų, kuriame bandoma analizuoti aspektinių šablonų kompozicijas didelėse sistemose, pateiktas Cacho ir bendraautorių darbe (Cacho et al., 2005).

Kitas objektinių projektavimo šablonų transformavimo į aspektinius būdas yra naujo aspektinio projektavimo sprendimo paieška tai pačiai projektavimo problemai spręsti (Hachani, Bardou, 2002; Hirschfeld et al., 2003). Abu jie (antrasis gal kiek mažiau) yra susiję su turinių išsibarstymu ir susipynimu objektinius projektavimo šablonus realizuojančiame programos kode.

Visiškai nauja aspektinių projektavimo šablonų grupė yra siūloma ne viename iš darbų (Hanenberg, Costanza, 2002; Hanenberg, Schmidmeier, 2003; Laddad, 2003;

Schmidmeier, 2004; Miles, 2004; Griswold et al., 2006; Lagaisse, Joosen, 2006; Bynens et al., 2007; Bynens, Joosen, 2009; Menkyna et al., 2010). Šie projektavimo šablonai yra skirti aspektams būdingoms problemoms spręsti. Tačiau daugelis šių tyrimų metu gautų šablonų yra vis dar vystymosi stadijoje arba jų taikymas yra paremtas atsitiktine patirtimi. Šiuos projektavimo šablonus yra siūloma skirstyti pagal juose naudojamas aspektines konstrukcijas:

- pjūvių;
- įterpiamojo kodo;
- tarptipinių aprašų.

Galima išskirti keletą aspektams būdingų projektavimo šablonų pavyzdžių:

- *Kirmgrauža* (angl. *Wormhole*) – naudojamas, kai reikia perkelti kontekstinę informaciją iš vieno metodų kreipinių grandinės galo į kitą, nenaudojant parametrų (pjūvių kategorija);
- *Dalyvis* (angl. *Participant*) – naudojamas, kai reikia realizuoti abstraktų pjūvį skirtingose posistemėse (pjūvių kategorija);
- *Direktorius* (angl. *Director*) – abstraktus aspektas, apibrėžiantis kelis skirtingus klasių interfeisus, nurodančius juos realizuojančių klasių vaidmenis (tarptipinių aprašų kategorija);
- *Sienų kontrolė* (angl. *Border Control*) – naudojamas, kai reikia kontroliuoti pjūvių galiojimo bazinėje programoje sritis (pjūvių kategorija);
- *Gegutės kiaušinis* (angl. *Cuckoo's Egg*) – naudojamas, kai reikia pakeisti kuriamus objektus kitais (įterpiamojo kodo kategorija);
- *Vykdančiojo objekto sukūrimas* (angl. *Worker Object Creation*) – naudojamas, kai reikia generuoti metodo kvietimą ir jį perduoti nurodytam vykdančiajam objektui (įterpiamojo kodo kategorija);
- *Išimčių apdorojimo kūrimas* (angl. *Exeption Introduction*) – naudojamas, kai reikia aspektuose generuoti išimtis apdorojantį kodą (įterpiamojo kodo kategorija);
- *Susitarimas apie elgseną* (angl. *Policy*) – naudojamas apibrėžti tokias elgsenos taisykles, kurias pažeidus turi būti generuojamas perspėjimas arba pranešimas apie padarytą klaidą (tarptipinių aprašų kategorija).

3. GoF projektavimo šablonams transformuoti į aspektinius šablonus skirtų metodų ir procedūrų projektavimas

Kai kurios projektavimo problemos gali būti vadinamos *nuo konkrečios paradigmos nepriklausomomis problemomis* (angl. *paradigm-independent design problems*). Dažnai pasitaiko, kad skirtingose paradigmose sprendžiamos problemos yra panašios. Pavyzdžiui, problema, kuri abstrakčiai gali būti formuluojama, kaip poreikis atskirti vartotoją ir naudojamus resursus, objektinėje paradigmoje yra sprendžiama Fasado projektavimo šablonu, tačiau pati problema gali būti suprantama ir kaip nuo konkrečios paradigmos nepriklausoma problema. Specifiškai objektinei paradigmai ši problema ir jos sprendimas yra formuluojami taip: problema – siekiant paprastesnio naudojimo yra reikalingas bendras interfeisas, jungiantis posistemių interfeisus; sprendimas – reikia apibrėžti naują klasę, kuri paslepia kelių kitų klasių interfeisus, sujungdama juos į vieną bendrą interfeisą.

Galima sakyti, kad projektavimo problema gali būti nuo paradigmos nepriklausoma, tačiau jos sprendimas ir realizacija visuomet yra išreiškiami per konkrečios paradigmos

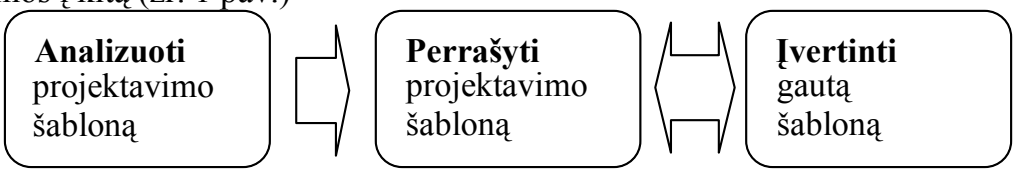
konceptus. Tuo tarpu tam tikrai paradigmai būdinga projektavimo problema yra nepriklausoma nuo konkrečios programavimo kalbos, tačiau jos sprendimas ir realizacija yra išreiškiami per konkrečios programavimo kalbos konceptus. Panašiai galima apibrėžti ir idiomų sprendžiamas problemas: tai tam tikrai programavimo kalbai būdinga projektavimo problema, kuri priklauso nuo konkrečios programavimo kalbos, o jos sprendimas bei realizacija yra išreiškiami per konkrečios programavimo kalbos konceptus.
Norint pritaikyti egzistuojančius objektinius šablonus aspektinei paradigmai, tai galima padaryti šiuo metu literatūroje aprašomais dviem būdais: perrašyti objektine paradigma gaunamas šablonų realizacijas aspektinės paradigmos konceptais arba ieškoti naujo aspektinio sprendimo nagrinėjamo objektinio šablono problemai spręsti.

1 lentelė Šablonų sprendžiamų problemų bei jų sprendimų klasifikacija

Sprendimai **Problemos**	**objektinis**	**aspektinis**	**mišrusis**
nepriklausomos nuo paradigmos (Adapterio šablonas angl. *Adapter pattern*)	naudojami šablonai, susidedantys iš objektų (Gamma et al., 1994)	naudojami šablonai, susidedantys iš aspektų	naudojami šablonai, susidedantys iš aspektų ir objektų (Hannemann, Kiczales, 2002)
būdingos objektinei paradigmai (Prototipo šablonas angl. *Prototype pattern*)	naudojami šablonai, susidedantys iš objektų (Gamma et al., 1994)	naudojami šablonai, susidedantys iš aspektų, supintų su bazine programa (Laddad, 2003, Miles, 2004)	naudojami šablonai, susidedantys iš aspektų, supintų su bazine programa, ir objektų (Hannemann, Kiczales, 2002, Laddad, 2003, Miles, 2004; Hanenberg, Unland, 2003)
būdingos aspektinei paradigmai (Grandininio įterptinio kodo šablonas angl. *Chained Advice pattern*)	naudojami šablonai, realizuoti bazine objektine programa, atsižvelgiant į projektuojamus aspektus (Griswold, et al., 2006; Bynens, Joosen, 2009)	naudojami šablonai, susidedantys iš aspektų (Miles, 2004, Hanenberg; Unland, 2003; Bynens et al., 2007)	naudojami šablonai, susidedantys iš aspektų, ir bazinė objektinė programa, realizuota atsižvelgiant į projektuojamus aspektus (Laddad, 2003; Hanenberg, Unland, 2003)

Niekas nenagrinėja tokio atvejo, kai objektinis projektavimo šablonas yra apibendrinamas, paverčiant sprendžiamą problemą nuo paradigmos nepriklausoma problema. Tuo atveju ši problema yra pritaikoma aspektinei paradigmai, o problemos sprendimas yra išreiškiamas naujos paradigmos konceptais. Pastarasis būdas gali pasirodyti sudėtingesnis nei esamieji. Iš pirmo žvilgsnio atrodo, kad galima analogiškai, kaip ir objektams, sukonstruoti projektavimo šablonus aspektams, tačiau ne visiems projektavimo šablonams tai galima taikyti, be to, neaišku, ar egzistuoja kontekstas, kuriame toks projektavimo šablonas gali būti pritaikytas. Tam tikrais atvejais projektavimo šablonas tampa nebereikalingas, nes gali būti tiesiogiai išreiškiamas programavimo kalboje. Pavyzdžiui, Singletono projektavimo šablonas nėra reikalingas aspektams, nes visi aspektai yra singletonai. Svarbu atkreipti dėmesį ir į tai, kad šiame tyrime analizuojamos tik dvi paradigmos: objektinė paradigma ir aspektinė paradigma. Aspektinė paradigma yra išskirtinė tuo, kad naudojama poroje kartu su objektine paradigma (arba kokia nors kita paradigma, kurios kalba yra realizuojama bazinė programa).

Projektavimo šablonų skirstymas objektinėje paradigmoje ir aspektinėje paradigmoje yra nevienodas. Kadangi šiame tyrime buvo reikalingas skirstymas, siejamas su sprendžiamomis problemomis bei apimantis visus ir aspektinius, ir objektinius šablonus, buvo pasiūlyta šablonų sprendžiamų problemų bei jų sprendimų klasifikacija (žr. 1 lentelę). Toks šablonų sprendžiamų problemų bei jų sprendimų skirstymas padėjo apibrėžti grynųjų aspektinių projektavimo šablonų, sprendžiančių nuo paradigmos nepriklausomas problemas, klasę. Siekiant užpildyti šią klasę disertacijoje buvo pasiūlytas naujas šablonų transformavimo būdas, nusakantis kaip šablonai, sprendžiantys nuo paradigmos nepriklausomas problemas, turi būti transformuojami iš vienos paradigmos į kitą (žr. 1 pav.)



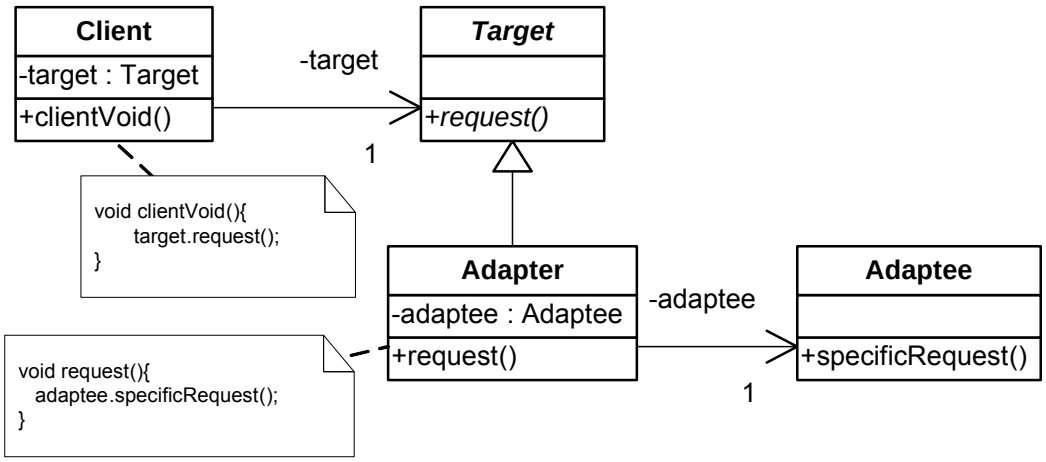
1 pav. Pagrindiniai objektinių šablonų transformavimo į aspektinius žingsniai

Jeigu GoF projektavimo šablonas gali būti realizuojamas naudojant tik klases singletonus, tai toks šablonas yra laikomas kandidatu tapti nuo paradigmos nepriklausomu projektavimo šablonu. Perrašant atrinktą projektavimo šabloną AspectJ programavimo kalba, visos jo klasės turi būti pakeičiamos aspektais, o visi jo objektų konstruktoriai – *AspectJ* statiniu metodu *aspectOf*, kuris grąžina prieigą prie aspekto egzemplioriaus. Gautas projektavimo šablonas turi būti analizuojamas, siekiant surasti ir išimti nereikalingus atributus bei metodus.

Pritaikius aptartą transformavimo būdą 23 objektiniams GoF projektavimo šablonams buvo gautas 20 grynųjų GoF aspektinių šablonų rinkinys. Šie 20 projektavimo šablonų sprendžia nuo konkrečios programų sistemų inžinerijos paradigmos nepriklausomas projektavimo problemas, todėl gali būti panaudoti projektuojant sistemos dalis ar komponentus, susidedančius tik iš aspektų.

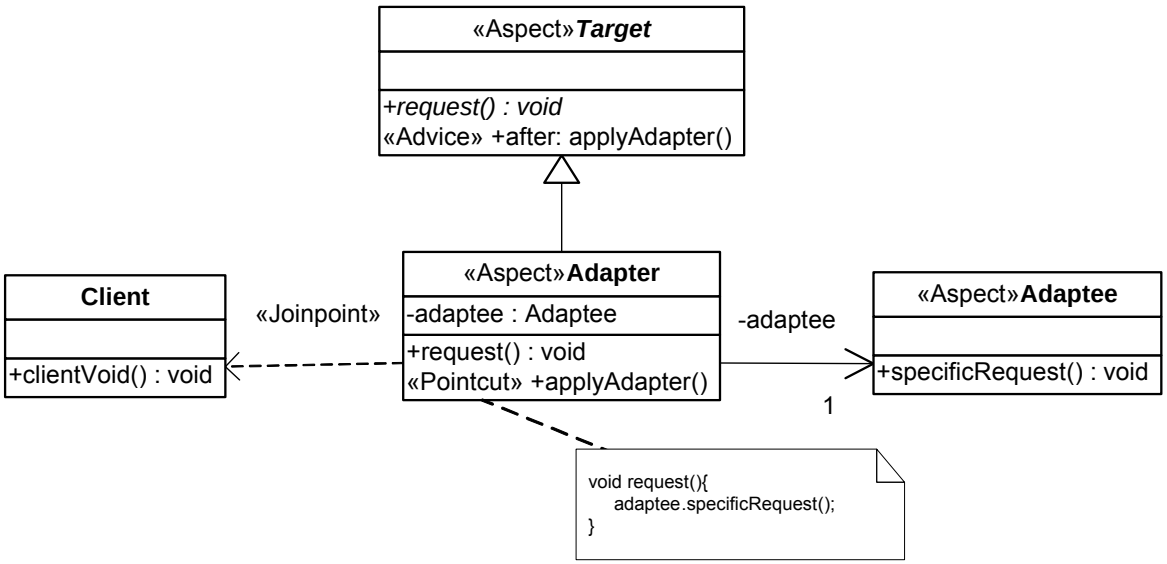
Vienas iš geriausiai šablonų transformavimo būdą iliustruojančių pavyzdžių yra objektinio adapterio projektavimo šablono transformavimo į aspektinį pavyzdys. Objektinių projektavimo šablonų struktūrą įprasta vaizduoti UML klasių diagramomis.

Abstrakti objektinio adapterio projektavimo šablono struktūra yra pateikiama 2 paveiksle.



2 pav. Objektinis adapterio projektavimo šablonas

Norint gauti aspektinį šio šablono variantą reikia pritaikyti pasiūlytus transformavimo būdo žingsnius. Paveiksle (2 pav.) vaizduojamos klasės *Target*, *Adapter* ir *Adaptee* yra pakeičiamos aspektais *Target*, *Adapter* ir *Adaptee*. Klasė *Client* išlieka nepakitusi, tačiau ji nebėra traktuojama kaip šablono dalis ir yra vaizduojama tik siekiant parodyti, kaip šablonas bus aktyvuojamas, t. y šioje klasėje yra dirbtinai sukuriamas supynimo taškas, kuriame bus vykdomas atitinkamo aspekto įterpiamasis kodas. Tokiu būdu yra gaunamas šablono sprendimas, kurį sudaro vien tik aspektai (žr. 3 pav.).



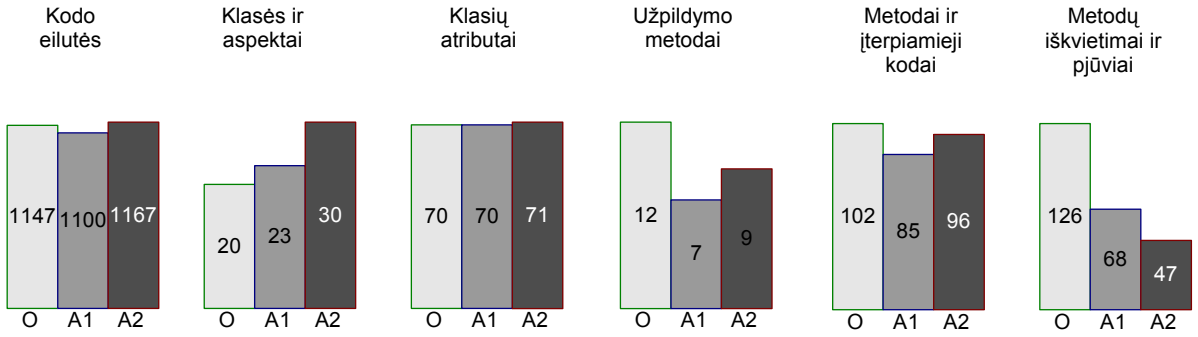
3 pav. Aspektinis adapterio projektavimo šablonas

Diagramoje (3 pav.) yra taikoma ta pati UML notacija, tačiau siekiant pavaizduoti aspektus ji yra išplėsta, panaudojant stereotipus: *Aspect* – nurodant, kad tai yra aspektas, *Pointcut* – nurodant, kad tai yra pjūvis, ir *Advice* – nurodant, kad tai yra įterptinis kodas. Abstrakčiame aspekte *Target* yra apibrėžta abstrakti operacija *request* ir *ApplyAdapter* pjūvio įterpiamasis kodas. Aspekte *Adaptee* yra apibrėžtas metodas *specificRequest*, kurį reikia adaptuoti panaudojant *Adapter* aspektą. Aspekte *Adapter* yra apibrėžiami abstrakčią operaciją realizuojantis *request* metodas ir *applyAdapter* pjūvis. Aspektas

Adapter pasinaudodamas *request* metodu kviečia *Adaptee* aspekto *specificRequest* operaciją.

4. Empirinis transformuotų projektavimo šablonų vertinimas

Šiame disertacijos skyriuje pateikti trijų atskirų atvejų empiriniai tyrimai, kuriuose yra demonstruojama, kaip objektiniai projektavimo šablonai gali būti transformuojami į aspektinius, ir nagrinėjama, kaip tikslinga šitaip transformuotus šablonus panaudoti aspektiniams dalykiniams karkasams projektuoti. Visais trimis atvejo analizės tyrimais siekiama išsiaiškinti pertvarkytų šablonų panaudojimo galimybes realiose programų sistemose. Visi įrodymai šiuose tyrimuose remiasi kokybinių duomenų, tokių kaip realizacijų diagramos bei detalūs aprašymai, analize. Antrame ir trečiame atvejo analizės tyrimuose, siekiant gauti tvirtesnių įrodymų, šalia kokybinių buvo analizuojami ir kiekybiniai tyrimo duomenys, tokie kaip kodo eilučių, klasių ir aspektų, klasių atributų, karkaso užpildymo metodų, metodų ir įterpiamųjų kodų bei metodų iškvietimų ir pjūvių skaičius. Matavimų rezultatai yra pateikiami diagramose. Pavyzdžiui, antrojo atvejo analizės tyrimo metu buvo gauti tokie duomenys (4 pav.), kur stulpeliai, pažymėti *O*, *A1* ir *A2,* yra atitinkamai objektinės sistemos duomenys ir pirmosios bei antrosios aspektinių sistemų projektavimo iteracijų duomenys.



4 pav. Antrojo atvejo analizės tyrimo statiniai matavimų duomenys

Atskiro atvejo analizė yra empirinis tyrimo metodas, kuriuo yra siekiama ištirti fenomeną tam tikrame kontekste (Runeson, Höst, 2009). Disertacijoje atskiro atvejo analizės tyrimo metodas yra naudojamas, siekiant išsiaiškinti, kokį poveikį GoF_{AO} šablonai padaro aspektinių dalykinių skaidriosios dėžės karkasų projektavimui. Šie tyrimai taip pat yra pozityvistiniai tyrimai (Benbasat et al., 1987), kuriuose yra matuojami kintamieji, tikrinamos hipotezės ir iš kelių pavyzdžių daromos išvados apie visą aspektinių dalykinių skaidriosios dėžės karkasų populiaciją. Tyrimo metu yra analizuojami tiek projektavimo rezultatai, tiek pats projektavimo procesas. Šiam tikslui yra taikomas tyrimo konstravimu metodas (Lukka, 2003). Tyrimas konstravimu yra eksperimentinio tyrimo procedūra, kuri gali būti naudojama hipotezėms tikrinti, kuriant naujovišką konstrukciją, kurioje yra realizuojamos tiriamųjų hipotezių prielaidos. Apskritai tai turi būti abstrakti konstrukcija, kuri gali būti realizuojama potencialiai begaliniu realizacijų skaičiumi. Šioje disertacijoje tokia konstrukcija yra aspektinis dalykinis karkasas. Nauja konstrukcija ir jos kūrimo procesas yra suprantami kaip pagrindiniai įrankiai, skirti naujoms hipotezėms tikrinti, tobulinti ar net kurti.

Įprasta manyti, kad atskiro atvejo analizės tyrimo metodas turėtų būti naudojamas tik hipotezėms falsifikuoti, tačiau tam tikrais atvejais tokių tyrimų rezultatai gali būti sėkmingai apibendrinami (Flyvbjerg, 2004). Apibendrinimo galimybė tiesiogiai

priklauso nuo to, kaip yra parenkami tiriamieji atvejai: norint padidinti apibendrinimo galimybę, reikia strategiškai parinkti kritinį arba tipinį atvejį.
Šioje disertacijoje yra pateikiami vienas kritinio ir du tipinio atvejų analizės tyrimai. Atskiro atvejo analizės tyrimo metodas yra naudojamas iškeltoms hipotezėms tikrinti, o tyrimo konstravimu metodas (Crnkovic, 2010) yra naudojamas atliekant eksperimentinius tyrimus, nagrinėjančius aspektinių šablonų taikymą aspektiniams dalykiniams karkasams projektuoti. Atlikti tyrimai parodė, kad 20 GoF projektavimo šablonų gali būti transformuojami į grynuosius aspektinius šablonus (20 GoF_{AO} šablonus), skirtus projektuoti aspektams, ir kad tokie šablonai yra reikalingi projektuojant aspektinius užpildymo taškus aspektiniuose dalykiniuose karkasuose. Eksperimentiniai tyrimai buvo suprojektuoti, siekiant patikrinti šias hipotezes:

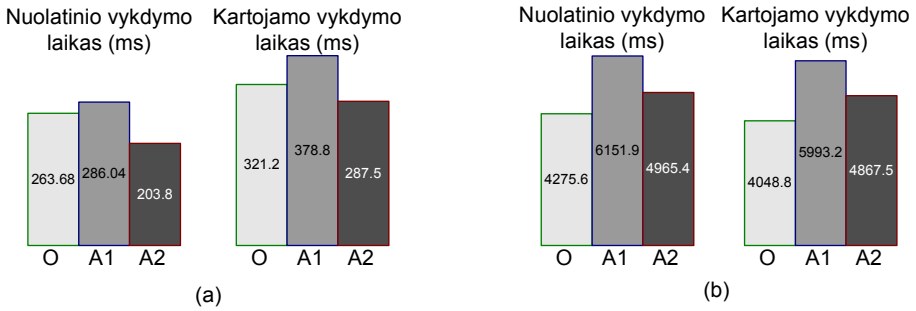
- Projektuojamos programų sistemos yra efektyvesnės tada, kai grynieji aspektiniai šablonai yra naudojami kartu su objektiniais šablonais.
- Skaidriosios dėžės tipo aspektiniuose dalykiniuose karkasuose grynųjų aspektinio projektavimo šablonų panaudojimas duoda galimybę projektuoti naujo tipo užpildymo taškus (t. y. užpildymo taškus suprojektuotus, panaudojant abstrakčiuosius aspektus).
- Aspektiniuose dalykiniuose karkasuose grynųjų aspektinių projektavimo šablonų panaudojimas sumažina tuose karkasuose turinių susipynimą.
- Aspektinių dalykinių karkasų projektavimas, naudojant grynuosius aspektinio projektavimo šablonus, nedaro poveikio programų sistemų, sukurtų panaudojant tuos karkasus, našumui.

Visiems trims eksperimentiniams tyrimams buvo suformuluota bendra tyrimo metodika. Nors kiekvienos atskiro atvejo analizės uždaviniai yra formuluojami specifiškai, tyrimo metodiką sudaro 6 bendri žingsniai:

1) identifikuoti karkasui reikalingus aspektus;
2) nustatyti, kokius projektavimo šablonus tikslinga panaudoti tiems aspektams projektuoti;
3) suprojektuoti ir realizuoti aspektus, aprašyti stebėjimus ir atradimus bei surinkti kitus kokybinius duomenis;
4) suprojektuoti gautam kodui testuoti reikalingus testus ir atlikti kodo testavimą, atlikti matavimus ir surinkti kiekybinius duomenis;
5) įvertinti kodo struktūrą pagal pasirinktus kriterijus;
6) išanalizuoti, apibendrinti surinktus duomenis ir įvertinti hipotezes.

Pirmame ir antrame atvejo analizės tyrime yra naudojamas *SimJ* karkasas, sukurtas imitacinio modeliavimo uždaviniams spręsti. Tyrimo metu karkasas yra pertvarkomas iš objektinio į aspektinį. Trečiame atvejo analizės tyrime yra naudojamas *SimpleW* karkasas, skirtas internetinėms dalykinėms programoms projektuoti. Kadangi projektuojant karkasus dažniausiai yra taikomas iteratyvusis procesas, duomenys buvo renkami kiekvienos iteracijos metu gautam karkasui atskirai.
Tyrimų rezultatų kokybiniai ir kiekybiniai duomenys yra pateikiami diagramose, jų interpretavimas yra detalizuojamas tekstiniame apraše. Kodo sudėtingumas yra matuojamas konkrečiu atveju matuojamų elementų kiekiais (pvz., kodo eilučių skaičius, klasių ir aspektų skaičius ir pan.).
Pirmos trys hipotezės buvo patvirtintos abiem aspektinėmis karkaso versijomis. Paskutinė hipotezė buvo paneigta, kadangi galimas našumo sumažėjimas vis dėlto gali pasireikšti atskirais atvejais, priklausomai nuo to, koks šablonas yra taikomas ir ar

taikomas teisingai. Našumo pokyčius geriausiai atspindi antrojo atvejo analizės tyrimo duomenys (žr. 5 pav.), kadangi šiame tyrime imitacinio karkaso užduotims vykdyti yra reikalingos nemažos laiko sąnaudos, o informacijos pateikimui ekrane laiko sugaištama palyginti nedaug.



5 pav. Antrojo atvejo analizės tyrimo programos vykdymo našumo matavimų duomenys

Abiejose (a) ir (b) (5 pav.) diagramose yra pateigiamas programos vykdymo laikas milisekundėmis. Diagramos yra kelios, kadangi buvo matuojamas atskirų užduočių vykdymas. Diagramų stulpeliai, kaip ir prieš tai buvusiose diagramose, pažymėti *O*, *A1* ir *A2*, yra atitinkamai objektinės sistemos duomenys ir pirmosios bei antrosios aspektinių sistemų projektavimo iteracijų duomenys.

5. Neatsakyti klausimai ir tyrimo kontekstas

Disertacijoje yra keletas diskutuotinų klausimų, kuriuos būtina paminėti. Visų pirma – galimybė sudaryti aspektų hierarchijas ir naudoti komponavimą bei delegavimą aspektams. Nors tai yra pagrindinės technologijos, naudojamos klasėms ir objektams projektuoti, jos taip pat yra susijusios ir su neigiamais efektais, tokiais kaip realizacijos kodo susipynimas ir išsibarstymas. Vis dėlto šioje disertacijoje remiamasi prielaida, kad aspektų hierarchijos ir komponavimas yra naudingi tol, kol tai neperžengia projektuojamų turinių ribų. Ši prielaida yra patvirtinama atskirų atvejo analizės tyrimų metu. Nėra jokių duomenų, leidžiančių teigti, kad nuosekliai projektuojant aspektų struktūras galima kaip nors padidinti projektuojamų programų sudėtingumą.

Diskutuotinas taip pat yra teiginys, kad visi aspektai yra singletonai. *AspectJ* kalboje yra galimybė naudoti kelis to paties aspekto egzempliorius, ribojant jų vykdymą taip, kad tam tikras aspekto egzempliorius egzistuoja tik konkretaus objekto gyvavimo metu arba valdymo perdavimo konkrečiam metodui metu. Vis dėlto toks aspektų egzempliorių susiejimas su objektų egzistavimu smarkiai skiriasi nuo pačių objektų egzempliorių sukūrimo ir gyvavimo. Be to, jei aspektų egzempliorius galima būtų kurti taip pat kaip ir objektų egzempliorius, kiltų keblumų, norint juos suvaldyti (pvz., užtikrinti, kad įterptinis kodas nepasikartotų kelis kartus).

Disertaciniame darbe yra apsiribojama tam tikru tyrimo kontekstu, susijusiu su naudojamomis technologijomis ir priemonėmis tyrimuose. Pagrindiniai apribojimai yra tokie: disertacijoje yra nagrinėjami skaidriosios dėžės karkasai, realizuojant programos kodą naudojamos *Java* ir *AspectJ* programavimo kalbos, nagrinėjami išimtinai GoF projektavimo šablonai, be to, kiekybinių matavimų patikimumas yra glaudžiai susijęs su naudojamais matavimo būdais.

Išvados

1. Aspektinio projektavimo šablonų, sukurtų tiesioginiu objektinio kodo perrašymu ir naudojančių abiejų – aspektinės ir objektinės – paradigmų konstrukcijas, nepakanka tam, kad programų sistemose būtų galima visiškai atskirti susipynusius

turinius vienas nuo kito. Juose nėra priemonių aspektinių dalykinių karkasų užpildymo taškams realizuoti. Be to, jie nėra pakankamai universalūs, nes gali būti naudojami tik labai specifiniuose kontekstuose.

2. Dvidešimt iš GoF23 objektinių projektavimo šablonų, pasiūlytų Gamma'os (Gamma et al., 1994) (GoF šablonų), tinka spręsti tokias projektavimo problemas, kurios yra aktualios taip pat ir aspektinėje programų inžinerijos paradigmoje. Šiuos šablonus aspektinėje paradigmoje galima realizuoti nenaudojant jokių objektinės paradigmos konstrukcijų, t. y. tokius šablonus galima realizuoti kaip grynuosius aspektinio projektavimo šablonus. Taigi aspektus galima naudoti kaip tarpusavyje sąveikaujančias esybes, t. y. galima kurti aspektų hierarchijas ir apibrėžti priklausomybes bei asociacijas tarp aspektų.
3. Grynųjų aspektinio projektavimo šablonų taikymas sumažina turinių susipynimą aspektiniuose dalykiniuose karkasuose ir leidžia juose projektuoti naujos rūšies užpildymo taškus, t. y. užpildymo taškus, realizuotus naudojant abstrakčiuosius aspektus.
4. Disertacijoje atliktos keleto atskirų atvejų analizės rezultatai patvirtino, kad 20 grynųjų aspektinio projektavimo šablonų sumažina kodo sudėtingumą, eliminuoja turinių susipynimą ir leidžia projektuoti papildomus aspektinius užpildymo taškus karkasuose. Našumo testai parodė, kad tam tikrais atvejais yra tikėtinas našumo sumažėjimas. Našumas iš esmės priklauso nuo konkretaus taikomo projektavimo šablono ir nuo pačių projektuotojų įgūdžių, t. y. kaip tinkamai buvo parinktas ir pritaikytas projektavimo šablonas.
5. Atskirų atvejų analizės metu atlikto aspektinio dalykinio karkaso konstravimo proceso analizė parodė, kad, norint gauti sėkmingus rezultatus, yra reikalingi tokie konstravimo proceso žingsniai:
    a. analizuojant reikalavimų specifikaciją, identifikuoti reikalingus aspektus karkaso moduliams, kuriuos reikia projektuoti kaip susikertančius turinius;
    b. nustatyti, kokius karkaso užpildo taškus reikia projektuoti objektams ir kokius aspektams bei kokius projektavimo šablonus tikslinga panaudoti jų projektavimui;
    c. suprojektuoti ir realizuoti aspektus,
    d. suprojektuoti gautam kodui testuoti reikalingus testus ir atlikti kodo testavimą, jei rezultatas netenkina papildyti projektą ir pereiti į *c* žingsnį.

Autoriaus mokslinių publikacijų disertacijos tema sąrašas

Straipsniai recenzuojamuose moksliniuose leidiniuose

Vaira, Ž.; Čaplinskas, A. (2011a). *Application of pure aspect-oriented design patterns in the development of AO frameworks: A case study*. Informacijos mokslai, 56 t., p. 146–155.

Vaira, Ž.; Čaplinskas, A. (2011b). *Paradigm-independent design problems, GoF 23 design patterns and aspect design*. Informatica, IOS Press, 22(2), p. 289–317.

Straipsniai kituose leidiniuose

Vaira, Ž.; Čaplinskas, A. (2011). Case Study Towards Implementation of Pure Aspect-oriented Factory Method Design Pattern. *III tarptautinės konferencijos darbai, International Conference on Pervasive Patterns and Applications,* PATTERNS 2011, rugsėjo 25–30 d., 2011, Roma, Italija.

Vaira, Ž.; Čaplinskas, A. (2009). Kompozicinės aspektinių šablonų savybės. *Lietuvos matematikų draugijos L konferencijos darbai, Lietuvos matematikos rinkinys*, p. 123–453, 2009, Vilnius, Lietuva.
Vaira, Ž. (2009). Aspektinis programų sistemų projektavimo metodas. *XII studentų mokslinės draugijos konferencijos darbai, Fundamentiniai tyrimai ir inovacijos mokslų sandūroje*. Klaipėdos universitetas, Gamtos ir matematikos mokslų fakultetas, 2009, Klaipėda, Lietuva.

Trumpos žinios apie autorių

Žilvinas Vaira gimė 1981 m. rugpjūčio 22 d. Klaipėdoje. 1999 m. baigė Klaipėdos „Ąžuolyno“ gimnaziją. 2005 m. Klaipėdos universiteto gamtos ir matematikos fakultete įgijo informatikos bakalauro laipsnį. 2007 m. Klaipėdos universiteto gamtos ir matematikos fakultete įgijo informatikos magistro laipsnį. 2007–2011 m. doktorantas Vilniaus universiteto matematikos ir informatikos institute.

INVESTIGATION, IMPROVEMENT AND DEVELOPMENT OF ASPECT-ORIENTED DESIGN PATTERNS

Research Context and Challenges

Mainly, software systems are permanently changed in order to meet new requirements and to adapt them to permanently changing technology. Design modularity decouples design concerns that probably can be changed and in this way facilitates further system changes (Bertrand Meyer, 1997). Object-oriented (OO) software engineering paradigm proposes a number of powerful modularization methods and techniques. Unfortunately, some design concerns, called crosscutting concerns, cannot be modularized using these methods and techniques. The solution of this problem has been proposed by the new emerging software engineering paradigm, aspect-oriented (AO) paradigm (Kiczales, et al., 1997). This paradigm proposes also the solutions of some other software engineering problems that have poor or even no solution in the OO paradigm. For example, one of such problems is the encapsulation of the multiplicity of subjective views in objects. It is very troubling to model several views by one object because different views require that, depending on the view, different properties of the same object would be accessible (Harrison, Ossher, 1993). AO paradigm proposes an elegant solution of this problem. However, this paradigm is still not enough mature. In particular, it is still unknown which design patterns developed in the object-oriented paradigm, for example, design patterns investigated by Gamma et al. (Gamma et al., 1994), can be adapted for aspect-oriented paradigm and how to transform them from one paradigm to another in a systematical way. Gamma et al. are often referred to as the Gang of Four, or GoF, and the patterns investigated by them as GoF design patterns.

Object-oriented design patterns have been developed as a result of in-depth analysis and generalization of best object-oriented design practices. The concept of design pattern has been highly influential to the field of software engineering, first of all, to object-oriented design theory and practice. However, 23 GoF and other OO design patterns have been investigated only in the narrow context of OO paradigm and the extent of their applicability in other paradigms is still an open research question. Although some researches (Hannemann, Kiczales, 2002; Hachani, Bardou, 2002; Hirschfeld et al., 2003) investigate how the 23 GoF design patterns can be rewritten in an AO manner, no one investigated systematically the problem of transformation of OO design patterns into

analogous design patterns in other software engineering paradigms. It means that it is still unknown which design patterns can be applied to solve paradigm independent design problems and which are paradigm-specific, therefore meaningless in other paradigms. In particular, it is very important to answer this question at least for OO and AO paradigms. It is important from both theoretical and practical points of view. OO design patterns have been extracted analyzing a huge amount of successful designs. Although there are ways proposed how to perform some automatic inference of new design patterns (Tonella, Antoniol, 1999), their acceptance is still directly related to successful application of common design ideas many times in many projects. Such pattern gathering process is very slow and expensive. It would not be reasonable to repeat this process for AO paradigm from scratch. It is obvious that it is preferable to rely on the experience gained in other software engineering paradigms, first of all, in OO paradigm and to adapt for AO paradigm the design patterns developed and well-investigated there.

Problem Statement

The subject of the thesis research is pure AO design patterns and their application in the design of AO frameworks. By pure AO design patterns the patterns implemented using aspects only are considered. Mixed AO design patterns, in contrast, are such patterns which are implemented using both, aspects and objects. Usually in mixed design patterns aspects play supporting role and mainly are used only to eliminate concern crosscutting in the pattern implementation code.

The research aims to identify these GoF design patterns that solve OO paradigm independent design problems, to develop techniques for transformation of such patterns to pure AO design patterns, and to investigate the properties of AO domain frameworks developed using the resulted design patterns.

Motivation

Aspect-oriented programming (AOP) emerged as a stand alone paradigm already almost 15 years ago (Kiczales, et al., 1997). However, still very few widely accepted and well documented pure AO design patterns have been proposed. Up to time, even basic OO design patterns – 23 GoF patterns – have not been transformed into pure AO form. Moreover, the question is still open which GoF design patterns can be transformed into pure AO design patterns and why. Although some researches (Bynens, Joosen, 2009; Hanenberg, Schmidmeier, 2003; Laddad, 2003; Miles, 2004) proposed a number of paradigm dependent AO design patterns and idioms, and others (Hannemann, Kiczales, 2002; Hachani, Bardou, 2002; Hirschfeld et al., 2003) investigated how some of GoF design patterns can be redesigned as mixed AO design patterns, all these researches had a sporadic, ad hoc character and they still do not answer the above presented question.

On the other hand, the design patterns play essential role in the development of many applications, especially in the development of various frameworks. There exists a large and well documented experience of application of GoF design patterns in the design of OO frameworks (Adair, 1995; Appleton, 1997; Fayad, Schmidt, 1997; Johnson 1997; Kaisler 2005). It is evident that these and other design patterns facilitate and improve the design of frameworks, make their design documentation more transparent. This is true for OO as well as for AO frameworks. It means that there exist a strong need in pure AO analogues of GoF design patterns and the investigation of the impact of application of these patterns on the run-time properties of framework implementations.

Aims and Objectives of the Research

The research aims to define the class of object-oriented design patterns which can be transformed into pure aspect-oriented ones, proposes a systematic procedure for such transformation and investigates properties of resulting patterns from the viewpoint of their applicability in the design of aspect-oriented domain frameworks. In order to achieve these aims, the following research objectives have been stated:

- evaluate the state of affairs, compare existing approaches to the development of AO design patterns, and highlight their advantages and shortcomings;
- investigate which GoF design patterns solve such design problems that arise in AO paradigm and how these patterns can be transformed into pure AO design patterns;
- investigate applicability of such design patterns in the design of AO domain frameworks and the impact of their application on the complexity of the resulting code, its performance and other run-time characteristics.

Research Questions and Hypotheses

Main questions that need to be answered in this research are:

- How mature is the AO software engineering paradigm currently?
- What techniques can be used to develop AO design patterns and what advantages and shortcomings has each of these techniques got? Does the aspect-oriented paradigm generate some new patterns that are specific only to this paradigm?
- In which different ways design patterns can be implemented, when they solve paradigm independent design problems and design problems that are specific to object-oriented or aspect-oriented software engineering paradigms?
- Is it possible to implement at least some of GoF design patterns using aspect-oriented constructs only? Which and how, if it is possible? Is such implementation in some way better than the object-oriented one? How to measure this?
- In which way are the aspects different as classes from the viewpoint of design patterns and what is the impact of such differences on the structure and other properties of pure AO design patterns?
- What are the advantages of application of pure AO design patterns in real-life applications in general and, particularly, in AO domain frameworks?
- What is the impact of application of pure AO design patterns in AO domain frameworks on the crosscutting, complexity of code implementation and framework run-time performance?

To answer these questions, the following hypotheses have been stated:

- there exist paradigm-independent design problems, at least in the context of OO and AO software engineering paradigms;
- aspect-oriented constructs are sufficient to implement those GoF design patterns that solve paradigm-independent design problems, despite the fact that aspects cannot be directly instantiated;
- efficiency of designs is improved by the usage of pure AO design patterns combined with GoF design patterns;
- the usage of pure AO design patterns allows designing of new kind of hot-spots in white-box AO domain frameworks (i.e. hot spots represented by abstract aspects);
- the usage of pure AO designs patterns reduces crosscutting in AO domain frameworks;

- the development of AO domain frameworks using pure AO design patterns has no particular impact on the overall run-time performance of the applications developed using such frameworks.

Research Design and Research Methods

The research design of present thesis is of an exploratory nature. Aspect-oriented software engineering paradigm is relatively young and the research in this area is still in its infancy. It means that relatively large amount of library research is required in order to define exact structure of a problem, to gain a better understanding of the environment within which the problem arises.

The exploratory nature of the research and the engineering nature of the research subject require that engineering methods would be used to solve the problem under consideration. In this context, the best candidate is constructive research.

Finally, according to (Cooper, Schindler, 1998), any exploratory research is mainly qualitative in his nature. For this reason, it is impossible to validate all obtained results quantitatively, by measurements, because qualitative factors cannot be measured in principle. Additionally, any dissertation research is a small-scale research from both financial and time points of view. It means that in such research it is too expensive and practically impossible ensure high statistical reliability and high level statistical significance of quantitative measurements in cases, when such measurements can be done. Thus, despite its possible biases, the case study methodology is the only practically acceptable methodology to validate the research results.

Taking into account all the discussed above, the research design provides three distinctive research phases, namely, conceptual analysis (Laurence, Margolis, 2003) of related work, constructive research that aims to develop the transformation techniques to transform GoF design patterns into pure AO design patterns, experimental investigation of the applicability of pure AO design patterns in the development of AO domain frameworks.

Conceptual analysis is the analysis of concepts, terms, variables, constructs, definitions, assertions, hypotheses, and theories. It involves examining these for clarity and coherence, critically scrutinizing their logical relations, and identifying assumptions and implications (Machado, Silva, 2007). The goal of conceptual analysis is to increase the conceptual clarity of the research subject. The primary utility of conceptual analysis is to determine the existing state of the research field so that further work may be strategically and appropriately planned (Penrod, Hupcey, 2004). The conceptual analysis of related works has been carried out to generate important theoretical constructs and to provide the theoretical basis for further research as well as to prevent from performing a research that has already been done by others (Hart, 1998). The main field on which conceptual analysis has been performed encompasses both object-oriented and aspect-oriented software design patterns. Generally, conceptual analysis allows answering the questions how mature the AO software engineering paradigm currently is, in which way the aspects are different as classes from the viewpoint of design patterns and what is the impact of such differences on the structure and other properties of pure AO design patterns.

An essential part of conceptual analysis is the categorization of concepts. The categorization has been used as a base to define the class of object-oriented design patterns which can be transformed into pure aspect-oriented ones.

The constructive research approach is a research procedure for producing innovative constructions, intended to solve the problems encountered in the real world and to make some contribution to the theory of the discipline in which it is applied (Lukka, 2003; Crnkovic, 2010). The central notion of this approach, the novel construction, is an abstract notion with a great variety of potential realizations. Models, designs, methods, algorithms, and most other artefacts are considered as constructions. It means that they are invented and developed, not discovered. Mathematical algorithms and new mathematical entities are examples of theoretical constructions. The constructive research approach is based on the belief that by a profound analysis of what works (or does not work) in practice one can make a significant contribution to theory. In the present thesis this approach is used as a methodological basis to develop the transformation rules transforming GoF design patterns to their pure AO analogues, GoF_{AO} design patterns. It is probable that not all of 23 GoF design patterns have pure AO analogues and GoF_{AO} include less than 23 patterns.

As a result of profound analysis of the problem, it has been discovered that aspects are similar to singleton classes. This result suggests that classes in OO design patterns can be replaced by aspects. The details of such transformation should be investigated for each particular pattern and the findings should be generalized in order to develop the rules applicable to all GoF design patterns. The similarities between classes and aspects suggest also that OO patterns, which cannot be implemented, using singleton classes only, cannot be transformed into GoF_{AO} patterns and, consequently, solve OO-specific design problems. Some design patterns out of 23 GoF patterns are dedicated to solve object creation problems. At first glance, the usefulness of such patterns in AO paradigm is highly questionable and should be investigated specifically, if even they can be transformed into GoF_{AO} patterns. Such class patterns are denoted by GoF_{*AO}.

The constructive research methodology is used also for testing of working hypotheses that has been provisionally accepted in the present thesis. One of the advantages of this methodology is that it allows not only to test and investigate the properties of the innovative construction but also to study its development process. On the other hand the constructive research, in parallel with some other methodologies of experimental research, can be viewed as a kind of case study methodology. However, according to the conventional view, case studies should be used for falsification of the hypothesis only. Case study itself cannot prove any hypothesis and should be linked to some hypothetico-deductive model of explanation. However, the correspondence of the case study to real-world situations and its multiple wealth of details state that this view is only partly correct (Flyvbjerg, 2004). Taking into account this argument and the fact that the dissertation research is a small-scale research from both, financial and time points of view, the case study methodology has been approved as the main hypothesis testing methodology. Mainly, the case study is an empirical research method that aims at investigating some phenomena in his context (Runeson, Höst, 2009). In present thesis the aim is to investigate the impact of application of pure AO design patterns on the design of AO domain (white-box) frameworks.

According to (Ragin, 1992) case studies can be enhanced by the strategic selection of cases: critical or typical. A critical case can be thought as an extreme case that is suitable to test hypotheses in critical situations. The case of such AO domain framework, which is designed using at least one GoF_{*AO} design pattern, has been chosen as a critical case. In addition, two typical cases have been chosen: redesign of an existing OO domain

framework into an AO domain framework using GoF_{AO} patterns and the design of a new AO domain framework using GoF_{AO} patterns.
The first typical case is constrained by the existing design of the OO framework and allows investigating the consequences of the redesign when a part of object-oriented framework design has been replaced by relevant pure AO design patterns. Only the parts of the framework that have been affected by some crosscutting of concerns have been reworked. The second typical case has no preliminary design constrains and allows choosing any design that is most suitable for designing aspects. As the result, three cases have been studied.
Generally, quantitative and qualitative data collection methods can be used for evaluation of the results of any case study. Quantitative data relies on numbers that are analyzed using statistics. Qualitative data relies on the text, diagrams and pictures that are analyzed using categorization and sorting. In case studies qualitative data analysis is used more often. The usage of both, qualitative and quantitative data, complimentary provides stronger evidence for the evaluation of the hypotheses (Runeson, Höst, 2009). Thus, both approaches have been used.
The main steps of applied case study methodology can be summarized shortly as follows:

1. identify the aspects that should be designed;
2. decide what design patterns should be applied in order to design identified aspects;
3. design and implement aspects, document observations and findings, and collect other qualitative data;
4. perform measurements, test the code and collect quantitative data;
5. evaluate the structure of the code according to criteria;
6. analyze, generalize the collected data and evaluate hypothesis.

Summary of Research Results
The results of the thesis research can be summarized as follows:

- The hypothesis has been proven that there exist paradigm-independent design problems, at least in the context of OO and AO software engineering paradigms.
- There has been identified the subset of 23 GoF object oriented design patterns (20 GoF patterns) which solve paradigm-independent design problems and can be transformed into pure AO design patterns (GoF_{AO} patterns).
- The hypothesis has been proven that aspect-oriented constructs are sufficient to implement 20 of GoF_{AO} design patterns, with regard that 5 of them are exposed to some reduced applicability.
- The rules have been proposed how to transform 20 GoF design patterns into GoF_{AO} design patterns.
- The hypothesis has been validated that the usage of GoF_{AO} design patterns (next to 23 GoF design patterns) improves the efficiency of domain frameworks designs.
- The hypothesis has been proven that the usage of GoF_{AO} design patterns allows designing a new class of hot-spots in white-box AO domain frameworks, (namely, hot spots represented by abstract aspects).
- The hypothesis has been validated that the usage of GoF_{AO} designs patterns reduces crosscutting in AO domain frameworks.

- The hypothesis has been rejected that the development of AO domain frameworks using GoF_{AO} design patterns has no particular impact on the overall run-time performance of the applications developed using such frameworks.

Contributions of the Dissertation

Present thesis is one of the first researches that aims to investigate pure AO design patterns and the application of such patterns in the design of AO domain frameworks. Although several attempts (Arpaia, et al, 2008; Santos et al., 2007; Kulesza et al., 2006) to design customizable aspects in frameworks have been made, none of them investigates the use of pure AO design patterns to design aspects as hot spots and none of them examines the design of AO frameworks in such detail. It is also the first work that states the question about the existence of design problems which are common to all or, at least, to several software engineering paradigms. Finally, the case study methodology applied in present thesis supports the empirical research approach in which constructive research and case study research methods can be used to validate hypotheses in software engineering.

The practical significance of the thesis is as follows:

- 20 pure aspect-oriented design patterns, that have been developed in the thesis research, can be applied developing any aspect-oriented domain frameworks as well as other aspect-oriented applications;
- the thesis demonstrates how abstract aspects should be designed so that to be applicable as hot-spots in aspect-oriented domain frameworks.

Approbation

The main results of the thesis were presented and approved at the following conferences:

- 3rd International Conference on Pervasive Patterns and Applications, PATTERNS 2011, September 25-30, 2011 – Rome, Italy;
- 15th Conference of Lithuanian Computer Society "Computer Days – 2011", September 22–24, 2011, Klaipeda, Lithuania;
- 50th Conference of Lithuanian Mathematicians Society, June 18–19, 2009, Vilnius, Lithuania.
- 12th Student Scientific Society conference "Fundamental Research and Innovation in Science Integration". Klaipeda University Faculty of Natural Science and Mathematics, 2009, Klaipeda, Lithuania.

Outline of the Dissertation

The text of the thesis consists of introduction, 5 main chapters, conclusions, list of references, list of publications and appendixes. Main chapters are provided with summary and (except chapter 1) with conclusions.

Introduction describes research context and challenges, presents the problem statement, discusses motivation, aims and objectives of the research, states research questions and hypotheses, describes research design and research methods, research results, contributions of the thesis, and approbation of obtained results.

Chapter 1 presents preliminaries on design patterns, aspect-oriented paradigm and frameworks.

Chapter 2 describes the results of critical analysis of related works.

Chapter 3 develops and discusses main theoretical results of the research. It proves the hypothesis that there exist, at least in the context of OO and AO software engineering paradigms, paradigm-independent design problems, identifies the subset of 23 GoF object oriented design patterns (20 GoF patterns) which solve paradigm-independent

design problems and can be transformed into pure AO design patterns (GoF_{AO} patterns), proves the hypothesis that aspect-oriented constructs are sufficient to implement 20 of GoF design patterns, with regard that 5 of them exposes some reduced applicability, and presents the rules to transform 20 GoF design patterns into GoF_{AO} design patterns.

Chapter 4 describes in details case studies on application of the transformed design patterns to design frameworks and validation of research hypothesis. It validates the hypotheses that the usage of GoF_{AO} design patterns (next to 23 GoF design patterns) improves the efficiency of domain frameworks designs, that the usage of GoF_{AO} design patterns allows designing a new class of hot-spots in white-box AO domain frameworks, namely, hot spots represented by abstract aspects, that the usage of GoF_{AO} designs patterns reduces crosscutting in AO domain frameworks, and that the development of AO domain frameworks using GoF_{AO} design patterns has no particular impact on the overall run-time performance of the applications developed using such frameworks.

Chapter 5 discusses some open questions.

Conclusions present the main conclusions of the dissertation.

Appendixes presents preliminaries about AspectJ programming language, list of remaining transformed GoF design pattern descriptions and extended versions of several diagrams.

Conclusions of the Dissertation

1. Aspect-oriented design patterns, developed using direct code rewriting techniques and represented using constructs, provided by both, aspect-oriented and object-oriented paradigms, are not sufficient for complete separation of concerns, do not allow to implement hot spots of aspect-oriented domain frameworks as abstract aspects and are not universal enough, therefore can be applied only to a specific application context.
2. The 20 out of 23 object-oriented design patterns (GoF patterns) proposed by Gamma et al. (Gamma et al., 1994) solve the design problems that are also relevant in the context of aspect-oriented software engineering paradigm. Constructs provided by aspect-oriented paradigm are sufficient to implement these design patterns – that is, to implement them as pure aspect-oriented design patterns without using any specific object-oriented constructs such as classes and objects. It means that aspects can be used as collaborative entities, making it possible and reasonable to create hierarchies of aspects and establish dependencies and associations among aspects.
3. The usage of pure aspect-oriented designs patterns reduces crosscutting in aspect-oriented domain frameworks and allows the designing of a new kind of hot spots, namely, the hot spots represented by abstract aspects in white-box AO domain frameworks.
4. The case studies have confirmed that 20 pure aspect-oriented design patterns decreases code complexity, eliminates crosscutting and allows designing additional AO hot spots in frameworks. Performance tests have revealed that in some cases the loss of performance is expected. However, it depends on the particular design pattern that is applied and on designer skills – that is, how he/she is able to choose proper design patterns.
5. The case studies and the analysis of aspect-oriented domain framework construction process demonstrate that the following construction steps are necessary in order to achieve successful design results:

a. identify aspects representing modules that have to be designed in a crosscutting manner by analyzing requirement specification;
b. decide which hot spots have to be designed using objects and which – using aspects; examine what design problems have to be solved and determine the design patterns that can be applied for this purpose;
c. design and implement the required aspects and objects;
d. prepare necessary test cases; check whether the resulted design is already acceptable; improve the design and go back to step *c* if the refactoring of code is still required.

About the Author

Žilvinas Vaira was born in Lithuania in Klaipėda on 22 August 1981. In 1999, he finished Klaipėda "Ąžuolynas" Gymnasium. He graduated from Klaipėda University Faculty of Nature and Mathematics in 2005 acquiring Bachelor's Degree in Informatics. He gained his Master's Degree in Informatics at Klaipėda University Faculty of Nature and Mathematics in year 2007. Since 2007 to 2011 he has been a doctorate at Vilnius University Institute of Mathematics and Informatics.