LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

Pielāgojamu (White-labled) interenetā bāzētu informācijas sistēmu izstrādes aspekti

BAKALAURA DARBS

Daraba autors: Mārtiņš Rudiards Dzirnieks

Studenta apliecības Nr.: md18024

Darba vadītājs: doc. dr. dat. Ivo Odītis

RĪGA, 2022

Anotācija

Bakalaura darbā tika pētīts, kas ir atkārtoti izmantojama sistēma un kādus pienesumus šīs sistēmas izstrāde spēj dot sistēmu izstrādes procesos. Papildus tika pētīts līdz šim oficiāli neapstiprinātais, bet teorijas līmenī izvirzītais atkārtoti izmantojamās sistēmas koeficients, kas palīdz noteikt atkārtoti izmantojamas sistēmas iespējamību, un citi ieteikumi un ievērojamie principi sistēmu atkātotai izmantošanai. Darba procesā tika apskatīts arī personīgais viedoklis, kas ir gūts izstrādājot atkārtoti izmantojamu sistēmu, attiecīgi apstiprinot vai noliedzot līdz šim izvirzītās hipotēzes par atkārtoti izmantojamām sistēmām.

Atslēgas vārdi: *atkārtoti izmantojama sistēmā, izstrādes process, iespējamība, koeficients*

Abstract

ASPECTS OF DEVELOPING WHITE-LABLED INTERNET BASED INFORMATION SYSTEMS

The bachelor's thesis investigated what a reusable system is and what contributions the development of this system can make to the system exhibition processes. In addition, the coefficient of the reusable system, which has not been officially confirmed so far, but has been put forward at the theoretical level, has been studied, which helps to determine the possibility of a reusable system. The personal opinion gained during the development of the reusable system was also considered in the course of the work, respectively confirming or denying the hypotheses about reusable systems put forward so far.

Keywords: *reusable system, development process, probability, coefficient*

Saturs

Apzīmējumi

GUI – grafiskā lietotāja interface.

DOD-STD-2167A – Kvalitātes standarts ko izstrādājis Amerikas Savienoti Valstu Aizsardzības departaments.

IT – informāciju tehnoloģijas.

IEEE – kvalitātes standarts.

IPR – Intelektuālais īpašums.

DSM – Dizaina struktūru matricas.

DC – Tiešais savienojums.

VFI - Redzamā izplūšana.

VFO – Redzamā ieplūšana.

GNU, GPL – Vispārējā publiskā licence.

BSD – Atļauto bezmaksas programatūru licenču saime, kas uzliek minimālus ierobežojumus aptvertās programmatūras izmantošanai un izplatīšanai.

Css – Stila lapas kaskadēšana ir īpašs stila lapas valoda, ko lieto, lai aprakstītu izskatu iezīmēšanas valodā veidotiem dokumentiem.

Api – Lietojumprogrammas saskarne.

SQA/IV&V/ - programmatūras kvalitātes nodrošināšanas un neatkarīgas validācijas un verifikācijas sertifikāts.

Ievads

Viena no jomām kas visvairāk attīstās, mūsdienu pasaulē, ir informācijas tehnoloģijas, jeb IT, kā rezultātā tiek izdomāti jauni veidi kā optimizēt izmatotos līdzekļus, lai pēc iespējas ātrāk un vairāk spētu piedāvāt klientam nepieciešamo produktu.

Kā jebkurā projektā, tai skaitā informācijas tehnoloģiju sistēmas izstādes procesā ir divi noteicošie faktori vai ir iespējams izstādāt šo projektu, tās ir finanses un laiks. Viens no primitīvākajiem veidiem kā šos abus resursus varētu ieekonomēt ir pārkopēt jau esošu projektu, kurš ir darba kārtībā, un palaist viņa darbību reālajā laikā, kā rezultātā nebūtu jātērē finansējums darba spēkam, jeb programmētājiem un tas patērētu krietni vien mazāk laika, nekā izstrādājot šo projektu no sākuma.

Attiecīgi tiek ieviests jauns jēdziens, kā atkāroti izmatojama sistēma (reusability). Jēdziens pats par sevi jau izsaka daudz, ka viena sistēma tiks izmantota daudz reizes, kas ir tieši tas, kas varētu palīdzēt ar iepriekš minēto problēmu. Atkārtoti izmantojama sistēma ir produkts, kuram tiek izmatota daļa funkcionalitāte, tā tiek adaptēta un tad to licencē.

Par cik šī ideja par atkārtoti izmantojamām sistēmām ir salīdzinoši jauna, daudzas līdz šim esošas sistēmas ir izstrādātas tieši viena projekta vajadzībām, kā rezultātā rodas jautājums vai šīs sistēmas ir iespējams pārstādāt un izveidot par atkārtoti izmantojamām sistēmā, vai tomēr finansiāli izdevīgāk ir veidot sistēmu no jauna ar domu, ka šī sistēma būs atkāroti izmantojama. Šajā gadījumā būt ļoti noderīgi, ja būtu koeficients ar kura palīdzību varētu noteikt līdz šim esošās sistēmas atkārtojamības iespējamību. Taču līdz šim nav vēl izveidota viennozīmīga formula pēc kuras varētu noteikt šo koeficientu. Tomēr hipotēzes līmenī ir atrasti vairāki mainīgie kas šo formulu varētu veidot.

Līdz šim atkārtotas izmantošanas iespējamības formulā ieviestie mainīgie ir: dokumentācija, atbilstība standartiem, atbalsts, licencēšana, iepakojums, pārnesamība kā arī divi no svarīgākajiem mainīgajiem testēšana un modularitāte. Katram no šiem mainīgajiem ir sīkāks sadalījums ar paskaidrojumiem, kas palīdzēs noteikt mainīgā vērtību.

Bakalaura darba pētījuma mērķis ir apkopot un izvērtēt līdz šim zināmo atkārtotas izmantojamas sistēmas iespējamības noteikšanas kritērijus, ņemot vērā savu pieredzi ar atkārtoti izmantojamām sistēmām.

Darba pirmajā daļā tiks apskatīta teorētiskā daļa, kas ir atkārtoti izmantojamas sistēmas, no kurienes šis jēdziens ir radies. Darba turpinājumā tiks apskatīta formula, kas izsaka atkārtojamības iespējamību, kā arī teorētiskā līmenī, kā šādas sistēma būt jābūvē. Visbeidzot tiks analizēta reāla atkārtoti izmantojama sistēma, kā arī tiks aprēķināts šīs sistēmas atkārtojamības koeficients. Kā rezultātā varēs secināt, vai atkārtoti izmantojamas sistēmas koeficients nodrošina nepieciešamo precizitāti un vai tās sasniedz nepieciešamos rezultātus.

1. Pielāgojama jeb atkārototi izmantojamas informācijas sistēma

Pielāgojamas informācijas sistēmai jeb *White labled* nav viennozīmīgā definīcija. Attiecīgi šajā nodaļā autors ir apkopojis dažādus iespējamos pielāgojamas sistēmas aprakstus un mēģinājis veidot darbam atbilstošu šī termina skaidrojumu.

Apskatot pielāgojamas informācijas sistēmas var secināt, ka tā ir sistēma, kas tiek izgatavota ar sākotnēju nodomu, radīt daudz dažādas sistēmas, kas atšķiras gan funkcionāli, gan ar savu dizainu. Karkass visām šīm sistēmām ir vienāds, bet tajā pašā laikā katrai sistēmai ir pievienots vai tieši pretēji noņemts, kas tāds kas citām sistēmām nav.

Līdz 1988. gadam nebija vienots sistēmu izstrādes protokols, kas noveda pie daudzu sarežģītu un grūti kontrolējamu sistēmu veidošanu, ar ko nebija mierā Amerikas Savienoto Valstu Aizsardzības departaments. 1988. gadā tika izstrādāts šis protokolos ar nosaukumu DOD-STD-2167A. DOD-STD-2167A protokols definēja doto dzīves cikla modeli, kā arī apraksta atkārtotu izmantošanu katram sistēmas izveides etapam. Šie etapi ir: sistēmas prasību analīze, sākotnējā projektēšana, sistēmas koncepciju izstrāde, programmatūras prasību analīze, detalizēts dizains, vienību testēšana, programmatūras komponentu integrācija un testēšana, kodēšana, kā arī sistēmas integrācija un testēšana.

Ņemot vērā šo protokolu mūsdienās mēģina apkopot šī protokola būtību un nodefinēt precīzi, kas ir atkārtoti izmantojama sistēma. Līdz šim precīzākais apraksts ir šī definīcija:

Atkārtota izmantošana ir darbība, kurā tiek izveidots risinājums, pamatojoties uz iepriekš jau zināmiem problēmu risinājumiem. Atkārtotas izmantošanas jēdziens ir sadalīts sešos daļās, katrā no šīm daļām ir definētas darbības, kas ir jāizdara gatavojoties nākamajai daļai. Šīs darbības ir [1]:

1. Tiek izveidots plāns vai stratēģija izvērtējot problēmas un to iepriekšzināmās problēmas risinājumus.
2. Tiek izveidota stratēģija problēmas risinājuma struktūras noteikšanai.
3. Risinājuma optimizācija un uzlabošana iepriekš definētajām komponentēm, kas ir nepieciešami nākamajā fāzē.
4. Esošo komponentu iegūšana un modificēšana.

5. Komponentu integrēšana esošajā posmā.
6. Produktu novērtēšana.

2. Atkārtotas izmantojamības vērtējumi

2.1. Atkārtoti izmantojamas sistēmas koeficients

Daži no autoriem norāda ka ir formula, kas palīdz noteikt atkārtotas izmantošanas iespējamību, tā ir "Atkārtotas izmantošanas indekss" formula [6]. Šīs formulas pamatā tiek izmantota svaru summas formula un līmeņi, kas noteikti definīcijām.

2.1.att. Atkārtotas izmantošanas indekss

$$RF = \sum_{i=1}^{i=8} Pi * Wi$$

- Pi: dokumentācija, atbalsts jeb kontaktinformācija, uzstādīšana, modularitāte un sarežģītība, testēšana, atkārtotas izmantošanas tiesības, licencēšana, paplašināšana.
- Wi ir norādītais līmenis parametram atkārtotas izmantošanas sistēmā.

Tabula ir apkopoti visi noteiktie līmeņi, to diapazona un tā ietekme attiecībā uz atkārtoti izmantojamām sistēmām, skatīt 2.1. tabulu:

2.1. tabula

Nr.	Parametrs	Parametra nosaukums	Vērtību diapazons	Ietekme
1	P1	Atbalsts un kontaktinformācija	No 1 līdz 3	1
2	P2	Dokumentācija	No 1 līdz 3	2
3	P3	Modularitāte un sarežģītība	No 1 līdz 3	3
4	P4	Uzstādīšana un iepakošana	No 1 līdz 3	4

5	P5	Atkārtotas izmantošanas tiesības	No 1 līdz 3	5
6	P6	Testēšana	No 1 līdz 7	6
7	P7	Sertifikācija	No 0 līdz 2	7
8	P8	Paplašināšana	No 1 līdz 3	8

Lai tiktu aprēķināts atkārtotas iespējamības koeficients, šiem mainīgajiem ir jābūt gana vispārīgiem, lai nebūtu pārāk sadrumstalots vienādojums, bet tajā pašā laikā gana pašpietiekamiem, kas spēj katrā mainīgajā ietvert nepieciešamo informāciju. Nākošajās tabulās tiks aprakstīts katrs no tabulā iekļautajiem mainīgajiem, kā arī to ietekme atkarībā no mainīgā izpildes.

P1 – Atbalsts un kontaktinformācija. Kā redzams tabulā mainīgajam ar vismazāko ietekmi uz atkārtotas iespējamības koeficientu ir "minimāls atbalsts", kas iekļauj sevī to, ka ir pieejama programmatūra, nav iespējams sazināties ar sistēmas izstrādātāju. Soli augstāks līmenis ir, ja sistēma ir mantota un ir pieejama saziņa ar sistēmas izstrādātājiem caur e-pastu vai sazvanoties. Tomēr vislielāko devumu dod pēdējais līmenis jeb "vispārējs atbalsts", ka sistēma ir mantota, kā arī nepieciešamības gadījumā ir iespējama saziņa gan elektroniski, gan ja ir radusies tāda vajadzība, tad arī klātienē, kas nodrošina sistēmas maksimālu adaptāciju. Visi šie parametra līmeņi ir norādīti 2.2. tabulā.

2.2. tabula

	Līmenis	Skaidrojums	Piešķirtais līmenis
a.	Minimāls atbalsts	Nodrošina pamatfunkcijas neiesaistoties tālāk	1
b.	Virtuāls atbalsts	Atbalsts caur e-pastu un telefonu	2
c.	Vispārējs atbalsts	Fiziska klātbūtne un skaidrojumi	3

P2 – Dokumentācija. Kā redzams tabulā mainīgajam ar vismazāko ietekmi uz atkārtotas iespējamības koeficientu ir "minimāla dokumentācija". Nekvalitatīva dokumentācija spēj aizkavēt sistēmas ieviešanu vairākas stundas vai pat dienas. Parasti šāds iznākums ir paredzams, ja dokumentācija tiek rakstīta pēc sistēmas izstādes, kā rezultātā netiek pieminētas daudzas lietas. Soli augstāks līmenis ir, ka dokumentāciju raksta paralēli sistēmas izstādei, līdz ar ko tiek iekļauta visa nepieciešamā informācija, bet netiek ievēroti dokumentācijas standarti, kā rezultātā klientam ir krietni vien grūtāk orientēties dokumentā, kas noved pie lieka laika patēriņa. Visbeidzot, dokumentācija ar vislielāko ietekmi uz atkārtoti izmantojamu sistēmas koeficientu ir dokumentācijas, kas tiek rakstīta saskaņā ar IEEE formātu un tiek rakstīta paralēli sistēmas izstrādei. Sistēmas dokumentācija ir viegli uztverama. Visi šie parametra līmeņi ir norādīti 2.3. tabulā.

2.3. tabula

	Līmenis	Skaidrojums	Piešķirtais līmenis
a.	Minimāla dokumentācija	Trūcīga un nav pārskatīta dokumentācija	1
b.	Noderīga dokumentācija	Pārskatīta un kontrolēta dokumentācija, bet nav ievērots dokumentācijas standarts un izmantojot prasa piepūli	2
c.	Pilnīga dokumentācija	Pilnīga dokumentācija saskaņā ar IEEE formātu	3

P3 – Modularitāte un sarežģītībā. Ir divi modularitātes veidi: arhitektūras jeb dizaina modularitāte un funkcionālā modularitāte. Daudzi uzskata ka modularitāte ir viens no galvenajiem atslēgas vārdiem, lai sistēma būtu atkārtoti izmantojama. Tāpēc sistēmas ar ļoti sarežģītiem un grūti izsekojamiem starp moduļiem tiek vērtētas ar viszemāko vērtējumu. Attiecīgi sistēmas ar viegli izsekojamu dizaina modularitāti un grūti izsekojamu funkcionālu modularitāti tiek vērtētas augstāk. Protams, sistēmas ar labi izsekojamu dizaina jeb arhitektūras modularitāti un funkcionālo modularitāti tiek vērtētas visaugstāk. Visi šie parametra līmeņi ir norādīti 2.4. tabulā.

2.4. tabula

	Līmenis	Skaidrojums	Piešķirtais līmenis
a.	Ļoti nestrukturēts	Sarežģīta starpmoduļu savienošana. Sarežģīta intramoduļu savienošana.	1
b.	Virtuāls atbalsts	Sarežģīta starpmoduļu savienošana. Vieglāka intramoduļu savienošana.	2
c.	Vispārējs atbalsts	Viegla starpmoduļu savienošana. Viegla intramoduļu savienošana.	3

P4 – Uzstādīšana un iepakošana. Kā redzams tabulā mainīgajam ar vismazāko ietekmi uz atkārtotas iespējamības koeficientu ir “manuāla instalēšana”. Manuāla instalēšana tiek uztverta, kā avota koda kopēšana un pievienošana produktam, kā arī bibliotēkas kas tiek izmantotas kodā, klientam ir jākonfigurē manuāli. Attiecīgi, iespējamība ka klients kļūdīsies ir kādā no šiem

posmiem ir visai liela. Kā rezultātā tiek patērēts nevajadzīgi daudz laiks, lai palaistu projekta darbību. Soli augstāk vērtējama uzstādīšana ir "daļēji automātiska instalēšana". Ir dokumentēti visi soļi kā pievienot kodu un bibliotēkas, tomēr ir jāveic vairākas instalācijas lai sistēma tiktu palaista. Visbeidzot uzstādīšana ar vislielāko ietekmi uz atkārtoti izmantojamu sistēmas koeficientu ir "automātiska instalēšana". Šajā līmenī viss atrodas vienā iepakojumā, kurā ir iebūvēta konfigurācija, kā rezultātā sistēmas palaišanai ir nepieciešama neliela piepūle. Visi šie parametra līmeņi ir norādīti 2.5. tabulā.

2.5. tabula

	Līmenis	Skaidrojums	Piešķirtais līmenis
a.	Manuāla instalēšana	Avota kods ir ievietots manuāli, bibliotēkas ir saistītas manuāli un tās atkarības ir konfigurētas manuāli.	1
b.	Daļēji automātiska instalēšana	Norādījumi lietošanai un atkarības ir dokumentētas soli pa solim, procedūras var izpildīt bez lielas piepūles.	2
c.	Automātiska instalēšana	Atkārtoti lietojams modulis kā instalācijas iepakojums ar iebūvētu konfigurāciju un moduļi viegli adaptējas ar atbalstīto dokumentāciju.	3

P5 – Atkārtotas izmantošanas tiesības. Vissliktākais gadījums ir, kad nav minēts kam ir definētas tiesības uz atkārtotu izmantošanu, kā iespēja ka radīsies domstarpības. Nākamais līmenis ir, kad tiek definētas tiesības tikai izstrādātāja organizācijā. Visbeidzot, visaugstāk novērtētais līmenis ir, kad tiek definētas atkārotas izmantošanas un IPR tiesības arī ārpus organizācijas. Visi šie parametra līmeņi ir norādīti 2.6. tabulā.

2.6. tabula

	Līmenis	Skaidrojums	Piešķirtais līmenis
a.	Nenoteikts	Atkārtotas izmantošanas aspekti un tiesības nav definētas.	1
b.	Iekšēji definēts	Atkārtota izmantošana un IPR tiesības ir adresētas lietošanas organizēšanas ietvaros dažādās entītijās.	2
c.	Ārēji definēts	Atkārtota izmantošana un IPR tiesības ir adresētas lietošanai ārpus organizācijas.	3

P6 – Testēšana. Testēšana tiek pieminēts kā otrs īpaši svarīgs mainīgais atkārtoti izmantojamas sistēmas koeficientam. Par cik sistēmu ir iespējams testēt daudz un dažādos veidos, tiek izdalīti 7 dažādi līmeņi. Attiecīgi visliktākais testēšanas veids ir "prototipēšana". Līdz šim sistēma tiek izstrādāta kā prototips, kā rezultātā nav veikta nekāda veida testēšana. Nākamais līmenis ir "funkcionālā testēšana". Šajā gadījumā sistēmai ir izveidots plāns, par to

kādi varētu būt sagaidāmie rezultāti un attiecīgie testa plāni. Nākamais līmenis “statiskā testēšana”. Šajā gadījumā sistēmas kods tiek pakļauts statiskās testēšanas rīkiem. Nākamais līmenis “inspekcija”. Šajā gadījumā kods ir izturējis līdz šim esošos testēšanas veidus, un tiek nodots trešajai personai, kas pārbauda funkcionalitāti, attiecīgi tiek saņemts ziņojums ar kļūdām , kas radušās testēšanas laikā . Nākamais solis “vienībtestēšana”. Šajā solī tiek padziļināti pārbaudīta visa sistēmas darbība un tieši tā pat kā iepriekš iesniegts ziņojums ar kļūdām, kas atklātas testēšanas laikā. Nākamais solis “performances testēšana”. Šajā testēšanas solī tiek pārbaudīta sistēmas ātrdarbība, un attiecīgi, ja sistēma darbojas tā pat, kā norādīts dokumentācijā, tests ir iziets. Visbeidzot, pēdējais līmenis “formālā testēšana”. Šajā solī tiek pārbaudīts un testēts vai tiek izmantotas sertificētas metodes. Visi šie parametra līmeņi ir norādīti 2.7. tabulā.

2.7. tabula

	Līmenis	Skaidrojums	Piešķirtais līmenis
a.	Prototipēšana	Lietojumprogramma ir izstrādāta kā prototips bez pārbaudes dokumentācijas	1
b.	Funkcionālā testēšana	Funkcionālās pārbaudes dokuments pieejams ar pārbaudes plānu un testa gadījumiem un to rezultātiem.	2
c.	Statiskā testēšana	Kods ir izgājis cauri statiskās testēšanas rīkiem.	3
d.	Inspekcija	Trešā puse ir izgājusi cauri un pārbaudījusi funkcionālās	4

		pārbaudes un iesniegusi ziņojumu.	
e.	Vienībtestēšana	Sistemātiskās vienību pārbaudes dokumentācija ir pieejama ar izpildes rezultātiem.	5
f.	Performances testēšana	Darbības laika izpilde un laiks ir apstiprināts tieši tāds pats kā dokumentēts sniegumu specifikācija.	6
g.	Formālā testēšana	Modernas formālās testēšanas metodes pēc modeļa pārbaudes vai tiek pielietota teorēmas pierādīšana kas ir akceptēta un sertificēta.	7

P7- Sertifikācija. Vissliktākais gadījums, protams, ir, kad sistēmai nav iziet sertifikācija. Attiecīgi nākamais solis ir, kad sistēma ir iesniegta sertifikācijas pārbaudes iestādēs un ir saņemts ziņojums par uzlabojumiem sistēmā. Attiecīgi, visaugstāk novērtētais līmenis ir, kad sistēma ir izturējusi sertifikācijas pārbaudes un ir atzīta par derīgu. Visi šie parametra līmeņi ir norādīti 2.8. tabulā.

2.8. tabula

	Līmenis	Skaidrojums	Piešķirtais līmenis

a.	Nav sertificēts	Programma nav SQA/IV&V/ sertifikācija.	0
b.	Iekšēji sertificēts	Visi posmi tiek pārbaudīti SQA/IV&V/ aģentūrā. Pamatojoties uz šo sertifikātu aģentūra izsniedz precizējumus un detaļas par procesiem.	1
c.	Ārēji definēts	Visi posmi tiek pārbaudīti ar ārēju sertifikācijas aģentūru koordināciju. Sertifikātu izsniedz aģentūra precizējot detaļas par procesiem.	2

P8 – Paplašināšana. Kā redzams tabulā mainīgajam ar vismazāko ietekmi uz atkārtotas iespējamības koeficientu ir "pārnesamība nav definēta". Kā pēc līmeņa virsraksta var noprast, tad šajā gadījumā nav veikti izpētes darbi par to kā sistēmu varētu pārnest uz citām ierīcēm un platformām, attiecīgi ir risks, ka sistēma varētu neiet uz kādas platformas. Soli augstāks līmenis ir, kad sistēmas pārnesamība ir apzināta un tiek dokumentēts uz kādām platformām vai platformu grupām šī sistēma stādā. Visbeidzot ar visaugstāko līmeni tiek definētas tādas sistēmas, kurām ir veikta sistēmas pārnesamības izpēte, sistēmas ir paredzēta vairākām platformām. Visi šie parametra līmeņi ir norādīti 2.9. tabulā

2.9. tabula

	Līmenis	Skaidrojums	Piešķirtais līmenis

a.	Pārnesamība nav definēta	Pārnesamība nav pētīta un nav saderīga atlase ar pārnesamības kodēšanas standartiem. Manuāla piepūle var palīdzēt atkārtoti izmantot.	1
b.	Specifiska pārnesamība	Programmatūra ir pārnēsājama uz specifiskām platformām vai platformas grupām.	2
c.	Vispārīga pārnesamība	Programmatūra ir sākotnēji paredzēta vairākām platformām un tiek veidotas atkarības ar konfigurācijas detaļām.	3

2.2. Arhitektūras modularitāte

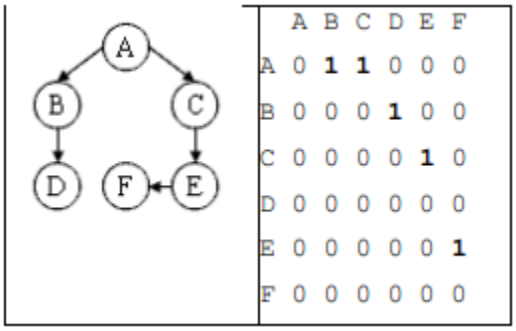
Pētījumi, kas pēta arhitektūras struktūras ir ienākusi programmatūras jomā. Tēmai ir īpaša nozīme kā tiek izstrādāta programmatūra. Reti programmatūras projekti sākas no nulles. Tā vietā, iepriekšējā versija tiek izmantota kā platforma, uz kuras tiek veidota jauna funkcionalitāte.

Oficiālā programmatūras modularitātes izpēte sākās ar Parnas (1972), kurš ierosināja konceptu par informācijas slēpšanu kā mehānismu koda sadalīšanai moduļu vienībās. Šis prasīja projektētājiem atdalīt moduļa iekšējās detaļas no ārējām saskarnēm, samazinot koordinācijas izmaksas, kas saistītas ar sistēmas izstrādi un veicinot izmaiņas moduļiem, neietekmējot citas

konstrukcijas daļas. Centieni izmērīt programmatūras modularitāti parasti koncentrējas uz tveršanas savienojuma līmeni starp dažādām konstrukcijas daļām.

Šos kritērijus var izpildīt, izmantojot dizaina struktūras matricas tehniku (DSM), lai analizētu saistību starp modularitāti un dizaina attīstību. DSM nodrošina veidu, kā analizēt un izmērīt raksturlielumus dizaina komponentu līmenī. Lai novērtētu komponentu modularitātes ietekmi uz dizaina attīstību, ir izstrādāti divi rādītāji, kas nosaka, cik lielā mērā komponenti ir savienoti viens ar otru. Pirmkārt, tiek novērtēts komponentei piemītošo tiešo atkarību skaits jeb tiešais savienojums (*Direct Connectivity*). Otrkārt, tiek novērtēts gan tiešo un netiešo atkarību skaits, kas piemīt komponentam jeb redzamība (*Visibility*). Abos gadījumos tiek izmantoti atsevišķi pasākumi atkarībām, kas ieplūst komponentā no tām (sauktas par "Fan-In"), kas izplūst no tā (saukta par "Fan-Out"), atspoguļojot atkarības attiecību asimetrisko raksturu. 2. attēlā var redzēt, ka elements A ir atkarīgs no elementiem (vai “izsauc funkcijas iekšienē”) B un C, tāpēc izmaiņas elementā C var tieši ietekmēt elementu A. Savukārt elements C ir atkarīgs no elementa E, tāpēc izmaiņas elementā E var tieši ietekmēt uz elementu C un netieša ietekme uz elementu A ar “ceļa garumu” divi. Līdzīgi, izmaiņas elementā F var tieši ietekmēt elementu E un netieši ietekmēt elementus C un A ar “ceļa garumiem” attiecīgi divi un trīs. Nav netiešu atkarību starp elementiem, ja ceļa garums ir četri vai vairāk.

2.2. att. Dizaina struktūru matrica



Tiešā savienojuma (DC) mēri ir iegūti tieši no DSM. Piemēram, elementam A ir DC *Fan-Out* no diviem, ņemot vērā, ka tas ir atkarīgs no elementiem B un C; un tam ir nulles DC *Fan-In* ieeja, ņemot vērā, ka no tā nav atkarīgi nekādi elementi. Lai identificētu Katra elementa redzamību, tiek izmantota matricas reizināšanas tehnika. It īpaši reizinot DSM ar secīgiem n pakāpēm, mēs iegūstam tiešās un netiešās atkarības, kas pastāv katram nākamajam ceļa garumam n. Šo matricu summēšanas rezultātā iegūst redzamības matricu V, kas parāda tiešās un netiešās

atkarības starp elementi visiem iespējamiem ceļa garumiem līdz maksimālajam, ko nosaka pēc izmēra. Kas ir parādīts 2.3. attēlā.

2.3. att. Dizaina struktūru summu matricas

Redzamības rādītāji tiek iegūti tieši no redzamības matricas. Redzamības izplūšana jeb *Visibility Fan-Out* (VFO) tiek iegūta, summējot pa rindām. Piemēram, elementam A ir VFO no sešiem, kas nozīmē, ka tas tieši vai netieši ir atkarīgs no visiem pārējiem elementiem. Redzamības ieplūšana jeb *Visibility Fan-In* (VFI) iegūst, summējot kolonnas. Piemēram, elementam A ir tikai viens VFI. tātad tas ir redzams tikai sev. Salīdzinājumam VFO un VFI var izteikt procentos no elementu skaita sistēmā.

Ņemot vērā, ka šie divi mēri attēlo kontinuuma(*continuum*) pretējos galus, pa kuru var izmērīt savienojuma līmeņus. Pirmajā tiek fiksētas tikai tiešās saites starp sastāvdaļām. Otrais aptver visas tiešās un netiešās saites starp komponentiem, piešķirot tiem vienādu svaru neatkarīgi no ceļa garuma. Šajā gadījumā plāns izgaismo dažādus iespējamos pasākumus, kas atrodas starp šīm divām galējībām.

Kad attiecīgie parametri un atkarības ir definētas, tad ir iespējams izveidot moduļu prototipus, ar kuru palīdzību ir iespējams izveidot moduļus. Šie moduļi palīdz tālāk izveidot jēgpilnu modulāru sistēmu. Tiesa gan jāpien, ka šiem moduļiem nedrīkst būt tiešas atkarības vienam ar otru. Lai to panāktu ir jānodefinē abstrakts moduļa interfeiss. Attiecīgi šādā veidā mainot kādu no jau pastāvošajiem moduļiem netiek ietekmēts cits modulis, bet gan tikai šis abstraktais interfeiss, kas padara sistēmu modulāru.

2.3. Funkcionālā modularitāte

Programmatūras arhitektūras disciplīna, kas galvenokārt attīstījusies pēdējo 20 gadu laikā, tiek uzskatīta par pamatu veiksmīgai programmatūras produktu izstrādei. Saskaņā ar Bass, Clements un Kazman [11] programmatūras arhitektūra:

- veido kopīgu valodu ieinteresētajām pusēm – arhitektiem, projekta vadītājiem, programmētājiem, konsultantiem, klientiem, mārketinga nodaļa;
- tiek definēti lēmumi programmatūras izstrādes sākumposmā, kuri tiek ieviesti vai tieši pretēji noņemti no implementācijas saraksta un turpmāk tiek izmantots kā atsauce projekta pārvaldībai;
- sastāv no vispārinātām konstrukcijām, kuras var pārnest un atkārtoti izmantot citās produktu līnijas.

Arhitektūras viedokļi programmatūras produktos sniedz vadlīnijas, lai vienoti aprakstītu kopējo sistēmu un tās apakšsistēmas, kā arhitektūrai vajadzētu risinātu sistēmas ieinteresēto pušu šaubas, var būt viedoklis tiek saprasts kā rāmis, kas identificē modelēšanas metodes, kuras vajadzētu izmantot aprakstot arhitektūru, lai risinātu noteiktu apakškopas problēmu. Pēc Broy et al. jēdzienu kategorizācijas. [10], arhitektūra tiek iedalīta trīs abstrakcijas slāņos:

- Funkcionālā arhitektūra apraksta sistēmu ar tās funkcionālo uzvedību un novērojamām mijiedarbībām vidē.
- Loģiskā arhitektūra apraksta sistēmas iekšējo struktūru, tiek attēlots kā loģisks komponentu tīkls, kas īsteno sistēmas funkcionalitāti.
- Tehniskā arhitektūra apraksta, kā sistēma tiek realizēta, tās programmatūra uzvedība un tās aparatūras struktūru.

Saskaņā ar Van Vliet [10], programmatūras arhitektūras projektēšanas fāzē produkts tiek ievietots produkta dzīves ciklā starp prasību izstrādes fāzi un projektēšanas fāzi. Arhitektūras projektēšanas posmā arhitektūras skati ir izstrādāti un attiecīgie dizaina lēmumi tiek pieņemti attiecībā uz visām ieinteresēto pušu bažām un interesēm [10]. Tādējādi prasības tiek uzskatītas par pieņemamām un attiecīgi arhitektūras projektēšanas fāzei tiek meklēta metode prasību

uztveršanai programmatūras produkta arhitektūrā. Lai gan daudzas labi zināmas metodes ir attīstījušās zemā līmenī (piemēram, lietošanas gadījumu diagrammas), tomēr nav formāls veids, kā modelēt funkcionālo arhitektūru augstā līmenī, lai tā varētu ietver visas prasības, kas attiecas uz produkta funkcijām, un to var efektīvi atainot projekta vadītājam, arhitektiem, netehniskajām ieinteresētajām personām (konsultantiem, mārketingam, klientiem, galalietotājiem).

3. Atkārtoti izmantojamu sistēmu izstrāde

3.1. Atvērtais kods un atkārtoti izmantojama sistēmas

Varētu šķist, ka atvērtais kods jeb *Open Source* kods ir tas pats, kas atkārtoti izmantojama sistēmas kods, jo kods tiek izmantots vairākos projektos, kas ir atkārtoti izmantojamas sistēmas koda ideoloģijas pamatā, tomēr tā nav. Lai labāk izceltu atšķirības starp atkārtoti izmantojamas sistēmas kodu un atvērto kodu, apskatīsim atvērtā koda definīciju, kas sastāv no 10 punktiem.

1. Bezmaksas pārdale

   Atvērtā koda licencei nav tiesības ierobežot kādu no pusēm, kas vēlas atdot vai pārdot programmatūras kodu, vai tās fragmentu. Kā rezultātā licences īpašnieks neiegūst finansiālu labumu.

2. Pirmkods

   Kodam ir jāsatur pirmkods kompilētā veidā. Ja atvērtais kods nesatur pirmkodu, tad attiecīgi ir jābūt norādēm, kā šo pirmkodu ir iespējams lejupielādēt bez maksas. Ir aizliegts slēpt pirmkodu no lietotāja. Avota kodam ir jābūt izstrādātam programmētājam nepieciešamajā valodā, attiecīgi programmētājs veicot koda manipulācijas nesaskaras ar lieliem sarežģījumiem.

3. Atvasinājums

   Licence nedrīkst ierobežot koda atvasinājumus, kas ietver sevī arī tālāku izplatību. Uz koda atvasinājumiem attiecas tieši tādi paši noteikumi, kā uz orģinālo kodu.

4. Autora pirmkoda integritāte

   Licence aizliedz koda izplatību tikai tādā gadījumā, ja tiek izplatīts pirmkods ar "patch files".

5. Nekādas personu vai grupu diskriminācijas

Licencē noteiktajos noteikumos nedrīkst diskriminēt vai aizskart personas vai peronu grupas.

6. Nekādas diskriminācijas pret darbības jomām

   Licence nedrīkst ierobežot izmantot programmas kodu kādā noteiktā darbības jomā.

7. Licences izplatīšana

   Programmas koda licencēšana attiecas uz visām iesaistītajām pusēm, kas izmanto atvērto kodu, neveicot jaunas licences iegādi.

8. Licence nedrīkst būt specifiska produktam

   Ja programmas kods vai programma tiek izplatīta un tiek modificēta, un tas tiek darīts saskaņā ar izplatīšanas noteikumiem, tad sākotnēja koda vai programmas licences noteikumi attiecas arī uz jaunizveidoto programmu un kodu.

9. Licence nedrīkst piesārņot citu kodu.

   Licence nedrīkst uzstāt, ka visas programmas vai kodi, kas tiek izplatīti tālāk, šim kodam obligāti ir jābūt atvērtajam pirmkodam.

10. Licenču piemēri

    GNU , BSD, GPL ir licences, kas atbilst atvērtā pirmkoda definīcijai.

Tātad pēc šim definīciju apskates gan atkārtoti izmantojamām sistēmām, gan atvērtajam pirmkodam, varam secināt, ka atvērtais kods lielākoties ir kādas noteiktas problēmas risinājums, attiecīgi to var adaptēt un pievienot savai sistēmai vai programmai. Savukārt, atkārtoti izmantojama sistēma ir sistēma, kas tiek būvēta no nulles. Tas neizslēdz, ka atkārtoti izmantojamā sistēmā nevar būt atvērtais kods.

3.2. Atkārtotas izmantošanas netehniskie faktori

Kā jau iepriekš tika minēts, sistēmas izstrādē divi no galvenajiem faktoriem, lai sistēma būtu rentabla, ir laiks un nauda. Tātad lai varētu izvērtēt šos divus faktorus, uzņēmumam ir jāizstrādā plāns, kurā tiek izanalizēti ieguvumi un izmaksas. Ir veikti pētījumi par to cik reizes sistēmu ir nepieciešams atkārtoti izmantot, lai spētu atmaksāties. Līdz šim pētnieki ir teikuši, ka sistēma ir atkārtoti jāizmanto vismaz desmit reižu. Ja sistēmu paredzēts izmantot vismaz desmit reižu, tad attiecīgie ieguvumi šīs sistēmas izstrādē ir uzlabota kvalitāte programmatūras produktam, jo laika gaitā tiek izskausti visi esošie programmatūras mīnusi. Nākamais ieguvums ir savlaicīgums programmatūras procesu izstrādē, jo vairāk nav nepieciešams pievērst pastiprinātu uzmanību līdz šim atrisinātam risinājumam. Visbeidzot, programmatūras izstrādes procesa kopējais efektivitātes pieaugums.[7]

Lai sistēma veiksmīgi attīstītos, būtu jāievēro daži principi, tomēr līdz šim nav definēti šie principi gana padziļināti. Lai programmatūra tiktu līdzi laikam, sistēmai ir jārisina gan netehniski, gan tehniski jautājumi. Netehniskie faktori ir:

Ekonomika. Koda komponentu ievietošana un klasificēšana atkārtoti izmantojamās bibliotēkās ir tā joma, kas prasa daudz darba stundas, kā rezultātā ir iespējams samazināt izmaksas atkārtoti taisot sistēmas, izmantot jau gatavās bibliotēkas.

Organizatoriskie jautājumi. Lai pārdotu šīs atkārtoti izmantojamas sistēmas, ir jāveic dziļa izpēte par to vai uzņēmumam, kuram grasās pārdot šo sistēmu, tik tiešām šī sistēma derēs. Attiecīgi ir jāveic izpēte par klienta biznesa prasībām un to vajadzībām.

Vadība. Kā jau iepriekš minēts, sistēma spēs atmaksāties, tikai sākot no desmitās sistēmas, kas tiks palaista apgrozībā, līdz ar ko uzņēmuma vadībai ir jāpieņem lēmums, vai sistēmas izstrāde būs rentabla. Ja uzņēmuma vadība pieņem lēmumu, taisīt šāda veida sistēmu, visticamāk būs nepieciešams pārstrukturēt finansēšanas un pārvaldības struktūras.

Izglītības jautājumi. Par cik mūsdienās informāciju sistēmas virzās straujiem soļiem uz priekšu, ir nepieciešams apgūt jaunākās zināšanas arī pieredzējušiem profesionāļiem, kā rezultātā ir jāveic dažādas apmācības, lai sistēmā būtu iekļautas jaunākās tendences, kas

spētu nodrošināt sistēmas darbību jaunākajās ierīcēs un sistēmas koda pārnešana uz jaunāku versiju būtu jāatliek uz pēc iespējas ilgāku laiku.

Psiholoģiskās problēmas. Viena no galvenajām psiholoģijas problēmām sistēmas izstādes vidē ir, ka programmētāji neuzticas citu kolēģu rakstītajam kodam. Lai gan koncepts par atkārtoti izmantojamām sistēmām ir gana jauns un skatoties pēc līdzšinējām pieredzēm sistēmas izstrādātājiem patīk ieviest jaunus konceptus sistēmas izstrādē, tomēr sistēmas izstrādātāji vēl nav līdz galam gatavi šim jaunajam konceptam.

Juridiskās problēmas. Viens no visvairāk sasāpējušiem jautājumiem juridiskajā jomā ir par līdzekļu atgūšanu, ja sistēmu neizdodas veiksmīgi pielietot.

Mērīšana. Par cik sistēmas programmēšana skaitās inženiertehniskā joma, tad pierasts, ka visu izsaka skaitļos. Sistemātiski atkārtoti izmantojamām sistēmām šie koeficienti ir produktivitāte, ko var mērīt laika vienībās un kvalitāte, ko var mērīt ar pieļautajām kļūdām skaitliski, jeb sistēmas darbību cik daudz procentuāli sistēmas darbība ir nodrošināta.

Krātuves. Kad tiek iegādāta atkārtoti izmantojama sistēma, klientam ir jābūt pieejai pie sistēmas bibliotēkām. Attiecīgi jābūt iespējai to uzglabāšanai, meklēšanai un izgūšanai. Lai gan lielākoties visi projekti balstās uz atkārtoti izmantojamām bibliotēkām, tomēr tas nav obligāts nosacījums, lai sistēma būtu atkārtoti izmantojama. Viens no šādiem piemēriem ir Agora, kas ir programmatūras prototips, kura mērķis ir izveidot datubāzi. Šajā datubāzē tiek uzglabāti programmatūras produkti, kas tiek automātiski ģenerēti. Šī datubāze apvieno interneta meklētājprogrammu un introspekciju, kā rezultātā tiek samazinātas sistēmas piegādāšanas izmaksas un palīdz vieglāk atrast dažādas komponentes programmas kodā.

Grūtības atrast atkārtoti lietojamu programmatūru. Viena no svarīgākajām lietām atkārtoti izmantojamas sistēmās ir organizēta repozitorija. Ja ir loģiski sakārtota repozitorija, tad vieglāk izgūt dažādas sistēmas komponentes.

Atrastās programmatūras atkārtota neizmantošana. Lai gan atkārtoti izmantojamas sistēmas funkcionalitāte lielākoties palīdz samazināt darba laiku, tomēr ir gadījumi, kad funkcionalitāte ir tika sarežģīta, ka izdevīgāk ir uzrakstīt šo kodu no nulles, nevis mēģināt iedziļināties jau uzrakstītajā kodā.

Mantotie komponenti nav piemēroti atkārtotai lietošanai. Lai gan idejas līmenī, atkārtoti izmantojama sistēma paredz sistēmas koda pārkopēšanu, tomēr īstenībā tā nav. Atkārtoti izmantojamās sistēmās, kods tiek pārkopēts un pēc tam pielāgots nepieciešamajām vajadzībām. Līdz ar to, ir nepieciešams iedziļināties un izprast pielietoto kodu.

Modifikācija. Kā jau iepriekš tika minēts, ne vienmēr ir iespējams atrast koda fragmentu, kas atbilst nepieciešamajai situācijai, kā rezultātā ir nepieciešamas koda modifikācijas. Bet jāpatur prātā, ka šim modifikācijām ir jābūt spējīgām sadarboties ar pārējo sistēmas kodu.

3.3. Pielāgojamas sistēmas izstrāde

Atkārtoti izmantojamas sistēmas lietošana, kā jau iepriekš tikai minēts, sniedz daudz priekšrocībās. Aparatūras dziņu moduļos ir iekļautas sarežģītas problēmu risināšanas zināšanas, kas atkārtoti izmantojot šo komponentu sniedz lielu laika ieekonomēšanu. Kā rezultātā nav nepieciešams velīt neskaitāmas stundas šīs komponentes izstrādei. Atkārtoti izmantojamas sistēmas kodēšana ir sadalīta vairākās kategorijās, kurās tiek apskatīta to atšķirības savā starpā [6]:

1. **Transformācijas pret kompozīcijas atkārtotu izmantošanu.**

   Transformācijas sistēmu atkārtota izmantošana paredz sistēmas koda automātisku pārkopēšanu tieši tādā pat veidā, kā tas ir izvietots pirmkodā. Savukārt, kompozīcijas gadījumā izstrādātājs izvēlas sev vēlamo veidu kā pārnest un izkārtot kodu.

2. **Melnās pret balstās kastes atkārotu izmantošanu.**

   Melnās kastes gadījumā tiek izmantots produkta kods tāds, kāds viņš ir, un nekas netiek mainīts. Savukārt, baltās kastes pielietojumā, koda fragmenti tiek pielāgoti nepieciešamajai situācijai.

3. **Abstrakcijas atkārtota izmantošana.**

   Vispārīgā atkārtota izmantošana, kuras laikā netiek pievērsta padziļināta uzmanība, kodam, dizainam un testiem, jeb tiek pārkopēts viss projekts bez papildus iedziļināšanās.

4. Vertikāla pret horizontālu atkārtotu izmantošanu.

Vertikāli atkārtotā izmantošana ir objektu modeļu, algoritmu un ietvaru izmantošana tam paradzētā sistēmā, savukārt horizontāli atkārtotā izmantošana viss ierpiekš minētais, bet tiek izmantots kādai citai sistēmai, piemēram, autentifikācija sistēma, datubāzes un bibliotēkas.

5. Atkārtotas izmantošanas procedūras.

Tiek pielietotas tās pašas prasmes kas līdz šim. Projektu vadītāju aprindās, tas ir saprotams, kā darbaspēka piesaiste, kuram šī joma ir jau zināma, attiecīgi netiek patērēts laiks uz jauna darbinieka apmācību. Sistēmas izstrāde tiek sadalīta piecos posmos:

- Dizains
- Prasību un specifikācijas analīze
- Apkope
- Testēšana
- Kodēšana

Attiecīgi atkārtotas izmantošanas sistēmas izstrāde tiek sadalīta vēl divās grupās, lai optimizētu izstrādi:

- Ģeneratīvā metode. Ģeneratīvās metodes pamat uzdevums ir optimizēt lietojumprogrammu jomu un transformāciju secību programmatūras izstrādes laikā. Idejiski, koncepts ir līdzīgs automātiskajai programmēšanai.
- Kompozīcijas metode. Kā jau iepriekš tika minēts viens no populārākajiem veidiem ir izmantot atkārtoti izmantojamās bibliotēkas komponentes. Līdz ar šo metodi, šīs bibliotēkas tiek sakārtotas atbilstoši izstrādātāja vajadzībām un tiek paaugstināta darba efektivitāte, kā rezultātā tiek samazināts darbam patērētais laiks.

Liela daļa no programmētājiem izmanto atkārtotu izmantošanu, tomēr to kodu rakstīšanas maniere neatbilst atkārtoti izmantojamas sistēmas standartiem, kas ietver koda modularitāti. Šīs metodes, kas tiek izmantotas ir gana vienkāršas un līdz galam nesaistītas, kas noved pie nopietnas koda rediģēšanas, ja sistēmu vēlētos uzskatīt pa atkārtoti izmantojamu. Šajā kodā tiek izmantotas bibliotēkas komponentes, kas nav paredzētas atkārtotai izmantošanai.

Atkārtotas izmantošanas metodes ir balstītas uz noteiktām vadlīnijām un procedūrām, kuras ir jāievēro izstrādājot atkārtoti izmantojamu sistēmu. Tiek izstrādās plāns pēc kura ir jāvadās visas izstrādes laikā, pretējā gadījumā, radīsies daudz un dažādas problēmas ar programmatūras koda savietojamību, bet līdzšinējā prakse programmētājiem ir vairāk tendēta uz "lēkāšanu" starp programmas izstrādes plāna punktiem. Sistēmas atkārtota izmantošana:

- Nospraust mērķi un vīziju, kā sistēmas atkārtota izmantošana spēs palīdzēt sistēmas izstādē un kur tālāk šo sistēmu izmantos.
- Ir jādefinē vadības stratēģija, ka viss izstrādes process tiks virzīts uz priekšu, kā arī tehnisko stratēģiju, kas būs nepieciešams, lai šo sistēmu veiksmīgi varētu izveidot.
- Ir jābūt definētam sistēmās integritātes plānam, kā šo sistēmu integrēt kopējā sistēmas izstrādes procesā, un kādā veidā tiks nodrošināta sistēmas procesu uzglabāšana.
- Ir jābūt nodrošinātam visam personālam ar attiecīgo aparatūru, lai veiksmīgi ritētu darbs uz priekšu un nebūtu nekādas aizķeršanās, piemēram, ar kādam programmatūras licencēšanas jautājumiem.
- Ir jābūt skaidri definētam budžetam, gan tehnikas jomā, gan organizatoriskajā.
- Ir jābūt definētiem mērījumiem, pēc kuriem darbinieki spētu novērtēt sistēmas atkārtotas izmantošanas iespējamību.

4. Atkārtotas sistēmas pielietojums izstrādē

Šīs nodaļas ietvaros tiks apskatīts divu izstrādātu sistēmu atšķirības, kā arī ieguvumi, kuri ir gūti izstrādes procesa laikā. Viena no sistēmām ir fayma.com, kura pamata mērķis ir piedāvāt lietotājiem apģērba piegrieztnes izmantojot tikai telefonā iebūvēto foto kameru. Pēc šiem uzņemtajiem attēliem sistēma spēj noteikt precīzas ķermeņa proporcijas, attiecīgi tālāk nododot šos mērījumus šuvējam, tiek izgatavots un izšūts apģērbs pēc klienta auguma proporcijām.

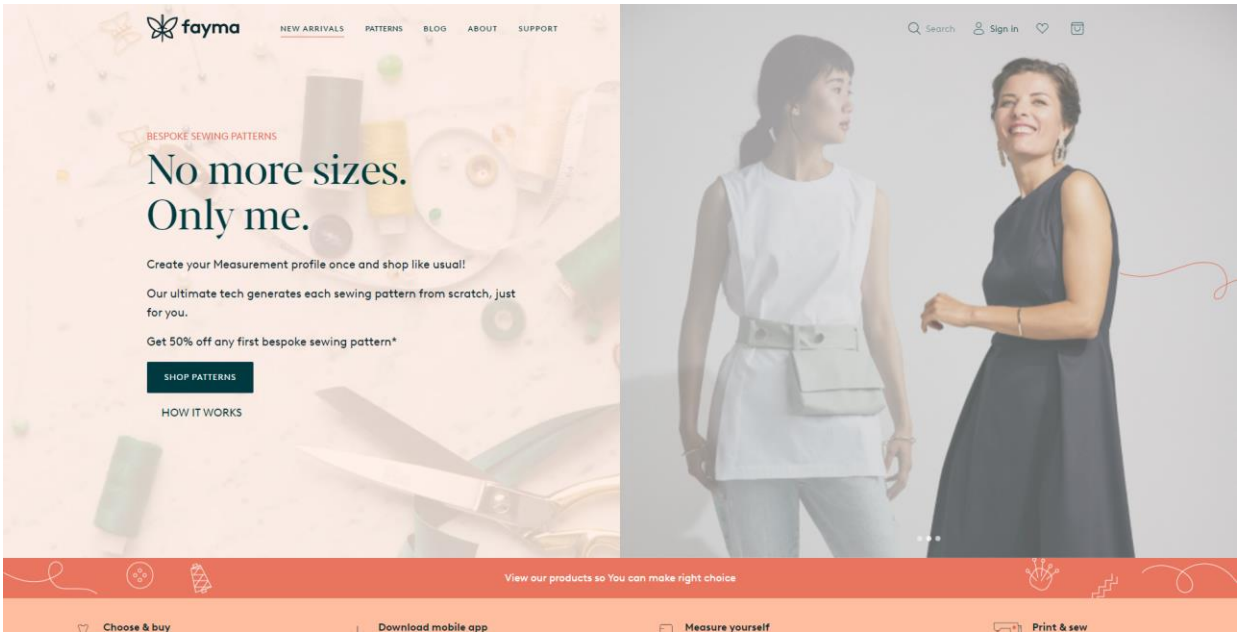
Otra sistēma, kurā ir iekļauta ļoti līdzīga produktu izvietošana ir hobbycraft.co.uk. Šis Lielbritānijas uzņēmums ievērojot visus licencēšanas noteikumus ir ieviesis tādu pat sistēmu kā fayma.com. Sistēmas uzstādīšanai tika izmantota atkārtoti izmatojamas sistēmas kods, kas palīdzēja šos sistēmu uzstādīt krietni vien ātrāk.

4.1. Atkāroti izmantojamas sistēmas pielietojums arhitektūrā

Kā jau iepriekš tika minēts atkārtoti izmantojamām sistēmām izskats var tik mainīts no logo, fona krāsām un teksta izkārtojumam, līdz pat sadaļu pilnīgu pārveidošanu, kuru dizains ir nesaistītas ar pamatsistēmas dizainu un nav iespējams saskatīt pamatsistēmas dizaina elementus.

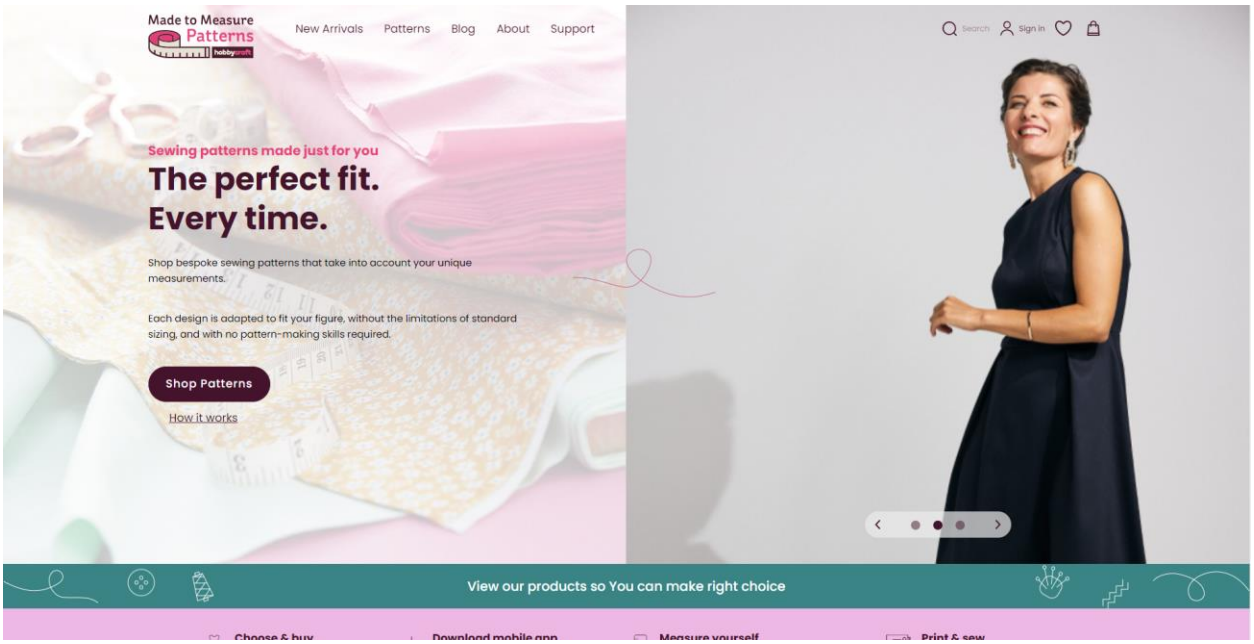
Šajā gadījumā tiks apskatīts pirmais variants. Ieskatam var apskatīt kā izskatās fayma.com mājas izklājlapa 4.1. attēlā.

4.1. att. Fayma.com izklājlapa



Un 4.2. attēlā ir iespējams apskatīt hobbycraft.co.uk mājas izklājlapu.

4.2. att. Hobbycraft.co.uk izklājlapa



Kā redzams abu mājaslapu izkārtojums ir gaužām līdzīgs, ir mainītas lielākoties tikai krāsas un tiek pielikti dažādi dizaina jeb css elementi. Kā redzams atkārtoti izmantojamas sistēmas pamat būtība šajos divos projektos ir ievērota, sistēmu ir viegla pārlikt un sistēma tiek

pielāgota klienta vajadzībām. Attiecīgi ielūkojoties koda fragmentos, 4.3. attēlā ir attēlots koda fragments no fayma.com projekta un 4.4. attēla attēlots hobycraft.co.uk projekts, nav redzamas būtiskas atšķirībās, tik vien kā teksts un tiek pieliktas dažas klases klāt pie html elementiem. Šīs klases hobbycraft.co.uk projektā netiek rakstītas no jauna, bet gan iegūtas no atkārtoti izmantojamu sistēmu bibliotēkām, kas krietni vien samazina nepieciešamo laiku, izveidotu nepieciešamo dizainu. Attiecīgi ja nepietiek ar esošajām klasēm, kas jau ir izveidotas, tiek izveidotas jaunas klases un pieliktas klāt bibliotēkā, kā šajā gadījumā ir izdarīts hobyycraft.co.uk projektā 7. rindiņā.

4.3. att. Fayma.com koda fragments

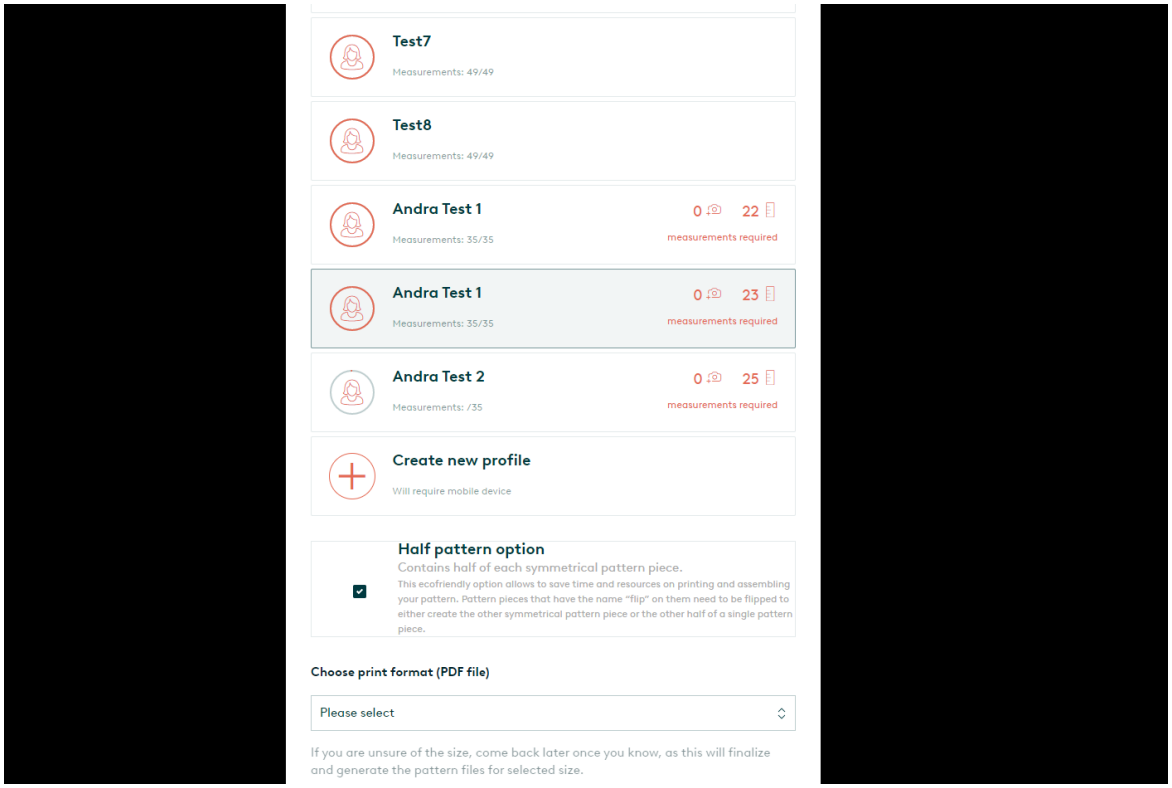
```
<div class="firstpage-teaser">
    <div class="firstpage-teaser__info">
        <div class="container-fluid">
            <div class="firstpage-teaser__text">
                <span class="text-secondary">
                    {{ __($translationRepository->get('BESPOKE SEWING PATTERNS')) }}
                </span>
                <h1>{{ __($translationRepository->get('No more sizes.')) }}<br />{{ __($translationRepository->get('Only me.')) }}</h1>
                <p>{{ __($translationRepository->get('Create your Measurement profile once and shop like usual!')) }}</p>
                <p>
                    {{ __($translationRepository->get('Our ultimate tech generates each sewing pattern from scratch, just for you.')) }}
                </p>
                <p>
                    {{ __($translationRepository->get('Get 50% off any first bespoke sewing pattern*')) }}
                </p>
                <p>{{ __($translationRepository->get('*Limited time offer')) }}</p>
                <a class="btn btn-primary btn-lg" href="/patterns">{{ __('Shop patterns') }}</a></br>
                 
                <a class="teaser-link" href="/support/how-it-works">{{ __('HOW IT WORKS') }}</a>

            </div>
        </div>
    </div>

    <style>
        .firstpage-teaser .firstpage-teaser__part-static {
            @if(config('settings.start-page-background-image'))
                background-image: url("{{config('settings.start-page-background-image')}}");
            @endif
        }
    </style>
```

4.4. att. hobbycraft.co.uk koda fragments

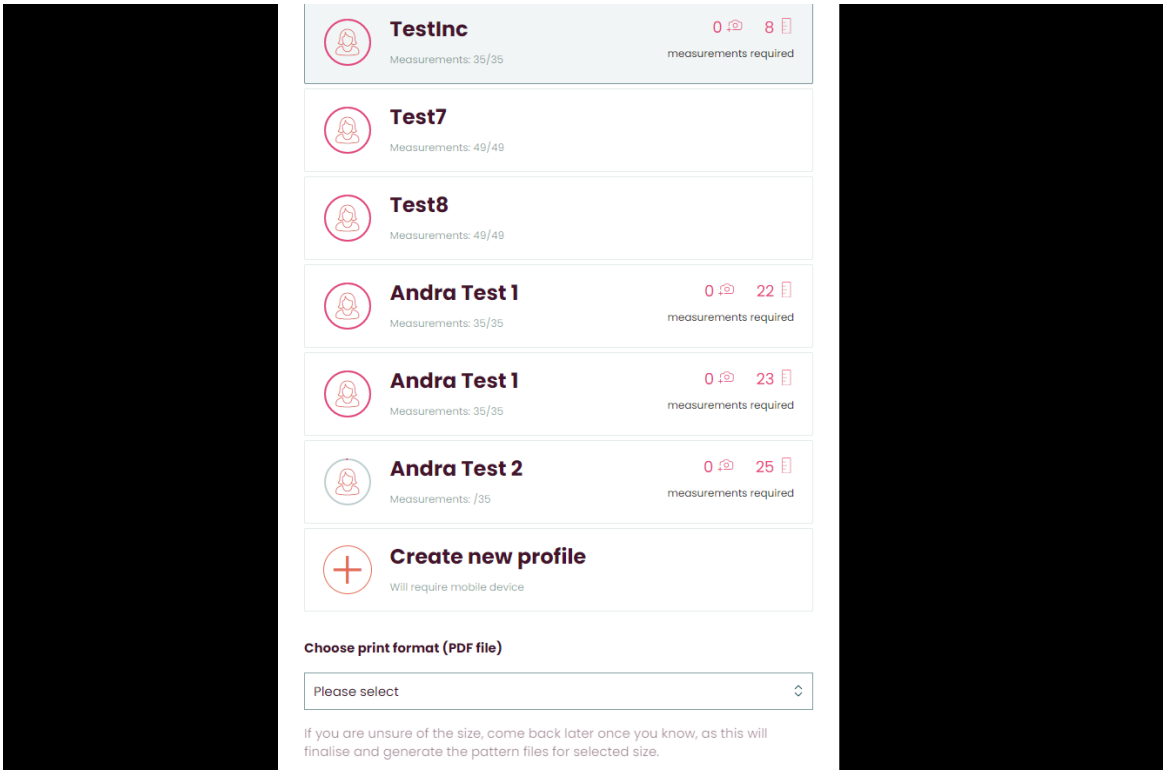
```
<div class="firstpage-teaser">
    <div class="firstpage-teaser__info">
        <div class="container-fluid">
            <div class="firstpage-teaser__text">
                <span class="text-secondary teaser-secondary-heading">
                    {{ __($translationRepository->get('Sewing patterns made just for you')) }}
                </span>
                <h1>{{ __($translationRepository->get('The perfect fit.')) }}<br />{{ __($translationRepository->get('Every time.')) }}</h1>

                <div class="teaser-description-text">
                    <p>{{ __($translationRepository->get('Shop bespoke sewing patterns that take into account your unique measurements.')) }}</p>
                </div>

                <div class="teaser-description-text">
                    <p>{{ __($translationRepository->get('Each design is adapted to fit your figure, without the limitations of standard sizing, and with no pattern-making skills re
                </div>

                <a class="btn btn-primary btn-lg mb-3 teaser-shop-button" href="/patterns/groups/all-patterns$order:latest">{{ __('Shop Patterns') }}</a></br>
                <a class="teaser-link" href="{{config('settings.how-it-works-url')}}">{{ __('How it works') }}</a>


            </div>
        </div>
    </div>

    <style>
        .firstpage-teaser .firstpage-teaser__part-static {
            @if(config('settings.start-page-background-image'))
                background-image: url("{{config('settings.start-page-background-image')}}");
            @endif
        }
    </style>
```

Protams, viens no galvenajiem faktoriem atkārtoti izmantojamām sistēmām, kas tika minēts iepriekš bija modularitāte, tad šis faktors ietekmē gan dizaina skatu, gan sistēmas uzbūvi kopumā. Viens no šo abu sistēmu piemēriem ir drukāt tikai pusi no apģērba piegrieztnes, par cik cilvēka ķermenis ir salīdzinoši simetrisks, un ir iespējams izmanot šo vienas puses piegrieztni uz abām pusēm. Šo ideju izmanto pašlaik tikai fayma.com projekts, kā redzams 4.5. attēlā, bet hobycraft.co.uk projekts šo ideju vēl neizmanto, kas ir redzams 4.6. attēlā.

4.5. att. fayma.com piegrieztņu skats

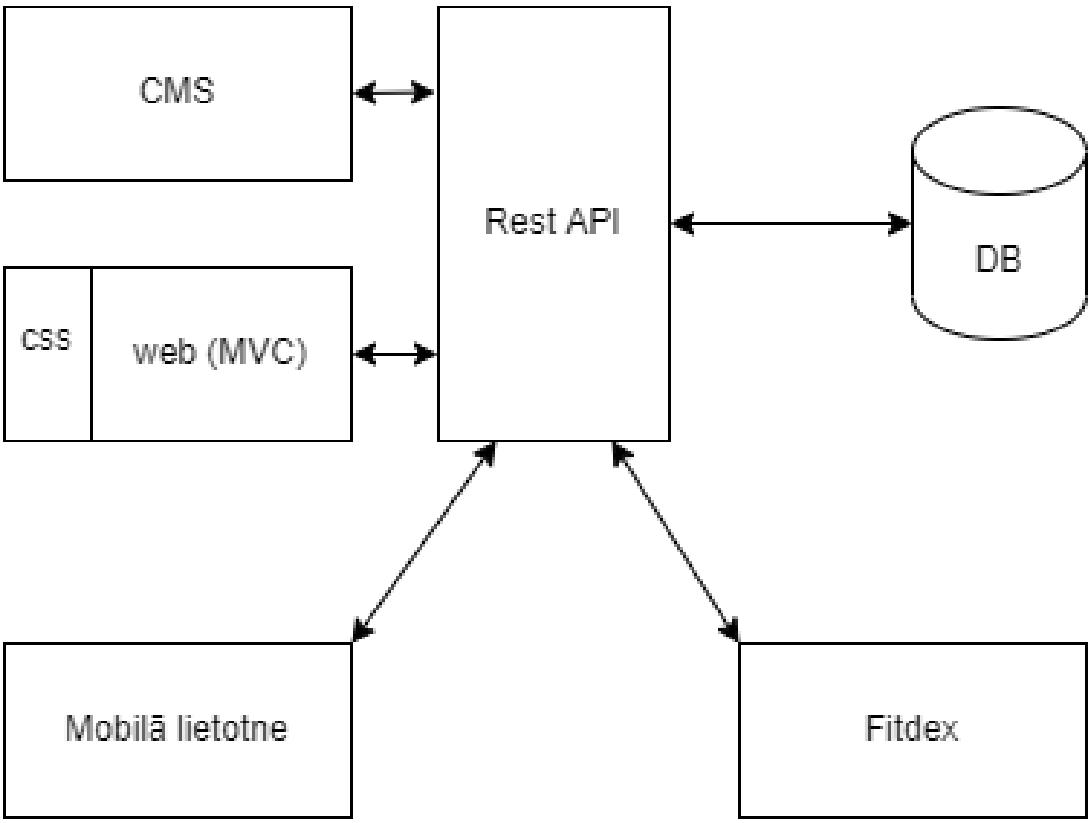


4.6. att. hobbycraft.co.uk piegriežņu skats



Lūk, šajā vietā ir redzams tas, ka izņemot vienu sadaļu no koda kopējais dizains netiek mainīts un viss pārējais paliek, tāds kāds bija iepriekš. Ja tiek apskatīta kopējās sistēmas arhitektūra, tad kopumā viss ir tāds pats, ir gan modeļi, gan kotroleri, gan skatu faili, gan repozitorijas, kuri tiek glabāti gan zem api, gan web, gan administrēšanas jeb cms projektiem. Visi šie faili tiek glabāti tieši tā pat kā pamatsistēmā, sistēmas arhitektūru var apskatīties 4.7 attēlā.

4.7 att. sistēmas arhitektūra



CMS – Satura pāraudzības sistēma (Content management system), jeb administrēšanas sistēma

Rest API - REST API (pazīstama arī kā RESTful API) ir lietojumprogrammu saskarne (API vai tīmekļa API), kas atbilst REST arhitektūras stila ierobežojumiem un ļauj mijiedarboties ar RESTful tīmekļa pakalpojumiem, jeb nodrošina komunikāciju starp datubāzi un sistēmām.

Web (MVC) - Model-View-Controller (MVC) ir arhitektūras modelis, kas sadala lietojumprogrammu trīs loģiskajās komponentēs: modelī, skatā un kontrollerī.

DB – datubāze.

Fitdex – sistēma, kas sagatavo pieggrieztnes.

Kā šīs izmaiņas ietekmē kopējo sistēmas kodu un darbību, tiks apskatīts nākamajā sadaļā.

4.2. Atkāroti izmantojama sistēma no aizmigursistēmas, jeb *backend* skatupunkta

Rakstot kodu aizmugursitēmai vienmēr ir jāpatur prātā, lai sistēmas rakstītais kods būt gana atdalāms jeb modulārs. Tad nu lūk, apskatīsim atkal piemēru par apģērba piegrieztni, kur tiek drukāta tikai puse no visas piegrieztnes. 4.8. attēlā ir attēlots koda fragments no fayma.com projekta, kurā ir pieejama šī opcija un 4.9. attēlā koda fragments no hobbycraft.co.uk, kurā ir izkomentēta šī opcija.

4.8. att. fayma.com koda fragments

```
<div id="halfPattern">
    <div class="custom-control custom-checkbox">
        <input value="1" type="checkbox"  class="custom-control-input" style="margin-left:20px; margin-top:50px;" id="print_half">
            <label class="custom-control-label" for="print_half">
                <h4 class="halfTitle">{{ __('Half pattern option') }}</h4>
                <p class="halfText">{{ __('Contains half of each symmetrical pattern piece.') }}</p>
                <small class="halfText">{{ __('This ecofriendly option allows to save time and resources on printing and assembling your pattern. Pattern
            </label>
    </div>
</div>
<div id="fp-print-format">
    <p class="print-instruction">{{__("Choose print format (PDF file)")}}</p>
    <select id="selectPrintId" data-bind="purchasedpatterns.selectPrint">
        <option value="empty" >Please select</option>
        <?php
        $printSizes = $finalizeResource->printSizes;
        //foreach($printSizes as $sizeKey => $sizeOption):
        foreach($printSizes as $sizeArray):
        ?>
            <option value="<?php print $sizeArray['id']; ?>"><?php print $sizeArray['name']; ?></option>
        <?php endforeach; ?>
    </select>
    <p class="print-hint">
        {{__("If you are unsure of the size, come back later once you know, as this will finalize and generate the pattern files for selected size.")}}
    </p>
    <p class="print-hint">
        {{__(" In printer settings make sure that "Actual Size" or "Custom Scale: 100%" is selected.")}}
    </p>
</div>
```

4.9. att. hobbycraft.co.uk koda fragments

```
</div>
    <div id="halfPattern">
    <div class="custom-control custom-checkbox">
        <input value="1" type="checkbox"  class="custom-control-input" style="margin-left:20px; margin-top:50px;" id="print_half">
            <label class="custom-control-label" for="print_half">
                <h4 class="halfTitle">{{ __('Half pattern option') }}</h4>
                <p class="halfText">{{ __('Contains half of each symmetrical pattern piece.') }}</p>
                <small class="halfText">{{ __('This ecofriendly option allows to save time and resources on printing and assembling your pattern. Patter
            </label>
    </div>
</div>-->
<div id="fp-print-format">
    <p class="print-instruction">{{__("Choose print format (PDF file)")}}</p>
    <select id="selectPrintId" data-bind="purchasedpatterns.selectPrint">
        <option value="empty" >Please select</option>
        <?php
        $printSizes = $finalizeResource->printSizes;
        //foreach($printSizes as $sizeKey => $sizeOption):
        foreach($printSizes as $sizeArray):
        ?>
            <option value="<?php print $sizeArray['id']; ?>"><?php print $sizeArray['name']; ?></option>
        <?php endforeach; ?>
    </select>
    <p class="print-hint">
        {{__("If you are unsure of the size, come back later once you know, as this will finalise and generate the pattern files for selected size.")}}
    </p>
    <p class="print-hint">
        {{__(" In printer settings make sure that "Actual Size" or "Custom Scale: 100%" is selected.")}}
    </p>
</div>
<div>
```

Kā jau daudzi programmētāji zina, formām lielākoties to elementi tiek uzstādīti kā obligāti, ja šis elements tālākai izstrādei ir gana nozīmīgs, kas ir arī šajā gadījumā. Tad nu lūk, šajā vietā nāk palīgā koda modularitāte, kas atļauj izņemt koda fragmentu un formas

aizpildīšanas rezultātā netiek izmestas kļūdas par nepareizi aizpildītu formu. Šis konkrētais gadījums tiek risināts sekojoši ar *javascript* palīdzību, kas ļauj klientam izņemt šo ievadlauku ārā no redzamās daļas. Par cik projekta laikā tiek veikti dažādi pieprasījuma ar *api* palīdzību, kas atgriež no datubāzes nepieciešamos datus vai tieši pretēji tie nosūtīti tie uz datubāzi. Kā redzam 4.10. attēlā ar *javascript* palīdzību vienmēr šis nepieciešamais lauks ir aizpildīts, tiek padots vai nu vērtība "0" vai vērtība "1".

4.10. att. fayma.com koda fragments

```
//Posts submit button atributes
createPattern: function (el, data) {
    el.click(function(){
        $("#fp-error").remove();
        var btnText = $(el).text();

            dataToSend =
            {
                order_item_id: data.orderitemid,
                change_print: data.changeprint,
                print_half: $('#print_half').is(':checked') ? 1 : 0,
                profile_id: $("#fp-submit").attr("data-profileid"),
                selected_size: ($("#fp-submit").attr("data-selectedsize") == "null" ? null : $("#fp-submit").attr("data-selectedsize"))

            };
```

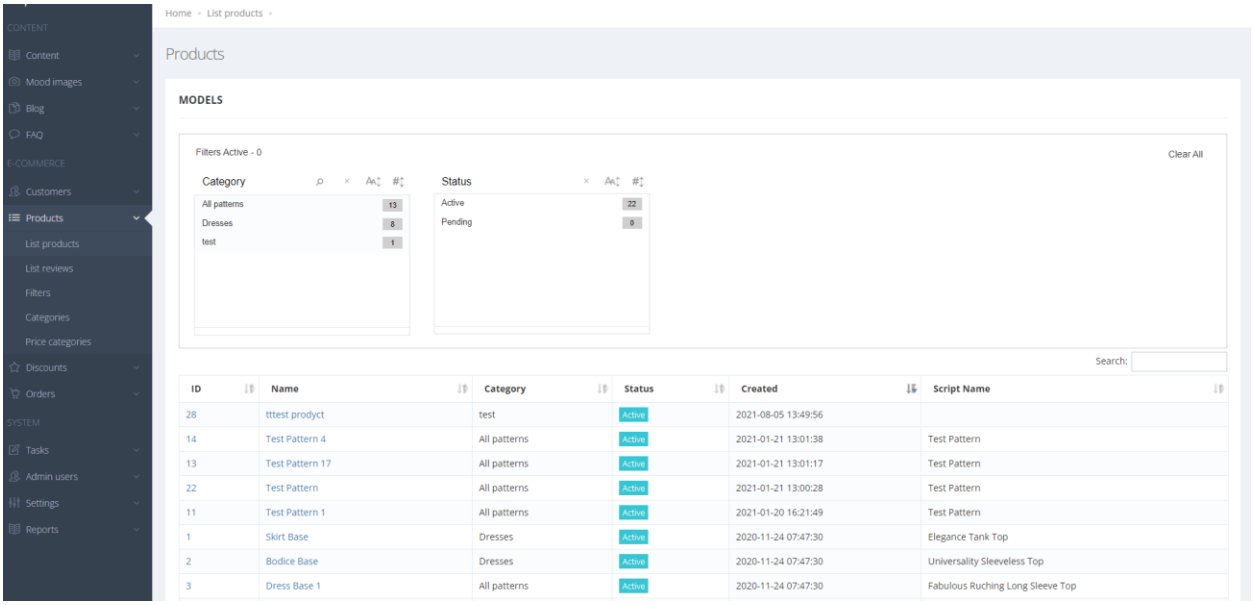
Šīs noved pie tā, ka klients fiziski šo lauku nevar saskatīt sistēmā un atzīmēt to, bet formas validācija nostrādā un netiek izmestas kļūdas. Kā arī tālāk uz datubāzes pieprasījumu, kas iet caur *api* šis pieprasījums netiek apturēts, jo visi nepieciešamie lauki no klienta skatu punkta teorētiski, bet no sistēmas izstrādes skatu punkta praktiski tiek aizpildīti.

Par cik šī funkcionalitāte jau sākotnēji tika programmēta ar domu, ka ir jābūt iespējai viņu arī neizmantot, tad tikai iegūts laiks uz to, ka nav jātaisa atsevišķa forma katram no projektiem, kā arī nav nepieciešams izveidot jaunu *api* izsaukumu speciāli šim gadījumam.

4.3. Atkārtoti izmantojamas sistēmas efektivitāte administrēšanā

Ja jau dizaina jeb *frontend* projekta pārnešana ir tik viegla, tad tik pat viegli ir jābūt arī pārnest sistēmas administrēšanas sadaļu. Tā tik tiešām arī ir. Ja sistēma ir paredzēta kā atkārtoti izmantojama sistēma, tad sistēmas kods un funkcijas tiek kodētas modulāri, kā rezultātā šiem abiem projektiem ir iespējams izmantot gandrīz identiskas sistēmas administrēšanas vietnes. Kā redzams 4.11. attēlā.

4.11. att. fayma.com un hobbycraft.co.uk administrēšanas sistēmas skats



Bet ja nepieciešams, ir iespējams izņemt vai tieši pretēji pielikt kādu no sadaļām. Lai nodrošinātu šo modularitāti, kas spēj mums dot šādas iespējas, ir jāspēj uzkodēt sistēmas, kas ir gana vispārīgas, bet tieši tajā pašā laikā gana konkrētas. Tieši tāpēc tiek izmantoti konkrēti mainīgie, kuriem visos projektos būs vieni un tie paši dati. Šie mainīgie vairāk vai mazāk ir piesieti konkrētam skatam, kas netiek mainīts. Tieši tāpēc projekta laikā rodas ļoti daudz mainīgo uz vienu skatu, kā piemēru var apskatīt 4.12. attēlu.

4.12. att. fayma.com koda fragments

```
public function edit(int $productId)
{

        $product = $this->productRepository->get($productId);
        $productCategories = $this->productCategoryRepository->parentless()->get();
        $priceCategories = $this->priceCategoryRepository->all();
        $measurements = $this->measurementRepository->all();
        $pvns = config('ecommerce.pvn');
        $productFilterCategories = $this->productFilterCategoryRepository->all();
        $products = Product::where('id','!=', $productId)->get();
        $videos = ProductVideo::where('product_id','=', $productId)->get();
        $relatedProductIds = $this->getRelatedIds($product);
        return view('admin::product.edit')
                ->with('pageTitle', $this->pageTitle)
                ->with('pageSubTitle', $this->pageSubTitle)
                ->with('product', $product)
                ->with('videos', $videos)
                ->with('pvns', $pvns)
                ->with('productCategories', $productCategories)
                ->with('priceCategories', $priceCategories)
                ->with('measurements', $measurements)
                ->with('products', $products)
                ->with('relatedProductIds', $relatedProductIds)
                ->with('productFilterCategories', $productFilterCategories);
}
```

Normālā programmēšanas praksē uz informācijas labošanu tiek padoti no 1-5 mainīgajiem, bet kā redzams šeit šis skaitlis ir dubultojies. Iemesls ir tāds, ka ja šie dati tiek izmantoti, pēc iespējas mazāk tie būtu atkarīgi viens no otra, kā rezultātā ir iespējams izņemt vai tieši pretēji pielikt jaunu funkcionalitāti esošajam projektam neizmainot jau esošo programmatūra kodu pārāk drastiski. Kā piemēram, šim pašam labošanas skatam, kur ir iespējams apskatīt 4.13. attēlā html koda formātā, netiek izmantoti visi padotie mainīgie, bet tikai kuri ir nepieciešami.

4.13. att. fayma.com koda fragments

```
<form class="forget-form form-horizontal form-row-seperated" enctype="multipart/form-data" id="object_form" action="{{$product->getAttribute('id') ? route('admin.products.update'
    @csrf
    @if($product->getAttribute('id'))
        @method('PATCH')
    @endif

    <div class="form-group">
        <label class="control-label col-md-3" for="name">Name</label>
        <div class="col-md-9">
            <input class="form-control placeholder-no-fix" type="text" autocomplete="off" placeholder="Name"
                name="name" value="{{old('name',$product->getAttribute('name'))}}"/> </div>
    </div>

    <div class="form-group">
        <label class="control-label col-md-3" for="slug">Slug</label>
        <div class="col-md-9">
            <input class="form-control" type="text" autocomplete="off" placeholder="Slug" name="slug" value="{{old('slug',$product->getAttribute('slug'))}}"/>
        </div>
    </div>

    <div class="form-group">
        <label class="control-label col-md-3" for="description">Description</label>
        <div class="col-md-9">
            <textarea style="width: 100%; height: 300px;" class="form-control" id="content" name="description">{{old('description',$product->getAttribute('description'))}}</texta
        </div>
    </div>

    <div class="form-group">
        <label class="control-label col-md-3" for="details">Details</label>
        <div class="col-md-9">
            <textarea style="width: 100%; height: 300px;" class="form-control" id="details" name="details">{{old('details',$product->getAttribute('details'))}}</textarea>
        </div>
    </div>
```

Tieši tā pat ir ar konkrētās lapas iekšējām vietnēm, kas ved uz citu skatu. Šajā lapās tiek izmantots gana vispārēja vietnes saite, bet tajā pašā laikā ar šiem mainīgajiem tiek nodrošināta konkrētība uz kuru lapu doties tālāk, kas nodrošina iespēju atkārtoti izmantot sistēmu, jeb precīzāk šajā gadījumā sistēmas skatu. Skatīt 4.14. attēlā.

4.14. att. fayma.com koda fragments

```
    @foreach($products as $product)
            <tr>
                <td>
                    <a href="{{route('admin.products.edit',['product' => $product->getAttribute('id')])}}">
                        {{$product->getAttribute('id')}}
                    </a>
                </td>
                <td>
                    <a href="{{route('admin.products.edit',['product' => $product->getAttribute('id')])}}">
                        {{$product->getAttribute('name')}}
                    </a>
```

4.4. Atkārtotas izmantšanas īpašību vērtējums piemēra sistēmi

Kopumā sistēmu programmēšanā tika ievērots princips, ka kodam ir jābūt modulāram, kas arī bija viens no galvenajiem nosacījumiem lai sistēma būtu atkārtoti izmantojama teorētiskajā aspektā. Lai gan sākotnēji rakstīt modulāru kodu likās diez gan grūti, bet ar laiku tas likās krietni vien loģiskāk un iespējams pat vieglāk. Kā rezultātā jau pie otrās sistēmas koda rakstīšanas teorija par ilgākām darba stundām nav patiesa.

Lai gan teorijas daļā ir minēts par dizaina struktūras matricām, tomēr šī projekta ietvaros šādas matricas netika izmantotas, kas brīžiem sagādāja neērtības ar dizaina elementu apstrādi, iespējams, ievērojot šo principu nebūtu radušās šādas problēmas. Atkārtotās sistēmas ieviešanā tika ievērota transformācijas atkārtotā izmantošana, kā rezultātā kods tika pārkopēts un tikai tad tika veiktas izmaiņas, lai pielāgotu sistēmu klienta prasībām, šajā gadījumā uzskatu ka šī izvēle bija pareiza, jo netika tērēts pārāk daudz laika uz atkārtotu koda izvērtēšanu.

Sistēmas izstrādē tika izmantota gan melnās, gan baltās kasres atkārtotas izmantošanas pieeja, jo, piemēram, melnās kastes pieeja tika pielietota maksājuma sistēmai, kurai nebija nepieciešami pārveidojumi, bet savukārt baltās kastes pieeja tika pielietota dizaina elementu pārnešanai. Darba autors uzskata šāda pieeja ir pozitīvi vērtējama, netika izmantoti papildus resursi sistēmu ieviešanai, kur tas nav bijis vajadzīgs, tādējādi tika ieekonomēts laiks.

Darba ietvaros netika izmantota arī abstrakcijas atkārtotā izmantošana, jo rakstot kodu pirmajai sistēmai visi bija iedziļinājušies kodā. Abu sistēmu izstrādē tika izmantota gan vertikālā, gan horizontālā atkārtotās izmantošanas metode. Vertikālā metode tika izmantota pirmās sistēmas kodēšanas ietvaros, savukārt horizontālā tika izmantota otrās sistēmas kodēšanā. Darba autors uzskata, ka šāds lēmums ir labs, jo abu sistēmu funkcionalitāte derēja abiem gadījumiem, tādējādi tika ieekonomēts laiks.

Projekta ietvaros tika izmantota arī atkārtotas izmantošanas metode, jo projekta vadītājs piesaistīja tos pašus programmētājus, kas izstrādāja pirmo sistēmu. Darba autors uzskata ka šāda pieeja ir pareiza, jo netiek patērēts laiks uz jaunu programmētāju apmācību un integrēšanu sistēmas izstādei.

Autors uzskata, ka sistēmā izmantotie atkārtotas sistēmas izstrādes principi, palīdzēja tās pielāgošanā jaunajai vietnei. Nozīmīgāko ietekmi uz sistēmas pielāgojamību atstāja sistēmas modularitāte.

4.5. Atkārtotas izmantošanas vērtējums piemēra sistēmai.

Lai novērtētu reāli atkārtojamu izmantojamu sistēmu ir nepieciešams apskatīt vēlreiz formulas mainīgos, kas ir : Atbalsts un kontaktinformācija, Dokumentācija, Modularitāte un

sarežģitība, Uzstādīšana un iepakošana, Atkārtotas izmantošanas tiesības, Testēšana, Sertifikācija, Paplašināšana.

- Atbalsts un kontaktinformācija ir vērtējama ar 3 punktiem, jo pamatsistēmas izstrādi veidoja tieši tas pats uzņēmums kurš tālāk izveidoja atkārtoto sistēmu, līdz ar to viss nepieciešamais atbalsts tika nodrošināts bez aizķeršanās.
- Dokumentācija ir vērtējama ar 0 punktiem, jo izstrādes laikā netika rakstīta dokumentācija un tālākas sistēmas nodošanai šī dokumentācija arī nebija nepieciešama, par cik viens un tas pats uzņēmums veidoja abas sistēmas.
- Modularitāte un sarežģītība ir vērtējama ar 3 punktiem, jo gan starpmoduļu, gan intramoduļu savienojamība ir gana vienkārša.
- Uzstādīšana un iepakošana ir vērtējama ar 2 punktiem, jo uzstādīšana ir veicama pusautomātiski, bet secīgi izpildot soļus ir iespējams nodrošināt pilnīgu sistēmas uzstādīšanu.
- Atkārtotas izmantošanas tiesības ir vērtējama ar 2 punktiem, jo ir iekšēji uzņēmumā definētas un pētītas atkārtoti izmantojamas tiesības.
- Testēšana ir vērtējama ar 4 punktiem, jo ir pieejams pārbaudes plāns, kā arī to testa rezultāti, kods ir izgājis cauri statiskās testēšanas rīkiem, kā arī trešā puse ir pārbaudījusi funkcionālās prasībās.
- Sertifikācija ir vērtējama ar 0 punktiem, jo sistēmai nav iziets SQA/IV&V/ sertifikāts.
- Paplašināšana ir vērtējama ar 3 punktiem, jo programmatūra ir sākotnēji paredzēta vairākām platformām un tiek veidotas atkarības ar konfigurācijas detaļām. Kopsummā: 3 * 1 + 0 * 2 + 3 * 3 + 2 * 4 + 2 * 5 + 4 * 6 + 0 * 7 + 3 * 8 = 78 punkti

Kopumā saskaitot visus punktus sanāk 78 punkti no 131 punktiem. Kā rezultātā izvērtējot rādītājus, kuri nesasniedz maksimālos punktus un kopējo punktu skaitu, var secināt, ka sistēma būtu atkārtojami izmantojama, un ja tiktu piestrādāts pie dokumentācija un sertifikācijas, kas šajā konkrētajā gadījumā līdz šim nav bijis nepieciešams tad sistēma būtu gandrīz 100% atkārtoti izmantojama.

Rezultāti

Darba ietvarā tika apskatīts kas ietekmē un kādas metodes būtu jāizmanto, lai pēc iespējas kvalitatīvāk būtu iespējams uztaisīt atkāroti izmantojamu sistēmu jeb *white-label* sistēmu. Apskatot darbā izmantotos avotus var secināt, ka ne viss kods ir vienlīdz atkārtoti izmantojams, bet ir trīs faktori, kas to ietekmē visvairāk. Pirmais faktors ir modularitāte, otrais ir testēšana un trešais ir pārvietojamība.

Pētot literatūras avotus var secināt, ka lai sasniegtu pēc iespējas augstāku koda modularitāti, kas ir arī viens no galvenajiem atslēgas vārdiem atkārtoti izmantojamām sistēmām, ir nepieciešams projektu sākumā ieviest šo faktoru sistēmas dizaina un koda izstrādes struktūrās. Izstrādājot kādu funkcionalitāti, programmētājiem būtu jau jāzina, kam šī funkcionalitāte būs nepieciešama, lai spētu ievērot savstarpējās atkarības starp sistēmas moduļiem.

Pēc izpētītajiem literatūras avotiem var secināt, ka sistēmas testēšanai ir jāpievērš papildus uzmanība, jo laicīgi neatklāta kļūda novedīs pie visai ievērojamām problēmām tālākā sistēmas darbībā. Bet savukārt, tālākos sistēmas izstrādes posmos, kad produkts ir nodots cita klienta izstrādei, būs nepieciešams krietni vien mazāk pavadīt laiku sistēmas testēšanai.

Apskatot dažādus literatūras avotus, var secināt, ka pārvietojamība ir atkarīga no tā cik ļoti labi ir uzrakstīta dokumentācija, kā arī no tā cik ļoti visa projekta ietveros tikai ievēroti noteiktie standarti, jo tālākas atkāpes no standartiem, jo krietni vien grūtāk būs pārvietot projektu.

Secinājumi

White-label jeb atkārtoti izmantojamas sistēmas ir sistēmas kods, kurš tiek rakstīts mērķtiecīgi ievērojot principus. Kodam un sistēmai ir jābūt izveidotai tā, lai izņemot, pieliekot vai izmainot kādu no koda daļām, sistēma spētu tupināt darboties un neradītu būtiskas kļūdas. Līdz šim daudziem zināmais jēdziens modularitāte ir atslēgvārds atkāroti izmantojamām sistēmām, jo modulārā kodā ir iespējams veikt visas trīs iepriekš minētās lietas.

Viens no galvenajiem ieguvumiem atkārtoti izmantojamām sistēmām ir nepieciešamo resursu un produkta izstrādes laika samazināšanās, jo nav vairs nepieciešams atkāroti rakstīt ne funkcijas kas reiz jau ir uzrakstītas, ne dizaina elementu atkārota pievienošana, jo viss jaut tiek uzglabāts atkārtoti izmantojamu sistēmu bibliotēkās.

Par cik mūsdienās laiks ir ļoti dārgs ir iespējams izvērtēt vai sistēmu būs rentabli taisīt, kā atkārtoti izmantojamu sistēmu, izmantojot atkārotas izmantošanas koeficientu.

Izmantotā literatūra un avoti

1. A Reuse-Based Software Development Methodology , K. C. Kang S. Cohen R. Holibaugh J. Perry A. S. Peterson , 1992
   Pieejams internetā: https://apps.dtic.mil/sti/pdfs/ADA258255.pdf

2. AN examination of issues related to the implementation of evolutionary development and DOD-STD-2167A in software development, Kathleen J Hall, 1990
   Pieejams internetā: https://vtechworks.lib.vt.edu/bitstream/handle/10919/41813/LD5655.V851_1990.H344.pdf?sequence=1&isAllowed=y

3. WHITE-LABELING SOFTWARE PRODUCTS, Cann, Roderick van, Brinkkemper, Sjaak, 2010
   Pieejams internetā: https://www.researchgate.net/publication/290179884_White-labeling_software_products

4. The Open Source Definition, Bruce Perens, 2008
   Pieejams internetā: https://www.researchgate.net/profile/Bruce-Perens/publication/200027107_Perens_Open_Source_Definition_LG_26/links/568bdeb408ae16c414a9c549/Perens-Open-Source-Definition-LG-26.pdf

5. Reusability Metrics for Object-Oriented System: An Alternative Approach, Parul Gandhi, Pradeep Kumar Bhatia
   Pieejams internetā: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.741.9123&rep=rep1&type=pdf

6. Review of Software Reuse Processes, Ljupcho Antovski, Florinda Imeri, 2013
Pieejams internetā: https://www.researchgate.net/profile/Florinda-Imeri/publication/262377645_Review_of_Software_Reuse_Processes/links/5633bbc208aebc003ffde2bd/Review-of-Software-Reuse-Processes.pdf

7. Software Engineering with Reusable Components, J.Sametinger, 1997
Pieejams internetā: https://books.google.lv/books?hl=lv&lr=&id=AxPQvGRs2wUC&oi=fnd&pg=PA1&dq=reusable+software+systems&ots=Ka8LA7RHIN&sig=270y7RT-HZRenRakHOondwgfurQ&redir_esc=y#v=onepage&q=reusable%20software%20systems&f=false

8. The Impact of Component Modularity on Design Evolution: Evidence from the Software Industry, 2017
Pieejams internetā: https://deliverypdf.ssrn.com/delivery.php?ID=931009020100123067093111093100008098096084018006060085124104024024104029086026080096056033123048021034111127091092102101067095123076062000041118085084027105076065037017014087088094095085119086085103071086085007105080112000004121071029007075118084081&EXT=pdf&INDEX=TRUE

9. Data Organisation and Process Design Based on Functional Modularity for a Standard Production Process, 2018
Pieejams internetā: https://www.proquest.com/openview/7cb612163f4beac7fc910c759f287aac/1?pq-origsite=gscholar&cbl=105444

10. Functional Architecture Modeling for the Software Product Industry, 2010
Pieejams internetā: https://link.springer.com/chapter/10.1007/978-3-642-15114-9_16

11. Software Architecture in Practice, 1988
Pieejams internetā:

https://books.google.lv/books?hl=lv&lr=&id=mdiIu8Kk1WMC&oi=fnd&pg=PA1&ots=UfO-RbfcMQ&sig=er37tjEa_83hV1m8bmnZXv2UC1w&redir_esc=y#v=onepage&q&f=false