LATVIJAS UNIVERSITĀTE
Matemātikas un informātikas institūts

MĀRTIŅŠ GILLS

PROGRAMMATŪRAS TESTĒŠANA UN TRASĒJAMĪBA

Promocijas darbs

Rīga - 2005

LATVIJAS UNIVERSITĀTE
Matemātikas un informātikas institūts

MĀRTIŅŠ GILLS

PROGRAMMATŪRAS TESTĒŠANA UN TRASĒJAMĪBA

Promocijas darbs
datorzinātņu doktora (Dr. sc. comp.) zinātniskā grāda iegūšanai

Nozare: datorzinātnes
Apakšnozare: datoru un sistēmu programmatūra

Zinātniskais vadītājs:
profesors, Dr. habil. sc. comp.,
JURIS BORZOVS

Rīga - 2005

Saturs

1. Ievads

1.1. Pētījuma motivācija

Programmatūras izstrāde veidojas no vairāku procesu mijiedarbības, kuri norit ar atšķirīgu intensitāti dažādos laika posmos. Ja sākotnēji intuitīvi vissvarīgākais no visiem procesiem varētu šķist implementāciju jeb kodēšanu veicošais, tad vairāku desmitu gadu laikā izmēģinātās programmatūras pieejas liecina, ka liela izmēra programmatūrai ir nepieciešama arī iepriekšēja specificēšana un projektēšana, kā arī implementēšanas laikā un pēc tās - testēšana. Tieši testēšana ir tā aktivitāte, kas pārbauda, vai iepriekšējos procesos notikušais ir nodrošinājis kvalitatīvas programmas izveidi. Ņemot vērā to, ka testēšanai ir pārbaudošs raksturs, tās formulētie mērķi un metodes parasti ir ar "destruktīvu" motivāciju. Jāņem vērā, ka testēšana nav tikai programmatūras izstrādes noslēdzošā aktivitāte - tā ļoti bieži ir pagrieziena punkts, lai sāktos jauna izstrādes iterācija - tiktu veiktas izmaiņas kodā, projektējumā vai prasībās.

Programminženierijā viena no kvalitātes problēmām saistās ar testēšanas spējām sniegt pārliecību par to, ka ir pārbaudīta un līdz ar to arī zināmas visas mūs interesējošās programmatūras īpašības. Mūsdienās lielākā daļa programmatūras, kas nonāk lietošanā, netiek pārbaudīta ar formālām validācijas un verifikācijas metodēm, bet gan tiek pielietotas uz dažādiem apsvērumiem balstītas testēšanas vai citas, pamatā selektīvas, metodes. Ja klasiskajās inženierzinātnēs eksistē metodes precīzu un uzticamu mehānisku vai elektronisku iekārtu izgatavošanai, tad programmatūras izstrāde ir pakļauta komerciālajiem principiem – nemitīgi papildināt programmatūru ar jaunām iespējām. Arī eksistējošās metodes uzticamu programmu iegūšanai pamatā koncentrējas uz programmas kā darbināma koda pareizību, bet bieži tiek ignorētas ārpus programmas esošās īpašības – programma kā sistēmas sastāvdaļa. Nereti programmizstrādes projektos veidojas tāda situācija, ka veidojamā produkta pārbaude koncentrējas uz tehniskas dabas jautājumiem, bet nākošajiem lietotājiem ir svarīga jaunās programmatūras dabiska integrācija viņu darbu procesā. Šāda veida pārbaude attiecas uz akcepttestēšanu, bet labi organizētas testēšanas gadījumā - arī uz izstrādes laikā veicamo sistēmtestēšanu.

Testēšana veic pārbaudi, izmantojot dažāda tipa projektā esošo informāciju, kā arī uz testēšanas rezultātiem balstās virkne tālāko programmatūras aktivitāšu. Šī saistība nozīmē to, ka ir jāeksistē atbilstošai trasējamībai jeb iespējai izsekot no programmas implementācijas informācijai līdz testēšanas informācijai un no testēšanas informācijas uz jauno implementācijas iterāciju. Lai arī trasējamības jēdziens ir dabiski apjaušams, un tas ir sistemātiski sastopams arī citās nozarēs, piemēram, farmācijā, programmatūras izstrādē tam bieži netiek veltīta pienācīga uzmanība, vai arī tas tiek veikts tik minimāli un formāli, ka reāli nedod projektiem reālo atdevi.

Trasējamību kā programmizstrādes īpašību prasa virkne starptautisku standartu. Piemēram, programmatūras tipveida izstrādes procesu nosakošajos standartos tādos, kā IEEE J-STD-016 [69] un ISO/IEC 12207 [76] trasējamība ir aplūkota gan kā primāro programmatūras dzīves cikla procesu, gan kā procesos iegūto produktu īpašība. Pētījumu sākumu posmā autora veiktais novērtējums deva rezultātu, ka standartos ietvertās prasības attiecībā uz trasējamību ir saprātīgas, bet tomēr netiek plaši praktizētas.

Viens no šeit sniegtā pētījuma iemesliem bija saprast, kādēļ saistībā ar trasējamības jautājumiem veidojas pretruna starp standartos prasīto un praksē pielietoto. Autors saskata, ka projekti un testēšanas process iegūtu no tā, ja varētu plašāk praktizēt trasējamību. Šajā kontekstā ir jānovērtē paņēmieni, lai mazinātu programmatūras procesa dalībnieku subjektīvo uztveri par trasējamību kā par apgrūtinājumu. Ignorējoša vai negatīva projektu nostāja pret trasējamību veidojas daļēji tā iemesla dēļ, ka to nav iespējams nodrošināt ar vienkāršiem tehniskiem līdzekļiem, bet pieejamo metožu pielietošana nav ekonomiski izdevīga. Ja trasējamību pēc būtības neizmanto, bet cenšas to realizēt tikai kā formalitāti, ir jāpatērē papildus darbs bez pozitīvas atgriezeniskās saites. Tādēļ viens no dabiskiem risinājumiem būtu maksimāli padarīt trasējamības informācijas uzturēšanu automatizētu vai kā blakusproduktu citām projekta aktivitātēm, kuras pieder pie pamatprocesa un kuru nepieciešamība netiek apšaubīta.

1.2. Darba mērķi

Kad aptuveni piecu gadu garumā, strādājot Rīgas Informācijas tehnoloģijas institūta Testēšanas laboratorijā, autors detalizēti bija apguvis programmatūras testēšanu, viņš veica vairākus pētījumus par testēšanas un kvalitātes nodrošināšanas praksi IT uzņēmumos. Šiem jautājumiem bija veltīts maģistra darbs, kā arī sagatavoti referāti starptautiskām konferencēm. Novērtējot to, kādas problēmas eksistē testēšanas procesā, turpmākajā darbā viņš novēroja, ka būtiska problēma ir uzturēt trasējamības informāciju reāla projekta apstākļos. Autors konstatēja trasējamības nozīmi testēšanas procesā, kā arī to, ka šis fakts līdz šim ir pētīts minimāli. Autors saskata iespēju, ka trasējamības uzlabošana var uzlabot testēšanu. Tika izvirzīts priekšlikums, ka trasējamību var uzlabot ar atbalstošas programmatūras (rīka) palīdzību. Autors nodefinēja šāda rīka sākotnējo koncepciju, kā arī noorganizēju rīka TraceIt izstrādi un eksperimentālu ieviešanu vairākos reālos programmatūras projektos, ko atspoguļoja publikācijās un referātos starptautiskās konferencēs. Ņemot vērā to, ka jau rīka koncepcijas izstrādes laikā kļuva skaidrs, ka minētais trasējamības risinājums varētu atbalstīt trasējamību ne tikai testēšanas procesā, bet arī citos programmatūras izstrādes procesos, rīka eksperimentālā ekspluatācija tika veikta dažāda tipa programmatūras projektos. Pēc pirmajos izmēģinājumos iegūtajiem veiksmīgajiem rezultātiem veica pētījumus par to, kādas trasējamības īpašības ir tieši testēšanas procesam, un par šo tēmu sagatavoja publikācijas un referēja starptautiskās

konferencēs. Papildus tam, apzināja trasējamības praksi citos Latvijas IT uzņēmumos, kā arī atbalstīja jaunu projektu interesi izmantot rīku TraceIt projektu darbu atbalstam. Pēc rīka plašākas izmantošanas 3 gadu garumā vairāk kā 15 projektos apzināja tā lietotāju vērtējumu par rīku, kā arī ir sagatavojis prasības tam, kā programmatūras projektā izmantojamajiem rīkiem ir jāatbalsta trasējamība.

1.3. Saistītie materiāli

Promocijas darbs balstās iepriekšējos gados veiktajiem pētījumiem, kuru rezultāti ir atspoguļoti publikācijas esot kā autoram vai līdzautoram. Šādi darbi pa tematiskām grupām:

- Programmatūras testēšana: [2], [3], [4]
- Trasējamība: [6], [12], [1], [7], [8], [11]
- Testēšana un trasējamība: [9], [10]
- Programmatūras kvalitāte: [5]

1.4. Pamatdefinīcijas

Testēšana – programmatūras analīzes process ar mērķi novērtēt tās atbilstību iepriekš definētām prasībām.

Trasējamība – spēja izsekot saitēm starp dažādiem informācijas pierakstiem, kas eksistē programmizstrādes projektā.

2. Testēšanas procesa problemātika

2.1. Nodaļas mērķi

Šajā nodaļā ir īsumā aprakstīts viens no programmatūras dzīves cikla procesiem – testēšana. Ir norādīti galvenie testēšanas veidi, kā arī analizētas problēmas, kas parādās testēšanas praktiskās pielietošanas gadījumos. Nodaļas ietvaros tiek analizēta testēšanas prakse Latvijā, kā arī praktiskā pieredze testētāju sagatavošanā un testēšanas projekta grupu izveidošanā IT uzņēmuma ietvaros. Mērķis ir sniegt priekšstatu par testēšanas problemātiku un veiktajiem autora pētījumiem šajā jomā.

2.2. Pamatnostādnes

Kopā ar programmatūras izstrādes pamatprincipu izveidošanos un dažādu metodoloģiju attīstīšanos, programmatūras testēšana ir nostabilizējusies kā viena no neatņemamām programmatūras izstrādes sastāvdaļām. Tās veidu nosaka mērķis, uzdevuma nostādne, pieejamā informācija, izvēlētās metodes u.c.. Testēšana ir viens no kvalitātes novērtējuma mehānismiem [17].

Definīciju daudzveidība

Programmatūras testēšanas mērķis atbilstoši definīcijai (nodaļa 1.4.) ir novērtēt programmatūras atbilstību prasībām. Šī nav vienīgā definīcija, un nozares literatūrā un standartos līdzās pastāv arī virkne citu definīciju (skat. 1.tabulu), kuras vienlīdz labi atbilst reālajai dzīvei – konkrētiem programmatūras testēšanas gadījumiem.

1. tabula. Programmatūras testēšanas definīcijas.

Nr.	Definīcija	Avots
1.	Programmatūras testēšana ir populāra riska pārvaldības stratēģija. Tā tiek izmantota, lai pārbaudītu, ka ir ievērotas funkcionālās prasības.	[89]
2.	Testēšana ir plānošanas, sagatavošanas un mērīšanas process ar mērķi noskaidrot informācijas sistēmas raksturlielumus un demonstrēt atšķirību starp reālo un pieprasīto statusu.	[85]
3.	Tehniska darbināšana, kas sastāv no produkta, procesa vai pakalpojuma vienas vai vairāku īpašību noteikšanas saskaņā ar specificēto procedūru.	[74]
4.	Sistēmtestēšana ir visas integrētās sistēmas pārbaude, lai pārliecinātos, vai tā dara to, ko vēlas lietotājs, kā arī, vai tā atbilst specifikācijai.	[103]
5.	Testēšana ir kļūdu meklēšanas process.	[83]
6.	Testēšana ir projektēšanas, atkļūdošanas un testu izpildes darbība. Tās mērķi ir: - Sniegt programmētājiem informāciju, ko viņi var izmantot kļūdu novēršanai; - Sniegt vadībai informāciju, kas tai nepieciešama objekta lietošanas riska racionālam novērtējumam;	[33]

	- Iegūt objektu, kas konkrētiem gadījumiem tiek apliecināts kā bez kļūdām esošs; - Sasniegt testējamu projektējumu; projektējumu, kuram ir viegli pārbaudāms derīgums, īstums un ko ir viegli uzturēt. - Mēģināt atrast objekta problēmas attiecībā pret deklarētām un nedeklarētām prasībām; arī tiek saukts pat "programmatūras lauzīšanu" - Pārbaudīt objekta derīgumu, tas ir, parādīt, ka tas strādā.	
7.	Testēšana ir viss, kas ietver vismaz četras šādas aktivitātes: konfigurēšana (...), darbināšana (...), novērošana (...) un novērtēšana (...).	[84]
8.	Programmatūras darbināšana ar testpiemēriem. Testam ir divi izteikti mērķi: atrast kļūdas un demonstrēt pareizu darbošanos.	[81]
9.	Sistēmtestēšana – programmatūras vai aparatūras izstrādes fāze, kas meklē kļūdas vispārējā un konkrētajā sistēmas uzvedībā, funkcijās un atbildēs testa ietvaros kā visa darbināšana reālistisku lietošanas scenāriju ietvaros. Dažādās sistēmas darbināšanas tiek veiktas, kad sistēma ir pilnībā integrēta.	[39]
10.	Programmatūras testēšana ir programmatūras analizēšanas vai darbināšanas process ar mērķi atrast kļūdas.	[51]
11.	Testēšana ir paralēls inženierijas dzīves cikla process, izmantojot un uzturot testēšanas rīkus, lai mērītu un uzlabotu testējamās programmatūras kvalitāti.	[49]
12.	Programmatūras testēšana - Programmatūras vienuma analīzes process, lai konstatētu atšķirības starp esošajiem un prasītajiem apstākļiem (t.i. atrastu kļūdas) un novērtētu programmatūras vienumu iezīmes.	[67]

Galvenā raksturojošā īpašība, kas testēšanu atšķir no citiem kvalitātes nodrošināšanas mehānismiem, ir gatava produkta vai produkta daļas pārbaude. Lai arī intuitīvi var pieņemt, ka ekonomiski izdevīgāk ir nepieļaut kļūdu veidošanos un implementēt programmu uzreiz bez kļūdām, nav sastopamas norādes zinātniskajā literatūrā par to, ka kādam tomēr būtu izdevies iztikt bez testēšanas (pie nosacījuma, ka programma beigās ir arī jālieto). Tajā pašā laikā diezgan regulāri tiek konstatēts un ieteikts, ka jāpielieto dažāda veida kvalitātes pasākumi (piemēram, versiju vadība un apskates) visā programmatūras dzīves cikla laikā [121], kā arī testēšana jāuzsāk pēc iespējas agri – tiklīdz ir pieejama kaut mazākā pārbaudāmā vienība. Tam pamatojums izriet no tā, ka katra izmaiņa, kas pēc sevis prasa virkni citu darbu un izmaiņu, izmaksā vairākas reizes dārgāk nekā tad, ja tā būtu konstatēta pietiekami savlaicīgi.

Testēšana kā atsevišķa izteikta aktivitāte vēsturiski nostiprinājās aptuveni 20.gadsimta 70-to gadu sākumā, kad tā tika atdalīta no programmēšanas, tādējādi atšķirot testēšanu no atkļūdošanas, kas faktiski ir daļa no implementācijas procesa [34]. Kā pirmā būtiskā grāmata ir minama 1979.gadā iznākusī G.Maiersa „Programmatūras testēšanas māksla" [109], kas iezīmēja būtiskākos virzienus tik veiksmīgi, ka arī trīsdesmit gadus vēlāk pamatjēdzieni un to būtība praktiski nav mainījušies (tam labs apliecinājums ir tas, ka uz to labprāt atsaucas arī pēdējos gados

publicētajos testēšanai veltītajos rakstos un grāmatās). Protams, līdz ar programmatūras attīstību ir parādījušās jaunas problēmas, tiek praktizēti jauni projektu pārvaldības principi, bet, piemēram, tādas lietas kā melnās kastes un baltās kastes testēšana ir arī mūsdienu testēšanai piederīgi jēdzieni.

Laiku pa laikam paceļas jautājums par to, vai testēšana ir brīva amatnieciska aktivitāte, vai arī tā balstās uz zinātniskiem pamatiem [35]. Ir testēšanas metodes, kuru būtība seko no grafu teorijas, automātu teorijas, vai arī pielieto pilnās pārlases principus – par šādu pieeju izmantošanu programmas īpašību noteikšanai īpašas šaubas neparādās. Tomēr testēšanā daudzas lietas balstās uz novērojumiem un vispārinājumiem. Piemēram, skaitlisku intervālu apstrādē tipiska programmēšanas kļūda var būt stingrās nevienādības sajaukšana ar nestingro vai otrādi, un tādēļ ir iesakņojies princips, ka skaitlisku intervālu gadījumos jāpārbauda ir robežgadījumi. Ja atkarībā no ievadītās vērtības ir iespējami vairāki tālākās darbības ceļi, tad testēšanā ir jāpārbauda katrs no tiem. Nenoteiktības faktoru testēšanas pieejā ievieš notikums, kad problēma tiek novērota nevis kā kļūda programmas kodā, bet gan funkcionalitātē – vai tiešām tā ir kļūda? cik stipri tā ietekmē programmas funkcionalitāti? kāds ir labākais veids to aprakstīt? Ja savlaicīgi nav nodefinēti nosacījumi, kā atšķirt korektu situāciju no kļūdainas, tad testēšanas process kļūst atkarīgs no subjektīviem vērtējumiem. Programmatūras testēšanā būtiska loma ir tendenču analīzei, lai izvēlētos atbilstošākās metodes un koncentrētu analīzi uz konkrētiem programmas apgabaliem. Piemēram, ja tiek novērots, ka kādā atsevišķā vietā rodas problēmas ar saraksta pēdējā elementa dzēšanu, tad pilnīgi noteikti jāpārbauda ir arī citās programmas vietās saraksta pēdējā elementa dzēšana.

Testēšanai programminženierijā ir atsevišķi elementi, kas ir kopīgi ar testēšanu citās nozarēs, tomēr katrai ir sava specifika [36]. No nozares uz nozari atšķiras tas, attiecībā pret ko salīdzina. Tas var būt robežās no stingri reglamentētiem kritērijiem (piem., būvniecībā) līdz absolūti subjektīviem (mākslas nozarēs). Arī programmatūras testēšanā atkarībā no tā, ko tieši un kādiem mērķiem pārbauda, kritēriji var būt vairāk objektīvi vai subjektīvi.

Ja aplūkojam testēšanas jeb pārbaudes problēmu plašākā nozīmē, tad var identificēt paralēles ar filozofiska rakstura jautājumu par to, vai kāda teorija ir laba, vai nē. Piemēram, K.Poppers [120] izvirza hipotēzi, ka īstas teorijas tests ir mēģinājums to padarīt to nederīgu, parādīt tās nepiemērotību. Ja ir tādas teorijas, kas izskaidro pilnīgi visu, tad tās nav derīgas (tādas ir dažas psiholoģijas teorijas). Līdzīgi arī programminženierijā – ja programmu var notestēt, tas nozīmē, ka tā ir labi veidota programma (tas neizslēdz iespēju, ka testa rezultāts var būt arī negatīvs). Ja programmu mēs nevaram notestēt vai nu tās neskaidro prasību dēļ, tad vai arī implementācijas īpatnību dēļ, to nevar ierindot starp laba tipa programmām. Testēšanas mērķim nav jābūt programmas darba attaisnošanai, bet gan jāmēģina atrast tajā problēmas.

Lai arī sākotnēji var šķist, ka programmatūras testēšanas nozare ir lielā mērā nostabilizējusies, un jauni risinājumi pēdējos gados nav novērojami, tomēr testēšanai ir jāpielāgojas jaunajām programmizstrādes tendencēm [15].

Testēšana un standarti

Testēšanu kā kvalitātes aktivitāti reglamentē virkne IT nozares standartu. Saskaņā ar standartu *ISO/IEC 12207:1995 Information technology - Software life cycle processes* [76] (jeb 2002.gadā apstiprināto Valsts standartu *LVS ISO/IEC 12207:2002 Informācijas tehnoloģija - Programmatūras dzīves cikla procesi*) tai ir jānotiek kopā ar programmas koda izveidi, kā arī pie produkta nodošanas pasūtītājam. Pielāgojot standartu lietošanai konkrētā organizācijā vai projektā, testēšanu var definēt kā vienu no pamatprocesiem. Detalizēti testēšanas prasības ir atspoguļotas arī standartā *J-Std-016-1995 EIA/IEEE Interim Standard for Information Technology - Software Life Cycle Processes - Software Development Acquirer-Supplier Agreement* [69], kas pamatā ir orientēts uz ISO/IEC 12207 definētā izstrādes procesa detalizāciju. Tieši J-Std-016 ir daudzās Latvijas IT organizācijās kļuvis par praksē izmantoto standartu dokumentācijas komplekta satura un formas noteikšanai. Lai arī ir sastopamas norādes, ka šis standarts netiks turpmāk attīstīts, to kāda IT organizāciju kopa var attīstīt savos ietvaros, pielāgojot jaunām kvalitātes tendencēm. Testēšanai tas definē trīs savstarpēji saistītus dokumentus: testēšanas plānu, testu aprakstu un testēšanas pārskatu. Jāatzīmē, ka papildus iepriekšminētajiem ir arī standarts *1465-1998 IEEE (12119:1998 ISO/IEC) Information Technology - Software Packages - Quality Requirements and Testing* [70], kas izvirza specifiskās prasības tieši pakotņu programmatūrai.

Latvijā kopš 1996.gada ir divi standarti, kas ir tiešā veidā attiecināmi uz testēšanu informācijas tehnoloģiju jomā (pārējās testēšanas jomās to kopskaitā ir vairāki desmiti). LVS 70:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras testēšanas dokumentācija un *LVS 73:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras vienībtestēšana* [99]. Abi ir atbilstošo IEEE standartu adaptējumi: attiecīgi ANSI/IEEE Std 829-1983 (pašlaik jaunākā versija - 829-1998) un ANSI/IEEE Std 1008-1987 [68]. Pirmais definē pieeju tam, kādai ir jābūt testēšanas dokumentācijai programmatūras projektos, savukārt otrais nosaka vienībtestēšanas (programmas funkciju, moduļu pārbaudes) procesu. Jāsaka, ka mūsdienu programminženierijas metodes ir visai daudzveidīgas, un vairumā gadījumu reālos projektos nav racionāli burtiskā veidā pielietot LVS 73:1996. Papildus iepriekšminētajiem ir arī standarts *LVS 71:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras verifikācijas un validācijas plāns* [97], kas arī ir atbilstošā IEEE standarta adaptējums. Tas definē testēšanu kā vienu no validēšanas un verificēšanas metodēm. Praksē bieži pasūtītāja un izpildītāja vienošanās par izveidojamo dokumentāciju neparedz testēšanas dokumentāciju, vai arī tā tiek noteikta ļoti konspektīva.

Programmatūras testēšanas standartu izstrādei viens no šķēršļiem ir problēma, ka ne visas testēšanas pieejas vai metodes ir formalizētas. Tādēļ nereta ir situācija, ka dažādās organizācijās ar vienu un to pašu terminu saprot atšķirīgas aktivitātes, vai arī viens un tas pats darbs tiek nosaukts dažādi. Autors ir novērojis mazliet cita vieda problēmu, kas saistās ar valodu atšķirībām – vairākos gadījumos Latvijā tiek atšķirīgi tulkoti angliskie kvalitātes termini. Zināms progress testēšanas vienoto nostādņu nostiprināšanā parādījās 20.gs deviņdesmitajos gados kopā ar standartiem *BS 7925-1:1998 Software testing - Vocabulary* [41] un *BS 7925-2:1998 Software testing - Software component testing* [42]. Pirmais minētais kompakti definē vairāk kā divsimt ar testēšanu saistītu terminu. Turpretim otrs ir formalizējis testēšanas metodes, kas pirms tam bija aprakstītas pārsvarā tikai akadēmiska rakstura literatūrā. Atšķirībā, piemēram, no mehānisko vai ķīmisko testu izpildes rūpniecībā, kur varētu būt aprakstīts testēšanas algoritms un pārbaudes kritēriji, programmatūras testēšanas metodes apraksta tikai galvenos principus, bet konkrēto pielietošanu ir jāizdomā personai, kas šo testēšanu veic.

Tradicionāli struktūrvienību, kurā notiek testēšana, sauc par testēšanas laboratoriju. Latvijā to darbību nosaka standarts *LVS EN ISO/IEC 17025:2001 Testēšanas un kalibrēšanas laboratoriju kompetences vispārīgās prasības* (agrākais starptautiskais nosaukums - ISO /IEC Guide 25 jeb Eiropai - EN 45001). Tomēr informācijas tehnoloģiju joma ir tik specifiska, ka ir arī izstrādāts speciāls ceļvedis šī standarta piemērošanai informācijas tehnoloģiju un telekomunikāciju testēšanas laboratorijām - *ISO/IEC TR 13233:1995 Information technology - Interpretation of accreditation requirements in ISO/IEC Guide 25 - Accreditation of Information Technology and Telecommunications testing laboratories for software and protocol testing services* [77]. Galvenās problēmas ir ar tādiem jēdzieniem, kā mērinstrumenti un to kalibrēšana. Kā jau iepriekš bija minēts, programmatūras jomā tādu vienkārši nav. Izņēmums var būt gadījumi, ja IT jomā ir jāpārbauda nevis funkcionāla rakstura jautājumi, bet gan tādi aspekti, kā ātrdarbība, tipveida situācijas atkārtošanās biežums u.tml. Mērinstruments šādā testēšanā ir programma, kurai jābūt pierādītai korektai funkcionalitātei (kas ir ievērojami sarežģītāks uzdevums nekā programmas testēšana). Cita problēma ir standartu noteiktā laboratoriju salīdzinošā testēšana (testē vienu un to pašu objektu vairākas laboratorijas) - katra programmatūras testēšanas laboratorija veic savus īpašus uzdevumus, kas ir veicami tieši konkrētā projekta ietvaros. Galvenokārt iepriekšminēto iemeslu dēļ programmatūras jomā faktiski nav iespējams sastapt testēšanas laboratorijas, kas būtu akreditētas saskaņā ar iepriekš minētajiem standartiem. Tomēr viens no risinājumiem darba kvalitātes uzlabošanai ir vispārējās kvalitātes sistēmas ieviešana, piemēram, saskaņā ar standartu ISO 9000:2001 [75].

2.3. Testēšanas veidi

Testēšanu ir iespējams klasificēt pēc atšķirīgiem kritērijiem atkarībā no tā, kādam mērķim attiecīgais dalījums tiek izmantots. Klasiski visbiežāk praktizētais ir

iedalījums pēc koda pieejamības – programmu var aplūkot kā tekstu (testētājam kods ir pieejams) vai kā darbināmu algoritmu (testētājam kods nav pieejams, bet ir informācija par iecerēto programmas uzvedību). Šāds dalījums ir tradicionāls testēšanā kā akadēmiskā disciplīnā, bet tas absolūti neņem vērā reālo projektu aspektus tādus, kā saistība ar programmatūras izstrādes procesiem un organizatoriskajiem jautājumiem. Komerciāla rakstura projektos būtisku lomu spēlē ekonomiskie apsvērumi, kas veido visdažādākā veida plānošanas un izpildes stratēģijas. Reālā dzīvē reti tiek izmantota pilnībā tieši viena izvēlētā metodika. Testēšanas norise sadarbībā ar citiem programmatūras procesiem ir būtiska, lai iegūtu kvalitatīvu programmatūras produktu.

Praksē empīriski nospraustie dalījumi mēdz nebūt ideāli tīri - bieži ir sastopams metožu vai organizatorisko pieeju apvienojums. Katru no dalījumiem var aplūkot kā atsevišķu dimensiju - tās ir neatkarīgas no pārējām, un ikvienu testēšanas aktivitāti var raksturot ar šo dimensiju vērtībām. Autora skatījumā tipiskākās dimensijas ir:

- programmas koda pieejamība,
- programmas lietošanas kritēriji,
- organizatoriskās attiecības un
- saistība ar programmatūras izstrādes etapu.

Testēšana un koda pieejamība

Atkarībā no koda pieejamības testēšanu var iedalīt strukturālajā un funkcionālajā testēšanā. Nav iespējams pateikt, kura no šīm pieejām ir labāka, jo katra no tām vadās pēc saviem kļūdu meklēšanas kritērijiem un ļauj identificēt atšķirīga tipa problēmas. Literatūrā ir sastopami abu pieeju aizstāvji. Par labu strukturālajai pieejai kalpo argumenti par nespecificētas vai nevēlamas darbības identificēšanas iespēju. Savukārt, funkcionālā pieeja tiek aplūkota kā vienīgā pielietojamā reālos projektos, jo tā pārbauda, ko programma tiešām dara no sagaidāmās funkcionalitātes viedokļa.

Funkcionālā pieeja

Galvenais nosacījums šai pieejai ir tas, ka testētājam ir iespēja novērtēt programmas darbību, un kods parasti nav pieejams (šī iemesla dēļ šādu testēšanu bieži sauc par melnās kastes testēšanu). Programma tiek uztverta kā funkcija, kas noteiktām ievadvērtībām atgriež rezultātu. No vienas puses, ir labi saprotams, ka visas iespējamās situācijas pārbaudīt nav iespējams, jo pat tikai pāris ievadvērtību gadījumā pārbaudāmo gadījumu skaits var sasniegt apjomu, kuru pārbaudīšanai būtu jāpatērē laiks, kas būtiski pārsniedz testēšanai un pat visam projektam atvēlēto laiku. No otras puses, praksē veiksmīgi tiek testētas programmas ar lielu skaitu dažādu ievaddatu. Populārākā pieeja balstās uz principa, ka sistēma līdzīga tipa ievaddatus apstrādā pēc vienotiem principiem. Testēšanas uzdevums ir saprast gadījumus, kuros ir spēkā vienotie principi, kā arī identificēt visus izņēmuma gadījumus. Funkcionālās

testēšanas priekšrocība ir arī tajā apstāklī, ka tai domātos testus var sagatavot vēl pirms ir pabeigta koda implementācija. Testi būs neatkarīgi no programmēšanas valodas un minimāli atkarīgi no izmantotās izstrādes metodikas.

Tipiskākās testēšanas metodes ir uzskaitītas standartā BS-7925 [42]. Uz funkcionālo testēšanu attiecas šādas:

- sadalīšana ekvivalences klasēs,
- robežvērtību analīze,
- stāvokļu pārejas testēšana,
- cēloņu - seku grafu veidošana,
- sintakses testēšana un
- gadījumiska testēšana.

Galvenā funkcionālās testēšanas priekšrocība – tā paaugstina pārliecību par programmas darbības atbilstību prasībām. Tomēr jāatzīst, ka funkcionālā testēšanas pieejai nav drošu līdzekļu, kā atrast nespecificētus speciālos gadījumus, piemēram, negodprātīgi iestrādātus izņēmuma gadījumus, kas kļūst aktīvi tikai konkrētā situācijā.

Strukturālā pieeja

Testēšanas strukturālo (sauktas arī par baltās jeb stikla kastes metodēm [81]) metožu pamatā ir priekšnosacījums, ka testētāju rīcībā ir programmas kods. Pat vēl vairāk, lielākā daļa no metodēm neaplūko atbildi uz jautājumu par to, vai programma strādā atbilstoši iecerētajai funkcionalitātei, bet gan pārbaudi balsta tieši uz koda īpašībām. Mērķis ir iegūt programmu, kas tehniski pareizi apstrādā programmas datus, izņēmuma situācijas, nesatur labās programmēšanas prakses nepieļautas konstrukcijas utt. Strukturālo testēšanas metožu būtiska īpašība ir to stingrais teorētiskais modelis, kas balstās uz programmu kā algoritma pierakstu. Proti, imperatīvā programmēšanas valodā rakstītu programmu ir iespējams aplūkot kā orientētu grafu ar dažāda tipa īpašībām virsotnēs un šķautnēs. Tas ļauj precīzi ieplānot un rezultātā saprast, kas tieši programmā tiks un tika pārbaudīts. Cita būtiska strukturālās testēšanas iezīme ir iespēja veikt skaitliskus novērtējumus un mērījumus. Ir sastopamas dažādas funkcionālās metodes, kuras ir vai nu dabiski izrietošas no grafa pamatīpašībām (piemēram, katras komandas pārbaude), vai arī balstās uz specializētām programmas īpašībām (piemēram, mainīgo izmaiņu novērtējums pie dažāda tipa ievaddatiem). Standarts BS-7925 norāda šādas strukturālās testēšanas metodes:

- komandu testēšana,
- zaru/lēmumu testēšana,
- datu plūsmas testēšana,
- zaru nosacījumu testēšana,
- zaru nosacījumu kombināciju testēšana,
- modificētā nosacījumu lēmumu testēšana un
- LCSAJ (*Linear Code Sequence And Jump*).

Strukturālo metožu pielietošana ir darbietilpīgs process. Reālos pielietojumos metodes tiek izmantotas gadījumos, kad ir īpaši augstas drošības prasības – aviācijā, dzelzceļa, automašīnu būves, enerģētikas un militārajās nozarēs [125]. Nozares, kur šādu prasību nav, izvēlas ekonomiski izdevīgākas testēšanas metodes – no funkcionālo metožu grupas. Galvenā strukturālās testēšanas priekšrocība – tā ir orientēta uz koda kļūdu atrašanu.

Papildus iepriekšminētajām pastāv strukturālo un funkcionālo metožu apvienojums, kas dažkārt tiek dēvēts par pelēkās kastes testēšanu. Ar šo terminu saprot vismaz divas dažādus testēšanas procesus:

1) Testus veido ar funkcionālajām metodēm, un testu izpildes laikā seko līdzi tam, kādas koda daļas tika darbinātas un kādas – nē. Atkarībā no testēšanai izvirzītajiem mērķiem var pieņemt lēmumu par papildus testu sagatavošanu vai koda optimizāciju.
2) Testētājs ir informēts par koda uzbūvi, kādām atsevišķām implementācijas detaļām, un šīs zināšanas var palīdzēt būvēt funkcionālos testus [84].

Līdz ar objektorientētās programmēšanas attīstību parādījās jautājums par īpašu testēšanu tieši šāda tipa programmām. Galvenā atšķirība ir apstāklī, ka visa funkcionalitāte atrodas objektos, un objekti programmas darbības laikā savstarpēji komunicē. Testu akcents varētu tikt vērsts uz objektu saskarņu testēšanu. Papildus problēmas parādās gadījumos, ja tiek izmantotas tādas objektorientācijas iespējas, kā mantošana un polimorfisms [38].

Testēšana un lietošanas kritēriji

Testēšana visbiežāk notiek atbilstoši programmatūras prasībām. Vēsturiski ir nostiprinājies dalījums: funkcionālās un nefunkcionālās prasības [121], kur pie pēdējā minētā pieder tāda prasību kategorijas, kā veiktspēja, lietojamība un drošība. Mazliet mānīgs ir pats termins „nefunkcionālās prasības". Tas minētās prasības it kā padara otršķirīgas, lai gan attiecīgās programmatūras izmantošana var balstīties tieši uz to, ka programma pienācīgi aizsargā informāciju, vai arī ir īpaši ērti lietojama. Lai arī tālāk ir minētas katra veida testēšanas galvenās īpašības, retos gadījumos testēšana pārbauda tikai vienu no tām. Parasti programmas pārbaude ietver vairākus no minētajiem kritērijiem, piemēram, lietotāja saskarnes elementu pārbaude var būt uz robežas starp funkcionālo un lietojamības testēšanu. Specializētām programmām var tikt identificēti savi specializēti testēšanas veidi tādiem kritērijiem, kā saderība, lokalizācija un internacionalizācija, programmas vizuāli mākslinieciskā vērtība utt.

Funkcionalitātes testēšana

Visbiežāk ar vārdu „testēšana" tiek domāts tieši par funkcionālo testēšanu. Klasiskā situācijā programmatūras izveides mērķis ir veikt kādas iepriekš definētas

funkcijas, un testēšana attiecīgi ir šo funkciju implementācijas pārbaude. Šeit ir lietojamas gan strukturālās, gan funkcionālās testēšanas metodes.

Veiktspējas testēšana

Veiktspējas testēšana pārbauda visus tos jautājumus, kas saistās ar programmas spēju apstrādāt noteikta apjoma datus un sniegt atbildi noteikta laika ietvaros pie noteiktiem ārējiem apstākļiem. Šādā testēšanā pats būtiskākais ir radīt atbilstošus apstākļus un veikt mērījumus. Veiktspēja ir būtiska sistēmām, kuru funkcionēšanai izšķiroša nozīme ir liela datu apjoma apstrāde un savlaicīgas atbildes sniegšana. Ne vienmēr šādas testēšanas rezultātā atklāto problēmu novēršanai ir jāmaina programmas kods, bet gan ievērojamus uzlabojumus var iegūt, ja konfigurē procesora darbību, operatīvo atmiņu, diskus un tīklu [108]. Veiktspējas testēšana ir aktualizējusies tīmekļa programmatūras attīstības kontekstā – ja servera pusē notiek datu apstrāde, dinamiska formu ģenerēšana un liela apjoma informācija tiek pārsūtīta caur tīklu, klienta pusē atbildes gaidīšana var aizņemt ilgu laiku. Veiktspējas testēšanas paveidi ir [121]: slodzes testi, ātrdarbības testi, apjoma testi, stresa testi un izturības testi.

Lietojamības testēšana

Lietojamības testēšana aplūko programmatūru ne tik daudz no funkcionālā viedokļa, bet gan to, cik ērti un saprotama būs programma lietotājam. Šāda veida testēšanai visizplatītākā un atbilstošākā ir pieeja ar lietotāju iesaistīšanu testēšanas procesā [127]. Būtiskākā atšķirība no funkcionalitātes testēšanas ir tāda, ka ne visas prasības, attiecībā pret kurām notiek testēšana, ir iepriekš definētas. Lai arī ir dažādas institūcijas un nozares eksperti ir sagatavojuši vadlīnijas dažāda tipa programmu lietojamības uzlabošanai, lietotāju testēšanas laikā var izrādīties, ka ir nepieciešams kaut kas citādi, un tieši tad lietotāju viedoklis būs izšķirošais nevis formālas vadlīnijas.

Drošības testēšana

Drošības testēšanas uzdevums ir pārliecināties, ka sistēmas informācija ir aizsargāta atbilstoši definētajām prasībām (piemēram, ikviens drīkst skatīties, bet drīkst labot tikai atsevišķi lietotāji, vai arī – piekļūšana informācijas apskatei tikai ar īpaša autentifikācijas mehānisma palīdzību). Lai arī drošības prasības varētu būt definētas un derīgas ilgstošam laikam, metodes ir jābalsta uz jaunākajām zināšanām par sistēmu nograušanas, uzlaušanas, darbības traucēšanas un spiegošanas metodēm [105].

Galvenā īpatnība, kas atšķir drošības testēšanas rezultātus no funkcionālās testēšanas rezultātiem, ir tā, ka drošības testēšanas atzinums iepriekš neparedzamā laikā var kļūt nederīgs, jo būs atklātas jaunas metodes, kā pārkāpt aizsardzības mehānismus. Katra jauna ļaundaru izstrādātā ielaušanās metode ir jauns tests drošības

testēšanai. Problēma reducējas uz to, ka testētājiem ir jābūt pastāvīgi informētiem par aktuālajām drošības pārkāpšanas metodēm un līdzekļiem, kā to novērst.

Organizatoriskais princips testēšanā

Testēšanas iedalījums pēc organizatoriskā principa izveidojās apstākļos, kad programmatūras izstrādē sāka nostiprināties iesaistīto personu un organizāciju specializācija, kā arī pieauga kvalitātes kritēriji un attīstījās sava veida kvalitātes kultūra (t.i. paveiktais darbs ir jāpārbauda, un vislabāk, ja to veic kāds cits). Testēšanas organizatorisko veidu izdalīšana īpaši nostiprinājās laikā, kad attīstījās komerciālās attiecības programmatūras izstrādē.

Galvenā pazīme, pēc kuras veido iedalījumu, ir testētāja līdzdalības līmenis testējamās programmas izstrādē. No šejienes ir divi pretstatījumi: autora testēšana un neatkarīgā testēšana. Neatkarīgajai testēšanai var piemist dažādi paveidi, piemēram, programmizstrādes projekta ietvaros neatkarība ir gadījumos, ja programmu testē programmas koda autora kolēģis, bet organizāciju līmenī neatkarīga testēšana ir tāda, ja to veic ne programmatūras pasūtītājs, ne ražotājs, bet gan pieaicināta organizācija objektīva vērtējuma sniegšanai.

Lai arī kāda būtu testēšanas organizatoriskā puse, tas maz ietekmē testēšanā pielietotās metodes. Teorētiski tās ir savstarpēji nesaistītas lietas. Tomēr praksē ir tā, ka neatkarīgās testēšanas ietvaros vairāk tiek izmantotas funkcionāla rakstura testēšanas metodes, jo ir mazākas iespējas analizēt programmas kodu – tas ir gan tehniski grūtāk, gan arī bieži pastāv komerciāli ierobežojumi. Neatkarīgās testēšanas nozīmi nosaka trīs faktori: psiholoģiskie, organizatoriskie un trešās personas loma.

Psiholoģiskie faktori ir vieni no svarīgākajiem. Motivējošie faktori darbā var būt visdažādākie - apziņa par labi paveiktu darbu, materiāls atalgojums, savu spēju realizēšana, labas attiecības ar kolēģiem, kārtības ieviešana utt. Programmas izveidē viens no būtiskākajiem ir ieceres pārvēršana reālā darbināmā sistēmā. Turpretim, testēšanā - labi paveikts darbināmās sistēmas precīzas analīzes un salīdzināšanas darbs. Izrādās, ka ar abām motivācijām vienlaikus vieni un tie paši cilvēki strādāt praktiski nevar. Ir novērots, ka programmētāji ir slikti piemēroti sava darba testēšanai [62]. Tās ir dabiskas sekas tam, ka viņi ir sistēmas veidotāji. Kad programmētājs testē programmu, ko viņš pats ir veidojis, viņš testē produktu, kam viņš pats tic (un ticībai ir lielāks spēks par spēju domāt objektīvi). Ja būtu pretēji, attiecīgā komponente nebūtu nonākusi līdz testēšanai. Tādēļ testēšanu ir nepieciešams veikt citai personai, lai būtu neatkarīgs skatījums bez konfliktējošas motivācijas. Testēšana nav nekas nedz sliktāks, nedz labāks par programmēšanu. Tā pozitīvā veidā izmanto cilvēku dabisko īpašību cita cilvēka darbus vērtēt kritiskāk nekā savējos. Ļoti būtiski ir uzturēt pozitīvu testēšanas psiholoģiju. Nav jāuzskata, ka testu mērķis ir pierādīt, ka izstrādātājs ir absolūti neprasmīgs vai pat ļaunprātīgs. Galvenais mērķis ir nodrošināt kvalitatīvu produktu, un tas ir panākams ar testu palīdzību.

Organizatoriskie faktori parasti parādās projekta iekšienē. Kvalitātes nodrošināšanas, tajā skaitā, testēšanas, personālam ir jābūt pēc iespējas neatkarīgam no projekta izstrādes jautājumiem. Galvenais iemesls ir, ka tipiska projekta vadītāja galvenās rūpes ir termiņi un budžets, un pie programmatūras izstrādē ierastās steigas savlaicīgi nodot sistēmu, kvalitāte bieži tiek nobīdīta malā. Tādēļ testēšanas personālam ir ieteicams būt nedaudz nošķirtiem no projekta vadības, lai minimizētu atkarību no izstrādi ierobežojošajiem faktoriem.

Trešās personas lomai ir nozīme pasūtītāja un piegādātāja nesaskaņu kontekstā, augsta riska situācijās un gadījumos, kad ir nepieciešami kompetenti testēšanas speciālisti. Neatkarīgai testēšanai ir uzticēts novērtēt, vai piegādājamais produkts atbilst izvirzītajām prasībām [13], [14]. Testēšanas rezultāti pieder pilnībā pasūtītājam, un testētāji uzņemas pilnas konfidencialitātes saistības. Šī situācija ir līdzīga finanšu vai arī jebkuram citam auditam, kur specializēta organizācija tiek pieaicināta ekspertīzes veikšanai. Visneatkarīgākā gadījumā, kad testētāji ir no juridiski citas organizācijas, šim darbam ir stipri izteiktas projekta iezīmes - pasūtītāja un piegādātāja attiecības, kuru pamatā ir līgums, īpašs testēšanas plāns, dokumentācija un savādāki darba apstākļi, tad projekta gadījumā valda lielāka sasaiste ar izstrādes procesu un apstākļiem. Ir aktuāli jautājumi par testētāju ciešu komunikāciju ar projekta darbiniekiem, īpaši sistēmanalītiķiem, ar projekta attīstības etapiem, kā arī jautājumi saistībā ar detalizētu problēmu apstrādi un īpašu testēšanas aktivitāšu organizēšanu.

Populāri jēdzieni neatkarīgās testēšanas kontekstā ir ārpakalpojuma vai attālināta testēšana. Pirmais izteikti raksturo tās situācijas, kad noteiktu testēšanas apjomu veic kāda ārēja organizācija. Tas var notikt gan paralēli projekta izstrādei, gan arī būt kā atsevišķs testēšanas pasākums. Savukārt, attālinātā testēšana attiecas uz gadījumiem, kad ģeogrāfiski testēšanas komanda neatrodas blakus programmētājiem, kas izvirza papildus nosacījumus pret komunikāciju starp abām komandām [113].

Viens skatījums, kā būtu iespējams aplūkot neatkarīgo testēšanu, ir tas, cik lielā mērā testētāja darbs ir koordinēts ar izstrādātāja aktivitātēm un otrādi. Te var būt runa par to, vai abas puses savstarpēji sazinās, vai arī strādā pilnīgi nošķirti (gan ģeogrāfiski, gan uzdevumu izpildes ziņā). Stipra nošķirtība var pastāvēt gadījumos, kad organizācija A ir pasūtījusi no organizācijas B programmu, un pēc tam A pēc savas iniciatīvas pasūta organizācijai C notestēt piegādāto programmu. Šāda situācija var būt īpaši aktuāla, ja pasūtītājs ir organizācija, kas pati nav sistēmas gala lietotājs, bet, piemēram, ir iecerējusi to tālāk pārdot vai ievietot savā interneta serverī, lai to lieto klienti. Šajā gadījumā pasūtītājs var izvirzīt galvenās prasības, bet tas var nezināt, vai, piemēram, saskarne, kas ir veidota dažādās valodās, ir vienlīdz korekta un vai piedāvātās iespējas ļauj veikt kādai konkrētai dzīves sfērai raksturīgos uzdevumus. Savukārt, citāda neatkarības situācija var būt tad, ja organizācija B, kas ir programmatūras izstrādātājs, pati pasūta organizācijai C pārbaudīt savu produkciju pirms tā tiek nodota A. Līdz ar to šeit jau var iezīmēties dažāda organizatoriskā pieeja

un saiknes līmenis ar izstrādātāju. Interesanta var būt situācija, kad pasūtītājs un izstrādātājs ir viena un tā pati organizācija, t.i. programma tiek veidota savām vajadzībām, bet tomēr arī šajos apstākļos ir nepieciešams vērtējums no malas.

No ģeogrāfiskā atrašanās viedokļa var būt gadījums, ka testēšana notiek pie organizācijas, kas ir veidojusi attiecīgo produktu (B), pie organizācijas, kas ir pasūtījusi šo produktu (A), vai arī neatkarīgi trešajā vietā - testēšanas laboratorijā pie testu veicēja (organizācija C).

Diezgan būtiski ir apzināties, vai testēšanas organizēšana tieši neatkarīgā formā ļaus iegūt labākus rezultātus, jo pie slikti plānota darba iznākums var būt pat pretējs. Kā vienu no galvenajiem ieguvumiem, uzticot pārbaudi specializētai organizācijai var minēt tās lielās pieredzes un kompetences izmantošanu, kas nāk kopā ar savām tradīcijām un metodiku. Tieši no ārienes ierosinātajai metodikai ir liela nozīme, bet ne mazāk svarīgs faktors ir, ka testēšanu veic arī paši izstrādātāji, jo visbiežāk trešās personas uzdevums ir pārbaudīt lietotāja līmeņa prasības, bet nevis tehniski iekšējas programmatūras lietas tādas, kā konkrētu procedūru komandu vai zarošanās nosacījumu pārklājums.

Latvijas IT uzņēmumiem, piemēram, A/S DATI ir bagāta pieredze projektu izpildē, kur pasūtītājs atrodas kādā no Rietumeiropas valstīm. Šajā gadījumā pastāv noteiktas prasības pret to, kā organizēt kvalitātes nodrošināšanu, jo ir jāspēj uzturēt netraucētu informācijas apmaiņu un kopēja mērķa, problēmu un risinājumu apzināšanu [MG-SQM].

Motivācija veidot vairākus atšķirīgus darbu centrus var veidoties, balstoties uz vēsturiskiem, ekonomiskiem vai uzņēmuma specifiskiem pamatiem. Viena situācija ir tāda, ka attālinātās grupas strādā laika zonu ziņā identiski vai ļoti līdzīgi, bet pavisam atšķirīgs gadījums ir, kad darba laiki nepārklājas, kas lielā mērā liedz dzīvās komunikācijas, bet vienlaikus piešķir arī savas priekšrocības. RITI prakse liecina, ka pilnībā ir veicami uzdevumi, kuri ir nodalāmi kā viens vesels veicamais darbs bez lielākām vai mazākām atkarībām. Vairāku organizāciju sadarbībā ir būtiski projekta ietvaros izveidot kopējas komandas principus, kā arī jāizmanto vienāda tipa metodika.

Programmatūras izstrādes procesa etapi un testēšana

Pēc procesa vai etapa

Ja mērķis ir saskatīt, kādos līdzīgos sīkākos procesos var sadalīt testēšanas procesu, tad iezīmējas tas, ka programmatūras dzīves ciklā vairākas reizes tiek uzsākta un nobeigta kāda testēšana, kur katrai ir mazliet atšķirīgi uzdevumi. Katrai koda funkcijai var tikt veidoti savi testi, katram funkciju apvienojumam var būt savi testi, visai sistēmai ir citi testi un pasūtītājs sagatavo vēl papildus testus, kas aplūko programmu tās reālās izmantošanas kontekstā. Ikvienai no šīm testēšanām ir iespējams uzturēt savu dokumentācijas komplektu, tai ir konkrēti ieejas un izejas dati, kā arī definējama procedūra, saskaņā ar ko notiek darbi. Katra no veicamajām testēšanām atbilst kādam no iepriekš notikušajiem izstrādes procesiem jeb etapiem.

Tas viss kopā uzskatāmi ir parādīts V-modelī [54]. Šis modelis ir veidojies kā tālāka attīstība vēsturiski populārajam ūdenskrituma modelim [121]. Tajā testēšana iznāk kā viena no aktivitātēm, kas seko implementēšanai. Lai arī tas ir stipri vienkāršots skatījums uz programmatūras izstrādi, šāda tipa modelis ilgu laiku bija pietiekami populārs, un tāds ir arī joprojām, jo kalpo kā pirmā uzskatāmā ilustrācija tam, kā top programmatūra. V-modelis testēšanas sabiedrībā tiek pozitīvi uztverts kā legalizācija tam, ka testēšana nav tikai viena neliela aktivitāte, bet gan vesela kopa ar dažāda veida programmatūras pārbaudēm.

V-modelis uzskatāmi parāda, ka atbilstoši tam, kāda līmeņa informācija tiek izmantota testu sagatavošanai, ir arī atbilstoša līmeņa testēšana, kur tipiski tiek izdalīti šādi līmeņi:

- vienībtestēšana – testējamā vienība ir mazākā atdalāmā izstrādātā programmas daļa, kas ir interešu sfērā;
- integrācijtestēšana – tiek pārbaudītas saskarnes starp vairākām vienībām, kur katra iepriekš ir pārbaudīta individuāli;
- sistēmtestēšana – visas programmas pārbaude apstākļos, kas tuvi ekspluatācijai;
- akcepttestēšana – pasūtītāja/lietotāja novērtējums par programmatūras atbilstību sākotnējām prasībām.

Nereti pirms akcepttestēšanas, vai arī kā tās paveids tiek pielietota arī atbilstības testēšana, kas būtībā ir organizatorisks pasākums - izstrādātājs demonstrē pasūtītājam programmatūras atbilstību definētajām prasībām.

Regresijas testēšana

Testēšanas process pēc savas uzdevuma nostādnes nozīmē to, ka problēmu konstatēšanas gadījumā ir jāveic izmaiņas programmā (t.i. jāatgriežas pie iepriekš veiktas aktivitātes), un pēc tam ir jāpārliecinās, ka izmaiņas tiešām ir notikušas. Ņemot vērā to, ka liela izmēra programmās var pastāvēt liels skaits dažādu savstarpējo atkarību starp funkcijām, klasēm un moduļiem, kā arī problēmas novēršana varētu būt veikta nekorekti, ir nepieciešams arī atkārtoti pārbaudīt, vai ir saglabājusies funkcionalitāte (vai kādas nefunkcionālās īpašības) tajos programmas apgabalos, kuri netika mainīti un kuri strādāja korekti. Šī atkārtotā jeb regresijas testēšana daudzos gadījumos ir izstrādes projektos definētais testēšanas darba režīms.

Ņemot vērā to, ka regresijas testēšana pēc būtības ir testu atkārtošana, tā visvieglāk pakļaujas dažāda veida automatizēšanai. Eksistē virkne rīku, piemēram, Mercury WinRunner, kas nodrošina šādu testu sagatavošanu un izpildi gan lietotāja saskarnes līmenī, gan programmas moduļu saskarnes līmenī.

Ātras novērtēšanas metodes

Mazāk formalizēta, bet praksē pielietota ir tā saucamā dūmu testēšana (vai dažviet, piemēram, uzņēmumā DATI saukta arī par diagonālo testēšanu). Tās būtība ir operatīvi pārliecināties par to, kas programmatūras sistēmā strādā un kas - nē. Šī nav formāla regresijas testēšana, bet gan testu komplekts, kas summāri izsauc pēc iespējas lielāku skaitu dažādu funkciju, lai testētāji iegūtu pirmo priekšstatu par sistēmas darbību. Iegūtos rezultātus parasti izmanto tālāko testēšanas darbību plānošanai. Ja dūmu testēšanas ietvaros tiek konstatēts, ka nekorekti strādā kāda būtiska funkcionalitāte, tad var tikt pieņemts lēmums nesākt apjomīgu sistēmas testēšanu, bet gan tiek precizēti problēmu iemesli un izstrādes grupai tiek dots uzdevums novērst konstatētās problēmas.

Veiklās testēšanas pieejas

Testēšanas attīstības viens no raksturotājiem ir tās formalizēto metožu attīstība. Tam pretstats ir pēdējos gados oficiāli deklarētās testēšanas pieejas, kas vairāk balstās uz cilvēku personīgo talantu un motivāciju nevis pieeju, kas ļautu formāli novērtēt programmatūras funkcionalitātes vai koda pārklājumu. Šādi principi izriet no pēdējā desmitgadē popularitāti ieguvušajām veiklajām (*agile*) izstrādes metodēm, no kurām populārākā ir ekstrēmā programmēšana [80]. Tādēļ gluži dabiski parādījās arī tādi jēdzieni, kā veiklā testēšana, izpētošā testēšana, ekstrēmā testēšana un ātrā testēšana. Vienojošais šīm veiklajām testēšanas pieejām ir mazs formalizācijas līmenis, programmas uzvedības izpēte atbilstoši novērojamo problēmu kontekstam, ātrs kvalitātes novērtējums un risku bāzēta testēšanas plānošana. Šādas metodes virknē gadījumu ir parādījušas labus rezultātus, tomēr nav atrodami pierādījumi, ka ar šādām metodēm ir veiksmīgi izstrādātas liela izmēra un komplicētas funkcionalitātes sistēmas. Veiklā pieeja pamatā der projektiem, kurā piedalās neliels cilvēku skaits, un ir ātri jāiegūst izmantojams produkts.

2.4. Testēšanas pārvaldība

Testēšanas sekmība ir atkarīga no tā, kādas metodes ir izvēlētas. Metodei ir jāatbilst programmatūras veidam, pieejamajiem resursiem, projekta struktūrai, izstrādes metodikai. Vispārējā gadījumā programmatūras izstrādes projektu pārvaldīšanā un arī testēšanā ir iespējams izmantot lielu daļu no vispārējām projektu pārvaldības metodēm, ja vien projekta vai procesa vadība ir apguvusi programminženierijas specifiku un nianses. Tajā pašā laikā arī testēšanas organizēšanā ir sastopamas vēl specifiskākas nianses, kas ir analizētas, piemēram, [39]. No organizatoriskā viedokļa projektā vienam no testētājiem ir jābūt atbildīgam par visa testēšanas procesa norisi un uzdevumu koordinēšanu ar pārējiem testētājiem. Kvalitatīva pieeja paredz, ka testēšanas process ir definēts dokumentā vai specializētā rīkā, un ir jābūt pierakstiem par testēšanas norisi – plānoto un faktiski novēroto. Autora pieredze liecina, ka nav iespējams precīzi saplānot testēšanu ar vienu piegājienu projekta sākuma etapā. Testēšana ir atkarīga no izstrādes gaitas un

sagatavotā produkta kvalitātes. Ja līdz paredzētajiem projekta rezultātu nodošanas datumiem nav iegūts pietiekams produkta kvalitātes līmenis (t.i., testēšana uzrāda problēmu skaitu, kas pārsniedz projektā noteikto pieļaujamo apjomu), tad testēšanai ir jāpielāgojas projekta turpinājumam jaunos apstākļos.

Vispārīgā gadījumā testēšanai neeksistē metodes, kā prognozēt nepieciešamo darbu apjomu tam, cik reizes un cik ilgstoši būs nepieciešams testēt programmu. Ja ir pieejama izstrādes dokumentācija un kods, ir iespējams novērtēt, cik testu saskaņā ar konkrētu izvēlēto pieeju būs jāsagatavo. Katram testam ir iespējams prognozēt tā izpildes laiku, bet nav iespējams prognozēt, cik daudz problēmu parādīsies testēšanas laikā, kā tās ietekmēs tālāko testēšanas gaitu, cik ilgi būs jāgaida programmas labojumi un cik reizes būs nepieciešams atkārtot visu testēšanas ciklu. Tradicionāls testu apjoma un to izpildes apjoma nosacījums izriet no izvēlētās testēšanas metodes. Piemēram, standarts BS-7925 definē, ka komponentu testēšanā ir jāpieturas pie cikliska procesa, kas sākas ar komponentu testu plānošanu un noslēdzas ar komponentu testu izpildes pārbaudi. No vienas puses izvēlētā metode ļauj kvantitatīvi noteikt, cik testu būs, bet no otras puses tā paredz modificēt testēšanas plānu atbilstoši iegūtajiem rezultātiem.

Praksē visbiežāk sastopamais ierobežojums veicamajiem testēšanas apjomiem ir projekta termiņš un resursi – cilvēki, finanses, testēšanas vides pieejamība. Ja projekta norise skaidri iezīmē regulāras jaunu versiju piegādes un nepieciešamību tās pārtestēt, tad testēšanas pieeja tiek veidota tā, lai katrā ciklā būtu iespējams pārliecināties par izmaiņām produkta kvalitātē (iepriekš konstatēto problēmu novēršanas sekmība, jaunu problēmu parādīšanās). Lai to veiktu, tiek izmantotas testēšanas automatizācijas metodes, kā arī tiek samazināta veicamo testu kopa, izpildot pēc noteiktiem kritērijiem izvēlētos būtiskākos testus.

2.5. Programmatūras testēšanas prakse Latvijā

Laba ilustrācija testēšanas kā nozares attīstībai pasaulē ir arī aktivitātes šajā jomā Latvijā, īpaši pēdējā desmitgadē. Latvijā pirmās sistemātiskās aktivitātes testēšanas jomā bija saistītas ar pētījumiem automātisko testpiemēru konstruēšanā, izstrādājot gan teorētisku, gan praktisku pieeju. Kad 20.gs 90.gadu sākumā Latvijā sāka attīstīties IT nozare, parādījās arī nepieciešamība pēc izstrādājamo programmatūras sistēmu testēšanas darbu izpildes. Lielākais nozares uzņēmums savā paspārnē izveidoja Testēšanas laboratoriju, kas specializējās tieši testēšanas darbu veikšanā.

Pirmajos projektos kvalitāte tika nodrošināta, balstoties uz iepriekšējo pieredzi un intuīciju. Ja sākotnēji gandrīz ikviena programma šķita pieņemama, ja vien tā pamatā atbilda galvenajām nostādnēm, tad situācijā, kad programmatūras sistēmas daudzos gadījumos ir pamats uzņēmuma funkciju nodrošināšanai, strauji pieauga pasūtītāju prasības pret kvalitāti, kas arī veicināja programmatūras kompāniju aktivitātes kvalitātes uzlabošanas virzienā. Šajā nolūkā IT kompānijām bija jāveic

atbildes gājiens – kopā ar savas darbības kvalitātes principu izstrādi tām bija jāvelta uzmanība pasūtītāja izglītošanai. Kvalitāte nevar būt vienpusējs pasākums. Arī pasūtītājs nevar rīkoties neorganizēti un ģenerēt pretenzijas, jo tas padarīs projekta norisi komplicētāku, ietekmēs izpildes termiņus un budžetu. Lai arī pēdējos gados, pateicoties pasūtītāju izglītošanai, neizpratne par testēšanas nepieciešamību ir būtiski mazinājusies, tomēr joprojām eksistē problēmas ar to, ka pasūtītājs neakceptē projekta plānos paredzētos testēšanas darbus kā obligātu projekta darba sastāvdaļu. Autora apzinātajos projektos pasūtītāju tipiskā nostādne ir tāda, ka programmatūra ir jāraksta uzreiz pareizi, un līdz ar to testēšanai nevajadzētu būt nepieciešamai vispār, vai arī tas varētu būt neliela apjoma darbs. Būtiskākais iemesls šādai pretestībai ir finansiālās sekas – pasūtītājam rodas iespaids, ka tiek maksāts par lieku darbu, jo programma jau ir pieejama lietošanai. Ir fiksēti gadījumi, kad projekti ir bijuši spiesti patērēt vairākas reizes vairāk laika un resursu testēšanai nekā tas bija oficiāli ieplānots ar pasūtītāju noslēgtā līguma ietvaros. Šāda veida piekāpšanās pasūtītāja viedoklim ir iespējama tikai gadījumā, ja IT uzņēmumam šis projekts neveido lielāko daļu plānoto ienākumu.

Līdzīgi, kā situācijā ar attieksmi pret testēšanu, var veidoties situācija ar citām aktivitātēm, piemēram, kopīgām projekta norises apskatēm (vai Izmaiņu vadības padomi). Ja uzņēmumā ir kvalitātes sistēma, kuras ievērošana ietilpst uzņēmuma darbības principos, tad ir minimālās kvalitātes prasības, no kurām uzņēmums nevar atteikties. Konkrēti testēšanas gadījumā no autora pieredzes ir bijuši vairāki projekti, kur uzņēmums uz sava rēķina piesaista papildus testētājus, lai būtu iespējams veikt adekvātu testēšanu.

Ja aplūkojam, kā attīstījās vienoti kvalitātes principi Latvijā, tad izšķirošs aizsākums bija 1993.-1995.gadā DATI kompānijas uzsāktā starptautisko programminženierijas standartu analīze. Mērķis bija atrast labākos un plašāk akceptētos nozares standartus, apkopot pieredzi no eksistējošiem projektiem, lai izveidotu savus standartus, projektu darbu atbalstošas vadlīnijas un sagataves. Tā rezultātā uz IEEE starptautisko standartu bāzes tika sagatavota virkne uzņēmuma standartu. 1996.gadā organizācijas standarti tika izmantoti par pamatu atbilstošo Latvijas Valsts standartu izstrādei. Tie aptvēra šādas programminženierijas sfēras:

- Kvalitātes nodrošināšana, konfigurācijas pārvaldība un audits [95], [100];
- Testēšana, verifikācija un validācija [96], [97], [99];
- Programmatūras dokumentācija [96], [101];
- Programmatūras prasību specifikācija un darbības koncepcijas apraksts [94], [101];
- Vadlīnijas programmatūras projektējuma aprakstam [98];
- Projekta pārvaldības plāni [93], [95], [91];
- Lietotāja dokumentācija [92].

Paralēli IT standartu adaptēšanai notiek arī darbs pie IT terminoloģijas latviešu valodā. Jāpiebilst, ka, neskatoties uz to, ka līdz 2004.gadam bija latviskoti vairāk kā 5000 IT terminu, joprojām ir vairāki testēšanas termini, kas oficiāli nav apstiprināti, bet tiek izmantoti to latviskie varianti. Piemēri – *regresijas testēšana, sistēmtestēšana, integrācijtestēšana, problēmziņojums, testžurnāls.*

Vadošie IT uzņēmumi pieturas pie starptautiskiem standartiem tādiem, kā IEEE J-STD-016-1995, kā arī savus uzņēmuma procesus organizē atbilstoši starptautiskiem standartiem, piemēram, organizācijas darbu saskaņā ar ISO 9001:2000 un programmatūras projektus atbilstoši ISO/IEC 12207. Standarts J-STD-016-1995 tika izvēlēts tādēļ, ka tas apraksta visas programmatūras izstrādes aktivitātes, satur galveno dokumentu sagataves un vadlīnijas to veidošanai, vadlīnijas pielāgošanai konkrētos projektos, definē labu kvalitātes nodrošināšanas mehānismu.

Kas attiecas uz testēšanas organizēšanu IT kompānijās, tad tās no savas pieredzes ir izdarījušas vairākus secinājumus:

1. Testēšana ir jāorganizē gan ar neatkarīgās testēšanas institūcijas (laboratorijas) palīdzību, gan ir jāorganizē testēšanas grupas projektu ietvaros.
2. Testētājiem ir jābūt ar plašām zināšanām un pieredzi. Prakse liecina, ka daudzos gadījums kvalifikācijai jābūt ne mazākai kā projektā strādājošajam sistēmanalītiķim vai projektētājam, kā arī papildus jāpārzina testēšanas specifiskie jautājumi. Testētājiem ir definējami kvalifikācijas līmeņi. Organizācijas iekšienē testētāja zināšanu apgabalu ir vēlams definēt atbilstoši SWEBOK [72] prasībām.
3. Vienīgais veids, kā nodrošināt kvalitatīvu testēšanu minimālā laikā ar iespējami maziem resursiem, ir pielietot labi organizētu procesu.
4. Daudzi programmatūras pasūtītāji ir slikti informēti par testēšanu, tādēļ ir nepieciešams veltīt uzmanību arī pasūtītāju izglītošanai.
5. Lai arī ir iespējams automatizēt daudzas testēšanas aktivitātes, nav izdevies atrast apliecinājumu tam, ka mazinātos manuālās testēšanas nozīme. Ir vairākas sfēras tādas, kā lietotāja saskarnes, lietotāja rokasgrāmatas, lokalizācijas u.c. testēšana, kur nav iespējams automatizēt testu sagatavošanu.
6. Ņemot vērā pieaugošo apziņu par testēšanas nozīmi, ir sagaidāms, ka pieaugs IT kompāniju pieprasījums pēc testētājiem.

No organizatoriskā viedokļa testēšanas laboratorijas vai nodaļas jēdziens ir tikai lielākajās IT kompānijās. Jāņem vērā, ka šādas struktūrvienības uzturēšana saistās ar atsevišķu resursu izdalīšanu un risku, ka tā var nebūt pastāvīgi nodrošināta ar darbiem. Ja rodas dīkstāve, struktūrvienības uzturēšana kļūst ekonomiski neizdevīga. Tipiska pieeja darbu veikšanai ir tāda, ka programmatūras izstrādes projekta sākuma etapos tajā no testēšanas laboratorijas tiek iesaistīta persona testēšanas aktivitāšu plānošanai. Lai arī iesaistītā persona ir it kā no malas, testēšanas plānošanas darbos aktīvi ir jāiesaistās arī citiem projekta darbiniekiem – īpaši projekta pārvaldniekam un

sistēmanalītiķiem. Testēšanas laboratorijas eksistēšanas galvenā priekšrocība ir vienotas metodikas izmantošana, neatkarīgas testēšanas pieejas garantēšana, kā arī noteiktas profesionāļu kopas pieejamība.

Mazliet citādāka ir situācija gadījumos, ja projektā netiek iesaistīti profesionāli testētāji. Tad šo lomu veic sistēmanalītiķi, vai arī programmētāji, pārbaudot kolēģa kodu. Šajā gadījumā testētājs ir loma, ko uz noteiktu laiku pieņem kāds no projekta dalībniekiem. Priekšrocības šādam modelim ir tādas, ka testēšanu ir iespējams veikt visos līmeņos, kā arī testējošās personas ļoti labi pārzina testējamo objektu. Problēmas ar šādu modeli ir nelielo projektu gadījumā, jo šādos gadījumos katrs dalībnieks zina samērā lielu daļu no visas projekta informācijas, un vairs nespēj objektīvi paskatīties uz notiekošo – tiek pazaudēts neatkarības princips.

Projekta iekšējās testēšanas gadījumā dokumentācija parasti ir atvieglotākā formā nekā tas ir neatkarīgās testēšanas gadījumā. Tā ciešāk ir saistīta ar projekta izstrādes procesu. Neatkarīgā testēšana sākas projekta vēlākos etapos – parasti tad, kad ir darbināma sistēma, un lielāks uzsvars tiek likts uz detalizētu dokumentāciju, jo neretas ir situācijas, kad neatkarīgās testēšanas rezultāts kalpo par pamatu lēmuma pieņemšanai par sistēmas ieviešanu vai līgumattiecībām starp pasūtītāju un izstrādātāju.

Industriālu projektu gadījumā ir novērots, ka ir nozīme pēc iespējas ātrākai testētāju iesaistīšanai projektā – arī tad, ja pat vēl neviena koda rindiņa nav uzrakstīta. Veicamās aktivitātes ir norādītas 2.tabulā.

2.tabula. Testētāja darbi projekta darbos.

Projekta darba veids	**Testētāja uzdevumi**
Sistēmanalīze un prasību specificēšana	Pārliecināties, vai prasību specifikācija un citi dokumenti ir piemēroti testēšanai. Plānot testēšanu.
Projektēšana	Detalizēti plānot testus. Gatavot testu aprakstus.
Kodēšana	Precizēt testēšanas plānus un aprakstus. Izpildīt testus. Gatavot automātiskos testus.
Testēšana	Izpildīt vispusīgus testus, ieskaitot sistēmas, instalācijas un dokumentācijas testus. Veikt regresa testēšanu.
Sagatavošanās nodošanai	Pārbaudīt labojumus. Izpildīt regresijas testus.

Ja aplūko testēšanā iesaistīto personālu, tad var novērot tendenci, ka augstāka līmeņa testēšanas gadījumos to biežāk veic neatkarīgs un profesionāls testētājs (3.tabula).

3.tabula. Testēšanas līmeņi un testu veidi.

Testēšanas līmenis	**Strukturālie testi**	**Funkcionālie testi**	**Testēšanu veic**

Vienībtestēšana	+		Izstrādātāji
Integrācijstestēšana	+	+	Izstrādātāji
Sistēmtestēšana		+	Izstrādātāji, neatkarīgie testētāji
Kvalifikācijas testēšana		+	Izstrādātāji, neatkarīgie testētāji
Akcepttestēšana		+	Pasūtītājs, neatkarīgie testētāji

Īpaša nozīme ir neatkarīgā testētāja komunikācijai ar projekta dalībniekiem. Tīrā formā testēšanai ir nepieciešamas divas lietas – programma, ko testēt, un informācija, kas apraksta tās darbību. Tomēr samērā plaši izplatīta ir situācija, kad dokumentācija ir nepilnīga, nav aktualizēta, un tā ir nepiemērota labu testu sagatavošanai. Tādēļ testētājam ir jāspēj izzināt no projekta pārvaldnieka, sistēmas arhitekta, sistēmanalītiķa un programmētājiem visu aktuālāko un precizēto informāciju.

Kopš 1993. gada Latvijas Universitātē tiek pasniegts izvēles kurss Testēšana. Novērojumi liecina, ka universitātēs tā nav plaši izplatīta prakse, ka datorzinātņu studiju programma saturētu arī atsevišķu kursu veltītu testēšanai.

2004.gadā veiktā informācijas apmaiņa starp IT uzņēmumiem un augstskolām par iespējām apgūt programmatūras testēšanu parādīja, ka eksistē četras mācību programmas: 1) izvēles kurss Latvijas Universitātes bakalaura programmā, 2) izvēles kurss Latvijas Universitātes maģistrantūras programmā, 3) mācību programma uzņēmumā Exigen Latvia un 4) mācību kurss Rīgas Informācijas tehnoloģijas institūtā, kuru ir sagatavojis un pasniedz šī darba autors.

Optimāls pamats testēšanas profesionāļa zināšanu kopas definīcijai ir IEEE organizētā projekta ietvaros sagatavotais SWEBOK standarts. Lai arī tas tiešā veidā netiek izmantots par pamatu kādai no minētajām programmām, tas kalpo kā labs satura rādītājs iegūtu zināšanu strukturēšanai un apgūstamo sfēru identificēšanai. Universitātes kursi vairāk orientējas uz teorētisko pamatu iestādīšanu, bet uzņēmumos organizētās mācības klausītājus vairāk sagatavo uz konkrētajā uzņēmumā nepieciešamajām zināšanām un iemaņām. Konkrēti, SWEBOK aplūko testēšanu šādā skatījumā: testēšanas koncepcija un definīcijas, testēšanas līmeņi, testēšanas metodes, testu automatizācija un ar testēšanu saistītie mērījumi.

Balstoties uz praktiskajā testēšanā gūto pieredzi un nozares standartu prasībām, autors piedalījās testētāju kvalifikācijas līmeņu projekta izstrādē IT uzņēmuma vajadzībām. Tā rezultāts ir priekšlikums uzturēt četru līmeņu klasifikāciju.

4.tabula. Testētāju kvalifikācijas līmeņi.

Līmenis	**Uzdevumi**	**Nepieciešamās iemaņas**
1	Iepriekš specificētu testu izpilde un citu uzdevumu veikšana, kas neprasa iepriekšēju sagatavošanos.	Attīstītas datorprogrammu lietošanas iemaņas, izpratne par programmatūras uzbūvi.

2	*1.līmeņa uzdevumi* + Testu specifikāciju sagatavošana saskaņā ar sistēmas dokumentāciju	*1.līmeņa iemaņas* + Spēja analizēt programmas dokumentāciju un rakstīt testēšanas dokumentāciju.
3	*2.līmeņa uzdevumi* + Testu plānošana un testēšanas grupas pārvaldīšana	*2.līmeņa iemaņas* + Spēja plānot nepieciešamos testus, iemaņas un personālu
4	*3.līmeņa uzdevumi* + Citu testētāju apmācība un testēšanas metožu izstrāde	*3.līmeņa iemaņas* + Mācīšanas iemaņas un eksperta līmeņa zināšanas testēšanā

Papildus 4.tabulā minētajiem līmeņiem nosacīti var definēt arī „0" līmeni, kam pieder ar IT jomu nesaistītas personas bez iepriekšējām zināšanām testēšanas jomā. Šādi testētāji varētu būt akcepttestēšanas procesā iesaistīti potenciālie lietotāji, vai arī lietojamības testēšanā nodarbinātās pieaicinātās personas.

Tajā pašā laikā, 2002.gadā, Latvijas IT uzņēmumi strādāja pie amatu klasifikatoriem, kas pamatā bija nepieciešami atalgojuma sistēmas nodefinēšanai. Piemēram, autors daļēji tika iesaistīts arī uzņēmuma DATI testētāju amatu prasību definēšanā. Par pamatu tika izvirzīts dalījums četros amatu līmeņos: Jaunākais testēšanas speciālists, Testēšanas speciālists, Vecākais testēšanas speciālists un Vadošais testēšanas speciālists. Analoģisks četru līmeņu dalījums ir arī vairākās citās amatu kategorijās tādās, kā programmēšana, tīklu administrēšana un sistēmas analīze. Arī citos Latvijas uzņēmumos eksistē definēti amatu līmeņi. Salīdzinošos nolūkos iegūta informācija liecina, ka ir sastopams dalījums 3 kvalifikācijas līmeņos.

Skatoties uz profesijas jautājumu no izglītības un iemaņu prasību viedokļa, ir jautājums, kā sagatavot šādus darbiniekus. Izglītības iestādēm ir jāsaprot, kādas profesijā veicamās tipiskās darbības, pienākumi, uzdevumi, nepieciešamās prasmes un zināšanas [19]. Testētāja profesijas kontekstā 2003.gada 30.aprīlī ar Latvijas Izglītības un zinātnes ministrijas rīkojumu Nr. 187 tika apstiprināts standarts profesijai Datorsistēmu testētājs [78], kura izstrādē ir piedalījies arī šī darba autors. Saskaņā ar Latvijā 30.06.1999 izsludināto Profesionālo izglītības likumu [128] pastāv pieci profesionālās kvalifikācijas līmeņi:

1) pirmais kvalifikācijas līmenis - teorētiskā un praktiskā sagatavotība, kas dod iespēju veikt vienkāršus uzdevumus noteiktā praktiskās darbības sfērā;

2) otrais kvalifikācijas līmenis - teorētiskā un praktiskā sagatavotība, kas dod iespēju patstāvīgi veikt kvalificētu izpildītāja darbu;

3) trešais kvalifikācijas līmenis - paaugstināta teorētiskā sagatavotība un profesionālā meistarība, kas dod iespēju veikt noteiktus izpildītāja pienākumus, kuros ietilpst arī izpildāmā darba plānošana un organizēšana;

4) ceturtais kvalifikācijas līmenis - teorētiskā un praktiskā sagatavotība, kas dod iespēju veikt sarežģītu izpildītāja darbu, kā arī organizēt un vadīt citu speciālistu darbu;

5) piektais kvalifikācijas līmenis - noteiktas nozares speciālista augstākā kvalifikācija, kas dod iespēju plānot un veikt arī zinātniskās pētniecības darbu attiecīgajā nozarē.

Šobrīd vienīgais sagatavotais un apstiprinātais Datorsistēmu testētāja standarts atbilst 4.kvalifikācijas līmenim. Tuvākajā nākotnē būtu lietderīgi definēt profesiju standartus vēl arī 3. un 5.līmenim. Tā kā 1. un 2.līmenis izvirza relatīvi zemas profesionālās prasības, un pieprasījums pēc atbilstošiem darbiniekiem darba tirgū varētu būt salīdzinoši zems, ir iepriekš jāizanalizē, cik lietderīga būtu šādu standartu izstrāde.

No iepriekš minētajām pieejām un to tapšanas vēstures var secināt, ka laika periodā ap 2002.gadu izglītības jomā un darba devēju aprindās bija nobriedusi nepieciešamība pēc precīzām profesiju un amatu definīcijām. Tikai kopš 1998.gada Latvijas Republikas Profesiju klasifikatorā ir profesija „Informācijas sistēmu testētājs" (no 2004.gada papildus ir nodefinēta profesija „Datorsistēmu testētājs"). Tā ir vēlme izveidot sistēmu, kas ļauj visām iesaistītajām pusēm – darba devējam, darba ņēmējam un izglītības iestādei – saprast savstarpēji izvirzītās prasības, izvēlēties profesiju, plānot karjeru un sagatavot mācību programmu. Iepriekš aprakstītās amatu un profesijas kvalifikācijas līmeņu sistēmas nav pretrunīgas, bet tomēr ievieš atšķirīgus dalījumus līmeņos, kā arī satur atšķirīgas konkrētās profesionālās prasības. Jāņem vērā, ka darba devējiem vienmēr būs savas specifiskās prasības, kas ir atkarīgas no konkrētās darba jomas –uzņēmums var pieturēties pie kāda starptautiskas uzņēmumu grupas standarta, vai arī tā veicamais testēšanas darbs ir šauri specializēts (piemēram, tikai drošības testi).

Vienotā nostādne testētāja profesijas gadījumā ir tā, ka testētājam ir labi jāpārzina ne tikai programmatūras izstrādes tehniskā puse un testēšanas specifika, bet arī testējamās programmatūras pielietošanas sfēra. Piemēram, uzņēmuma finanšu pārvaldības sistēmas testēšana nebūs efektīva, ja testētājs neorientēsies grāmatvedības jautājumos.

2.6. Manuālā testēšana / testēšanas darba organizēšana

Lai arī saprātīgs solis darba efektivitātes paaugstināšanai ir pēc iespējas visas testēšanas aktivitātes automatizēt, britu kompānijas Grove Consultants [60] pētījums liecina, ka tomēr liela daļa industriālo projektu nav guvuši iecerēto automatizācijas atbalstu no testēšanas rīkiem. Galvenās problēmas ir:

- Katras konkrētās programmatūras izstrādes un darbināšanas vides specifika;
- Rīki neplāno un neprojektē testus cilvēku vietā (izņemot konkrētus strukturālo testu veidus);
- Regresijas testēšanā sagatavotie testi ir pastāvīgi jāmaina atbilstoši programmas izmaiņām.

Automatizācija, kas visbiežāk izpaužas tieši kā regresijas testi, ir parasti pielietojama gadījumos, kad testējamā programmai ir nostabilizējušies no funkcionāli apgabali. Praksē tas nozīmē, ka programmai ir jābūt faktiski gatavai, un no kāda brīža turpmāk tajā tiek veiktas tikai atsevišķas izmaiņas.

Tieši iepriekšminētā iemesla dēļ stipri aktuāla ir manuālā testēšana, t.i. testēšana, kur būtiskos testa soļus un novērtējumu veic testētājs. Autors ir veicis novērojumus par izmantotajām testēšanas metodēm Latvijas IT uzņēmumos. Lai arī pētījums nav veikts sistemātiski un tā veikšanā ir jāsaskaras ar konfidenciālu uzņēmumu projektu informāciju, vairāku gadu garumā līdzdalība dažādos testēšanas forumos un komunikācija ar nozares kolēģiem liecina, ka automatizācija netiek plaši praktizēta. Alternatīvas situācijas ir tādas, ka lieli resursi tiek ieguldīti manuālajā testēšanā, vai arī testēšana tiek veikta nepietiekamā apjomā. Bieži vien pat pie labākās gribas nav iespējams veikt automatizēšanu projekta specifisko ekonomisko apsvērumu dēļ, tādēļ joprojām ir aktuālas metodes, kā tieši veidot manuālās testēšanas procesu. Vairāku gadu garumā autors ir praktiski veicis testēšanu bez automatizācijas, vadījis šādas testēšanas grupas un guvis pieredzi par to, kādas ir iespējamās pieejas manuālās testēšanas organizēšanā. Tas ir atspoguļots rakstā [3].

Manuāla rakstura testēšana galvenokārt ir sastopama neatkarīgās testēšanas gadījumā, jo tā bieži saistās ar kādiem īpašiem uzdevumiem, kas tiek veikti vienu reizi, un prasa profesionālu testētāju vērtējumu. Šāda testēšana ir aktuāla arī tad, ja ir nepieciešams veikt programmatūras analīzi attiecībā pret tās dokumentāciju un apkārtnē esošo sistēmu (kas varētu arī nebūt aprakstīta programmatūru pavadošajos dokumentos), nepieciešams novērtēt sistēmas lietojamību vai atbilstību kādiem ārējiem standartiem. Manuāla testēšana nenozīmē zemāku kvalitāti. Tā vairāk ir orientēta uz novatorismu, jaunu īpašību analīzi un kompetentu kvalitātes novērtējumu.

Ja novērtē to, kāds ir neatkarīgās (un arī manuālās) testēšanas īpatsvars tradicionālajos testēšanas līmeņos, tad var secināt, ka vienībtestēšanā tas ir vismazākais, jo parasti testējamā vienība tiek izsaukta ar citas programmas palīdzību, un ir skaidri definēta testu kopa, ar kuru vienība pēc katras izmaiņas tiek pārbaudīta – iesaistīt šādā pārbaudē citu cilvēku nevarētu būt lietderīgi. Integrācijas līmenī situācija ir analoģiska, vienīgi izņēmuma kārtā atsevišķas saskarnes tiek pārbaudītas arī ar manuālu izsaukšanu. Sistēmas līmenī automatizācija parasti ir regresijas testēšanas formā, bet akcepttestēšana parasti ir īpaši organizēts pasākums, un gadījumā, ja sistēma ir pamatā balstīta uz cilvēku izraisītu notikumu apstrādi, tad automatizācijas iespējas un lietderība ir neliela.

Automatizācija tiek izmantota, ja ir kaut kas jāatkārto, vienveidīgi jāizpilda kādas operācijas, jāģenerē un jāfiksē notikumi, jāimitē saskarnes ar citām sistēmām. Tā nevar aizvietot cilvēku testēšanas mērķu definēšanā, plānošanā un testu aprakstīšanā. Būtiskākais ir tas, ka nereti nepieciešams novērtēt testēšanas starprezultātus, lai pieņemtu lēmumu par tālāko testu izpildes gaitu. Tas ir īpaši tipiski, piemēram, veiklajai un izpētošajai testēšanām. Būtiska testēšanas sastāvdaļa ir

problēmu situāciju izpēte, problēmu cēloņu identificēšana. Automatizētā pieeja visbiežāk spēj dot bināras formas atbildi "jā"/"nē", bet ar lietotāju prasībām saistītos jautājumos ar to bieži vien nav pietiekami. Ir nepieciešams arī saprast problēmas cēloņus.

Manuālajai testēšanai labvēlīgu vidi rada tādi projekti, kas balstās uz unikālu risinājumu. Automatizācija ir vāji iespējama nepilnīgi definētu un mainīgu prasību apstākļos, kā arī tad, ja testējamā programmatūra ir izteikti orientēta uz lietotāja prasībām. Negatīvu ietekmi uz automatizāciju atstāj arī tādi projekta organizācijas apstākļi, kā sasteigti darbu izpildes termiņi un nesistemātisks programmatūras izstrādes process.

Pieredze darbā ar reāliem projektiem RITI ietvaros liecina, ka tīri organizatoriski testēšanas grupā var nokļūt cilvēki ar atšķirīgām pieredzēm un specializācijām. Ne vienmēr testētājs ir ar datorzinātņu izglītību, un bieži vien tas pat nav nepieciešams, jo funkcionālo testu sagatavošanā būtiski ir pārzināt atbilstošo jomu, kur programma tiks izmantota, bet testēšanā izmantojamo programmatūru var apgūt relatīvi īsā laikā. Cits jautājums ir par šāda cilvēka spēju izprast problēmas cēloņus, diskutēt par tiem ar izstrādātāju un izpētīt citas vietas programmā, kas varētu saturēt problēmas ar līdzīgu izcelsmi.

Izstrādātāji no testētāju komandas sagaida ātru integrēšanos gan programmas darbības jomas jautājumos, gan pietiekamu kompetenci, lai atbalstītu izstrādātāju kvalitātes procesus (piemēram, konfigurācijas pārvaldību vai apskates), sniegtu kompetentu slēdzienu par programmā esošajām problēmām.

Testētāju zināšanu apgabalu dažādu projektu gadījumā var definēt atšķirīgi. IT uzņēmumu pārstāvji pārsvarā uzsver to, ka viņiem ir nepieciešami cilvēki ar praktiskām testēšanas iemaņām. Lai arī ir saprotams, ka katrā konkrētā projektā netiks izmantotas visas mācībās apgūtās metodes, tomēr jāsaprot, ka mācības nedrīkst aprobežoties tikai ar to, kas būs nepieciešams tuvākajos mēnešos. Pirmkārt, nav zināms, vai tiešām tikai ar iemācītajām metodēm pietiks tiešo darba pienākumu veikšanai. Otrkārt, projekta norises laikā var nebūt iespējas testētājus apmācīt papildus. Mācībām ir jāsniedz priekšstats par līdzšinējām teorētiskajām un praktiskajām atziņām programmatūras testēšanas nozarē, kā arī tās pozīciju citu kvalitātes pasākumu kontekstā. Profesijas standarts pārskata tās zināšanas, kas būtu nepieciešamas datorsistēmu testētājam. Papildus tam, ka augstākajā (pielietošanas) līmenī ir jāpārzina visi uz testēšanu tieši attiecināmie jautājumi, vidējā līmenī (izpratnes) tiek prasīts pārzināt programmatūras dzīves ciklu, konfigurācijas pārvaldību, sistēmanalīzi, programmēšanas valodas, operētājsistēmas, statistiku, prezentācijas māku un ergonomiku. Ir jābūt priekšstatam arī par programminženieriju kopumā, datoru arhitektūru, klienta servera tehnoloģijām, tīklu tehnoloģijām, statistiku, projektu vadību un darba tiesībām.

Lai sagatavotu jaunos testētājus reālam testēšanas darbam, autors RITI ietvaros ir vadījis vismaz 10 testēšanas mācību kursus „Programmatūras testēšanas pamati".

Ņemot vērā to, ka testēšana tiek mācīta tikai vienā Latvijas augstskolā, kā arī minētais priekšmets ir izvēles, autors bieži saskārās ar situāciju, ka jaunie testētāji ir ar datorzinātņu izglītību, bet testēšanu universitātē nebija apguvuši. Tādēļ apmācību kurss satur gan testēšanas teorijas pamatus, gan praktiskos norādījumus testu plānošanai, aprakstīšanai un izpildei uzņēmuma projektos. Vadītie kursi ir bijuši robežās no 10 līdz 26 akadēmiskajām stundām, bet tipveida programma ir orientēta uz apjomu 16 vai 20 stundas (teorētiskā daļa vienāda, bet ir atšķirīgs praktisko darbu apjoms). Kursa mērķis ir sagatavot testētāju, kas atbilstu vismaz 2.kvalifikācijas līmenim saskaņā ar [3]. Mācību programma aptver šādas tēmas:

- Programmatūras testēšanas pamatjautājumi - definīcijas, testēšanas mērķi, īss ieskats vēsturē, testēšana pie izstrādātāja un neatkarīgā testēšana, testēšanas veidi un līmeņi;
- Testēšanas metodes - melnās kastes metodes, baltās kastes metodes;
- Testēšanas process - aktivitātes un dalībnieki, testēšanas organizēšana, dokumentācijas veidi, standartu izmantošana, testu veikšana, trasējamība;
- Testu plānošana - plānošanas mērķi, iespējamās formas, standartu prasības;
- Testu aprakstīšana - testu specificēšanas veidi, standartu prasības;
- Testu izpildīšana - darba organizēšana, dokumentācija, problēmu reģistrēšana un apstrāde;
- Testēšanas rezultātu apkopošana un analīze - iegūto rezultātu pielietojums, standartu prasības;
- Specializētie jautājumi - tīmekļa testēšana, automatizācija.

Iepriekšminētais kurss ir veidots no vairākām komponentēm – lekcijām, praktiskajiem darbiem, mājas darbiem un noslēguma pārbaudes darba. Viss darbs ir organizēts 3 vai 4 klātienes tikšanās reizēs. Lekcijas ir 11 stundu apjomā. Praktiskie darbi ir 2-6 stundas, 1 stunda ir noslēguma pārbaudes darbiem, un 2 stundas ir mājas darbu analīzei. Pie mācību kursa apjoma netiek rēķināts laiks mājas darbu izpildei – tas ir katram individuāls. Mājas darbi ir veidoti tā, lai 2 prasītu strukturālās testēšanas metožu pielietojumu, 2 – funkcionālo metožu lietošanu, bet viens ir orientēts uz testu plānošanu. Lai attīstītu praktiskas testēšanas iemaņas, mācību kursam tika sagatavota īpaša datorprogramma, kas satur speciāli ieliktas kļūdas. Tā ir neliela datubāzes sistēma, kas varētu būt izmantojama nelielā transporta uzņēmumā. Tā satur ekrāna formas klasifikatoru datu uzturēšanai, kā arī ekrānus finansiālu aprēķinu veikšanai. Testēšanas ietvaros var atklāt gan saskarnes, gan funkcionālas kļūdas. Programmas teksts nav pieejams, bet sistemātiska melnās kastes metožu pielietošana ļauj identificēt lielāko daļu problēmu.

Kā bija minēts, iepriekš, autora pieredze testēšanas laboratorijā liecina, ka vēsturiski tajā darbu ir sākuši cilvēki ar atšķirīgu iepriekšējo pieredzi. Ne visi ir nesen

beiguši augstskolu un ieguvuši kādu no datorzinātņu grādiem. Ir tādi, kuriem studijas ir bijušas citā nozarē, vai arī iepriekšējos gadus darbs nav bijis saistīts ar IT jomu. Lai arī IT zināšanām ir nozīme, prakse liecina, ka testēšanas laboratorijai vērtīgs ieguvums ir cilvēka zināšanu daudzpusība. Piemēram, būtiskas ir dažādu dzīvo valodu zināšanas, finanšu un ekonomikas pamatu pārzināšana, orientēšanās telekomunikāciju sfērā utt. Lai arī daudzi testēšanu joprojām uzskata par mākslu, testētājam pēc dabas nav jābūt brīvmāksliniekam, bet gan jāspēj pakļauties uzstādītajiem mērķiem, sastādītajiem plāniem, kā arī rezultāti jāsagatavo atbilstoši projekta prasībām. Tajā pašā laikā testētāja plašās zināšanas un pieredze palīdz sagatavot un izpildīt labus testus.

Tipiska kļūda testētāju komandas organizēšanā ir tajā iesaistīt cilvēkus, kas ir bijuši slikti programmētāji vai sistēmanalītiķi. Testētājus nedrīkst uztvert kā mazāk kvalificētus darbiniekus. No tā cietīs programmatūras kvalitāte – produkts nebūs atbilstoši notestēts. Analoģiski ir gadījumā, kad kādu programmētāju par neatbilstošu darbu uz laiku nosūta uz testēšanas darbiem kā sodu. Pati par sevi darbu maiņa uz laiku nebūtu nekas slikts, bet testēšanas darbu nostādīšana sodīšanas instrumenta lomā diskreditē darba kvalitāti. Piespiedu apstākļos labu darbu paveikt nav iespējams. Cita pieļautā kļūda, kas mēdz atgadīties neatkarīgajā testēšanā iesaistītām personām, ir testēšana, kas neņem vērā programmatūras izstrādes un ekspluatācijas tehniskos aspektus, bet gan atsevišķi formāli pieturas pie sistēmas dokumentācijas. Būtībā tā ir nepilnība profesionālā darba izpildē – testētājam ir jāņem vērā dažādi informācijas avoti un aspekti par testējamo programmatūru. Viņam ir jābūt programmatūras inženierim, kas specializējies testēšanas jomā.

Jautājums, kas nodarbina testētājus, ir izaugsmes vai karjeras veidošanas iespēja. Daudzi testēšanu saskata kā strupceļu tālākai attīstībai. Konkrēta IT uzņēmuma pieredze liecina, ka testēšanas darbs projektos ir diezgan daudzveidīgs, tādēļ rutīna nemēdz veidoties. Tālākas attīstības iespējas ir gan tādā virzienā, ka ir iespēja pāriet uz augstākiem darba devēja definētajiem kvalifikācijas līmeņiem, vai arī pēc kāda laika sākt nodarboties ar sistēmanalīzi vai programmēšanu. Cilvēks būs ieguvis pieredzi vairākās jomās, tādējādi pilnveidojot savas eksperta zināšanas vairākās programminženierijas nozarēs.

Neatkarīgās testēšanas projektos, ja tā ir palīdzēšana akcepttestēšanā, tradicionāla ir situācija, ka līdzās testēšanas grupai strādā arī pasūtītāja pārstāvji – personas, kas nākotnē būs attiecīgās sistēmas lietotāji. Šiem cilvēkiem ir labas zināšanas par sistēmas darbības sfēru, prasībām. Tiesa, viņiem parasti nav pietiekami lielas IT zināšanas, kā arī nav iespēja no viņiem prasīt pietiekami labi formulētus problēmziņojumus. Ja neatkarīgā testēšanas institūcija uzņemas metodisko atbildību par akcepttestēšanas norisi, tad tai ir jāveic arī pasūtītāja testētāju apmācība. Prasības šādu testētāju apmācībai var aprobežoties tikai tiktāl, lai pasūtītājs zinātu, kā izmantot testus, kā fiksēt izpildes rezultātus un kā pieteikt problēmziņojumus.

2.7. Testēšanas procesa kvalitātes kritēriji

Process saskaņā ar tradicionālo [IEEE 610] definīciju ir ar noteiktu mērķi veicama soļu virkne, piemēram, programmatūras izstrādes process. Savukārt, standarts ISO/IEC 12207 [76] procesu definē kā savstarpēji saistītu aktivitāšu kopu, kas pārvērš ievadi (informāciju, objektus) par izvadi. Tradicionāli procesu var iedalīt vai nu mazākos procesos, vai arī konkrētās izpildāmās darbībās jeb soļos. Kā izriet no 2.2.nodaļā minētajām testēšanas definīcijām un tālākajās nodaļās minētajām praksē pielietotajām testēšanas metodēm un veidiem, testēšanas procesam ir iespējamas 5.tabulā minētā tipa ievades un izvades. Konkrētais veids ir atkarīgs no testēšanas veida.

5.tabula. Testēšanas procesa iespējamās ievades un izvades.

Ievade	Izvade
prasības projektējumi modeļi izmaiņu pieprasījumi standarti programmas kods darbināma programmatūra projekta organizatoriskie nosacījumi	testu saraksts testēšanas rezultātu pieraksts – testžurnāls problēmziņojumi vērtējums par programmatūras kvalitāti

Testēšanas procesa kvalitāte ir vērtējama pēc iegūstamo rezultātu (izvades) atbilstības procesa dalībnieku un procesa rezultātu izmantotāju izvirzītajām prasībām. Pamatnosacījums ir tādu informācijas tipu eksistence, kas atbilst procesa ievadei un izvadei (tas nozīmē, ka katrā konkrētajā projektā var tikt izmantota sava terminoloģija, bet saturiski informācijai jāatbilst procesa mērķiem). Ir iespējams nodefinēt trīs šādas galvenās prasības, kuras var izmantot par kritērijiem testēšanas procesa kvalitātes novērtēšanai:

1. Visai procesa informācijai jābūt pieejamai konkrētu vienumu vai vienumu kopas analīzei – Informācijas pieejamība.
2. Jābūt iespējai ieinteresētajām personām pēc atbilstošo aktivitāšu norises operatīvi iegūt izvades informāciju - Operativitāte.
3. Procesā iegūtajiem rezultātiem jābūt pilnīgiem attiecībā pret ievades informāciju un izmantotās metodes sagaidāmajiem rezultātiem - Pilnīgums.

Iepriekšminētās prasības var būt apmierinātas vienā trim līmeņiem:

- zems – prasība tiek ievērota minimāli vai nemaz ;
- vidējs – prasība tiek ievērota daļēji;
- augsts – prasība tiek ievērota pilnībā.

Minētie kvalitātes kritēriji ir izvirzīti šī darba ietvaros, un autors nav novērojis līdzīgas procesa kvalitātes novērtējuma pieejas izmantošanu IT organizācijās. Iespējams detalizētākā un izvērstākā formā šāds modelis var kalpot par vienkāršu veidu kā salīdzināt divu testēšanas procesa kvalitāti. Ir iespējami arī citi kritēriji testēšanas procesa vērtēšanai. Ņemot vērā to, ka minētie kritēriji neizvirza procesam specifiskus saturiskos kritērijus, tie vienlīdz labi var būt attiecināmi arī uz citiem programmatūras procesiem. Prasību vērtējuma līmeņus var raksturot ar noteiktu punktu skaitu – 0 (zems), 1 (vidējs) un 2 (augsts). Līdz ar to, testēšanas procesa kvalitāti ir iespējams raksturot arī skaitliski ar vērtību robežās no 0 līdz 6.

2.8. Nodaļas secinājumi

Šajā nodaļā tika aprakstīti testēšanas pamatprincipi. Konkrēti, tika raksturoti jautājumi, kas ir būtiski testēšanas praktiskai pielietošanai reālos programmatūras projektos. Autors ir izcēlis jautājumus, kas izrietēja no pētījumos par testēšanas praksi Latvijā iegūtajiem rezultātiem, kā arī no praktiska darba testētāju sagatavošanā IT uzņēmumā. Šajā nolūkā ir izstrādāta mācību programma un pieeja apmācībai, kas ir veidota ar mērķi sagatavot IT uzņēmuma darbiniekus ar testēšanas pamatzināšanām un iemaņām.

Nodaļā atspoguļotie jautājumi ļauj secināt sekojošo:

- Programmatūras testēšanai ir atšķirīgi mērķi dažādos programmizstrādes prasību kontekstos, un tas nosaka dažāda tipa testēšanu.
- Neatkarīgi no pielietotajām programmizstrādes metodikām testēšanas pamatnostādnes ir vienotas, un dažādu autoru pētījumi/atziņas atšķiras tikai ar personiskajā pieredzē gūtajām atziņām konkrētās projekta situācijās.
- Lai arī Latvijā ir vēsturiski iedibināta akadēmiska interese par testēšanu, tikai lielākajos IT uzņēmumos tiek veikts sistemātisks un profesionāli orientēts darbs pie testēšanas metožu, metodikas un testētāju profesionālās izaugsmes attīstības.
- Lai arī sērijveida testēšanas automatizācijas rīki ir pieejami kopš 1990.gadu sākuma, praksē tikai neliela daļa organizāciju un projektu tos reāli izmanto. Joprojām saglabājas būtiska loma manuālajai testēšanai, un ir aktuāls jautājums par atbilstošu darbinieku atlasi un sagatavošanu.
- Testētājiem IT uzņēmumā ir nepieciešamas daudzpusīgas zināšanas ne tikai par programmatūras izstrādi, testēšanas metodēm, bet arī par lietojuma sfēru, kurā tiks izmantota testējamā programmatūra.

Nodaļas ietvaros ir izvirzīti kritēriji testēšanas procesa kvalitātes novērtēšanai, kas darba tālākajā gaitā tiks izmantoti divu testēšanas procesu salīdzinājumam.

3. Trasējamība programmatūras izstrādē

3.1. Nodaļas mērķi

Šajā nodaļā ir aprakstīti trasējamības pamatprincipi – trasējamības problēmas izcelsme, tās definīcijas programminženierijā, kā arī tās loma testēšanā. Trasējamība kā saistītu objektu kopas vai procesa īpašība ir pazīstama jau stipri sen – arī pirms programminženierijas attīstības. Trasējamība ir ieguvusi savu aktualitāti kā objektīva nepieciešamība, tirgvedības līdzeklis, uzticamības avots un papildus instruments procesa pārvaldīšanai. Interese par trasējamību nereti rodas problēmu gadījumā, bet par tās uzturēšanu ir jādomā visa projekta norises laiku. Trasējamība var kalpot kā verifikācijas līdzeklis testu vai projektēto funkciju pamatotībai.

Nodaļā ir aplūkotas aktuālās pētījumu tendences trasējamības jomā. Sniegtas literatūrā sastopamās definīcijas un klasifikācija. Trasējamības jomā sistemātiski pētījumi notiek aptuveni desmit gadu garumā, un šajā laikā ir iestrādājušās noteiktas tradīcijas, bet ne konkrēti reglamentējoši standarti. Galvenās problēmas, kas tiek pētītas, ir veidi, kā un kuru aktivitāšu ietvaros uzturēt trasējamību, metodes, kā iegūt informāciju par saitēm, kā arī raksturīgo trasējamības modeļu izpēte. Ņemot vērā trasējamība iesaistīto lielo informācijas apjomu, dabiska vēlme ir to nodrošināt ar rīku palīdzību. Tomēr materiālu analīze liecina, ka pārsvarā šāda tipa rīki ir palikuši akadēmisku pētījumu ietvaros, un nav pieejamu datu par šādu rīku reālas lietošanas pieredzi.

Problēmas ar trasējamību parādās saskarnē starp pasūtītāju un izpildītāju. Lai arī eksistē daudz dažādas prasību specificēšanas metodes, bieži viens otru nav īsti sapratis. Klienti nespēj novērtēt visu prasību specifikācijā ietverto formālismu, bet neformāli izteiktas prasības nav piemērotas programmatūras izstrādei. Trasējamības problēmas var rasties arī projektā eksistējošo atšķirīgo procesu saskarsmē – viens process var būt vairāk ieinteresēts trasējamības nodrošināšanā, bet cits – nestrādā pietiekami organizēti vai kvalitatīvi, lai varētu atļauties trasējamību. Tieši tas apstāklis, ka bieži nav labu tehnisko līdzekļu trasējamības atbalstam nosaka to, ka vieni var to uzturēt, bet citiem ir nopietni jādomā, par kādiem līdzekļiem to veikt.

3.2. Trasējamības konteksts ārpus programminženierijas

Trasējamība kā jēdziens programminženierijā ir ienācis no rūpnieciskās ražošanas. Tur tās galvenais uzdevums ir identificēt produkta izcelsmi. Šādai interesei ir dažādi iemesli – lai nodrošinātu izmantojamo sastāvdaļu viendabību (kas varētu būt būtiski, piemēram, medikamentu ražošanā), lai nesajauktu vienkopus līdzīgus produktus vai sastāvdaļas, lai atvieglotu problēmu situāciju izpēti, kā arī lai samazinātu apjomus, kādos ir jāatsauc produkti, ja ar tiem ir notikušas problēmas [82]. Tradicionālā izpratnē produktiem, kas veidoti no daudziem dažādiem materiāliem, trasējamības nodrošināšana parasti ir apjomīgs darbs, kas nereti prasījis

daudz dažādu dokumentu, cilvēkus un vietu to uzturēšanai, fiziskas identifikācijas zīmes. Dārdzības dēļ to parasti visvairāk praktizējuši augsta riska nozarēs tādās, kā medikamentu ražošana, aerokosmiskā rūpniecība, kodolenerģētgika u.tml. Reālās situācijās ne vienmēr ir nepieciešamība uzskaitīt visu komponentu izcelsmi, bet gan tikai to, kuri ar vislielāko varbūtību var ietekmēt produkta pamatfunkciju izpildi.

Protams, iepriekšminētais diezgan labi mūsdienās pakļaujas realizācijai informācijas sistēmas veidā, kur dokumentācijas vietā ir detalizēta datubāze. Piemēram, labi organizētā pārtikas ražošanas uzņēmumā pēc saražotās partijas numura ir iespējams uzzināt par izejvielu piegādātājiem un piegādāto komponenšu partijas numuriem, no kuriem var tālāk uzzināt arī atsevišķu sastāvdaļu sākotnējo izcelsmi (valsts, rajons, zemnieku saimniecība, datums).

Ja ir vēlme iepriekšminēto pieeju realizēt arī programmatūras ražošanā, tad jāņem vērā, ka tur valda mazliet citādas nostādnes, un ne viena vien lieta atšķiras no tradicionālā ražošanas procesa. Viena no būtiskākajām atšķirībām ir tā, ka programminženierijā netiek transformēts pats produkts. Katrā attīstības etapā no viena veida dokumenta vai koda tas tiek pārlikts citā dokumentā vai kodā. Tiek pārliktas idejas, informācija, modeļi, algoritmi utt. Jautājums ir par to, cik precīzi šāda pārlikšana ir notikusi, ko vispār var un vajag pārcelt attiecīgā formālisma ietvaros. Bieži ir jānovēro situācija, ka neviens no programmatūras projekta dalībniekiem konkrēti nevar norādīt uz to, kā īsti prasības saistās ar iegūto programmu, bet gan programmas pasūtītājs, gan izstrādātājs ir vienisprātis, ka programma atbilst visām prasībām.

Pēdējā laikā, kad visās jomās ir pieaugušas kvalitātes prasības, nereti būtisks ir arī produktu izcelsmes sertifikāts (jeb cita veida apliecinājums). Tas kalpo kā garantija tam, ka iegādātais produkts atbildīs mūsu vajadzībām. Programmatūras jomā tas varētu būt analoģiski tam, lai mēs zinātu, ka, piemēram, prasību par iekārtu montāžas laika reģistrāciju būtu izvirzījuši programmatūras pasūtītāja uzņēmuma ražošanas procesa pārstāvji, bet nevis reklāmas nodaļas darbinieki, vai arī gatavu programmatūras komponentu izmantošanas gadījumā ir būtiski zināt, kas tiek ne tikai apsolīts saistībā ar šo komponentu, bet kāda ir tā izmantošanas pieredze citur (vai vismaz ražotāja garantijas par tā kvalitāti).

Ne mazāk būtisks trasējamības pielietojuma aspekts ir problēmu analīze. Labi nodrošināta trasējamība ļauj precizēt iespējamos cēloņus un apzināt citus produktus, kuri arī varētu būt ietekmēti ar negatīvo rezultātu. Tieši šī iemesla dēļ ir sastopami gadījumi, kad kādi ražotāji atsauc noteiktu daļu savas produkcijas, lai tajā varētu preventīvi atrisināt konstatētās problēmas (skat., piemēram, ASV patēriņa preču drošības komisijas regulāros ziņojumus par atsaucamajiem produktiem [48]). Arī programmatūras izstrādē ir iespējamas slēptas problēmas, kuru atklāšana var izraisīt programmatūras labojumu sagatavošanu. Visbiežāk labojumu uzlikšana notiek bez būtiskiem traucējumiem esošās sistēmas darbā – ražotājs atsūta lietotājam

programmu, kas jāuzinstalē, vai arī instalētā sistēma pati regulāri pārbauda labojumu esamību. Ja runa ir par iebūvēto programmatūru, tad ražotājam jāatsauc viss produkts.

Fiziskiem produktiem tipiski trasējamība notiek attiecībā pret divu veidu atribūtiem: satura atribūti (tādi, kas ļauj izsekot produkta izejvielām un sastāvam) un procesa atribūti (neiespaido saturu, bet raksturo ražošanas procesu – izcelsmes valsts, izmantota videi draudzīga tehnoloģija u.tml.) [58]. Programmatūras gadījumā formāli šāds dalījums arī var būt spēkā, bet tomēr precizētā formā tas atbilstu tam, ka programmatūras izstrādei par pamatu ir izmantotas korektas prasības, kā arī pats izstrādes process ir kvalitatīvs (piemēram, tiek ievērots ISO 9001:2000 standarts). Pirmais minētais tieši iespaidos produkta kvalitāti (jo tas tiešā veidā saistās ar to, kā ir ievērots tas, ko no programmas sagaida klients), bet otrais minētais nav garantija iegūstamā produkta kvalitātei.

Fiziskiem objektiem aktuāla ir arī trasējamība mērījumu kontekstā. Tā atšķiras no iepriekšējās pēc sava uzdevuma nostādnes, un ir mazliet tuvāka programmatūras izstrādei – netiek transformēts pats objekts, bet viens objekts tiek veidots pēc cita objekta līdzības. Runa ir par mērvienību standartiem un instrumentu kalibrēšanu. Eksistē prasības kārtībai, kā un cik bieži ir jāveic mērinstrumentu precizitātes pārbaude un kalibrēšana. Tas tiek darīts attiecībā pret saviem etaloniem, kuri, savukārt, tiek regulāri salīdzināti ar primārajiem references objektiem. Tie tiek veidoti savstarpēji maksimāli vienādi, lai pie jebkuriem apstākļiem, kas var ietekmēt vienu no tiem, analoģiskas izmaiņas skartu arī otru. Trasējamības centrālais uzdevums ir nodrošināt atbildi uz jautājumu par to, kā tieši mērinstruments atbilst etalonam – apzināt salīdzinājumu ceļu. Jo īsāks ir šis ceļš, jo lielāka ir varbūtība, ka mērinstruments atbilst etalonam.

3.3. Trasējamības definīcijas

Klasiskais skatījums

Ņemot vērā to, ka trasējamībai programmatūras izstrādē var saskatīt dažādas funkcijas, standartos un literatūras avotos ir sastopamas atšķirīgas tās definīcijas. O.Gēteles un A.Finkelsteina klasiskajā pētījumā [59] par trasējamību bija apkopoti IT jomas profesionāļu viedokļi par to, kā definēt trasējamību. Autori saskatīja četru veidu definīcijas:

- mērķa vadīta (nosaka, kas būtu jādara): „*spēja neatkāpties no darbības sfēras, projekta aptvēruma un apstiprinātajām būtiskākajām prasībām*";
- risinājuma vadīta (nosaka, kā būtu jādara): „*spēja izsekot no vienas entītijas uz citu, bastoties uz dotajām semantiskajām attiecībām*";
- informācijas vadīta (uzsverot trasējamo informāciju): „*spēja sasaistīt funkciju, datus, prasības un jebkuru tekstu sākotnējās prasībās, kas attiecas viens uz otru*";

- virziena vadīta (uzsverot trasējamības virzienus): *„spēja sekot konkrētam vienumam no dzīves cikla fāzes ievada līdz šīs fāzes izvadam"*.

Programminženierijā bieži aplūko nevis trasējamību vispār, bet jēdzienu „prasību trasējamība". Viena no klasiskajām definīcijām [59] definē trasējamību tieši minētajā kontekstā:

> *„Prasību trasējamība norāda uz spēju aprakstīt un sekot prasības dzīvei gan turp, gan atpakaļ virzienā (t.i. no tās izcelsmes caur izstrādi un specifikāciju uz tam sekojošu ieviešanu un izmantošanu, kā arī caur visiem detalizēšanas darbiem un iterācijām ikvienā no šīm fāzēm)".*

Vairākos ar trasējamību saistītos pētījumos autori min prasību trasējamību, bet faktiski runa ir par trasējamību arī gadījumos, kad pašas prasības netiek aplūkotas. Standarts ANSI/IEEE Standard 830-1984 [25] (un tam atbilstošais Latvijas standarts LVS 68:1996 [94]) trasējamību definē:

> *„Programmatūras prasību specifikācija (PPS) ir trasējama, ja tā atvieglo sekošanu katrai prasībai turpmākās izstrādēs vai uzlabošanas dokumentācijā. Tiek rekomendēti divi trasējamības tipi: (1) atpakaļejošā trasējamība (uz iepriekšējām izstrādes stadijām) ir atkarīga no katras prasības skaidras sasaistes ar tās avotu iepriekšējos dokumentos (2) Turpejošā trasējamība (uz visiem saskaņā ar PPS rakstītiem dokumentiem) ir atkarīga no katras PPS prasības, kurai ir unikāls vārds vai atsauces numurs".*

Lai arī prasības piešķir saturu programmatūras izstrādei, tas nav vienīgais informācijas tips projektā, attiecībā pret kuru projekta darbiniekiem ir būtiski iegūt saišu informāciju. Projektā eksistē vairāki procesi, kuri veic atbalstošu funkciju, un nebūtu uzskatāmi tieši saistāmiem ar prasībām. Tādēļ atbilstošākas ir standartā IEEE 610 minētās divas trasējamības definīcijas versijas [67]:

> *„(1) Pakāpe, kādā var tikt nodibinātas attiecības starp izstrādes procesa diviem vai vairāk produktiem, īpaši produktiem, kam ir priekšgājēja-sekotāja vai vadītāja-pakļautā attiecības vienam pret otru; piemēram, pakāpe, kādā savstarpēji atbilst konkrētas programmatūras komponentes prasības un projektējums.*
> *(2) Pakāpe, kādā katrs programmatūras izstrādes produkta elements pamato savu eksistēšanu; piemēram, pakāpe, kādā katrs burbuļu diagrammas elements atsaucas uz prasībām, ko tas apmierina."*

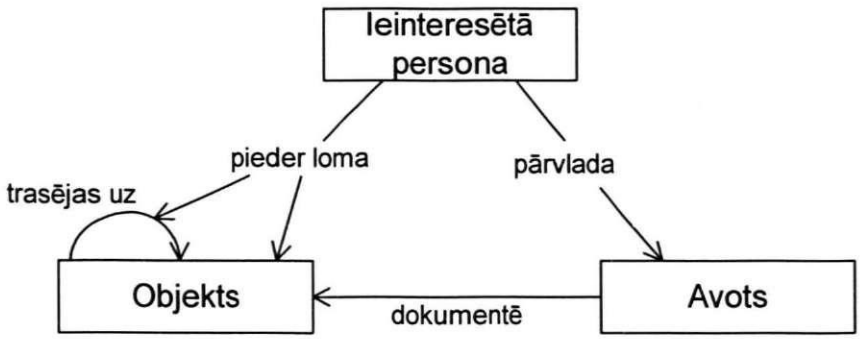
Ja aplūkojam, starp ko notiek trasējamība, tad tas pārsvarā ir atkarīgs no aplūkotā aptvēruma un attiecīgajā procesā izmantotās terminoloģijas. Raksturīgākie ir:

- programmatūras artefakti [131]
- programmatūras dokumenti [111]
- produkta fragmenti [134]

- objekti [123]
- izstrādes procesa produkti [67]

No saturiskās analīzes var secināt, ka visos gadījumos tiek aplūkoti vieni un tie paši projektā eksistējošie vienumi – dokumenti, pieraksti u.tml. Terminoloģiskā daudzveidība liecina, ka arī trasējamības pamatjēdzieni vēl ir attīstības procesā, un tie ir aplūkojami dažādos kontekstos.

[123] ir nodefinēts trasējamības metamodelis, kas raksturo trasējamības būtiskākās sastāvdaļas (1.att). Objekts ir programmatūras artefakti jeb programmatūras izstrādes procesa ievades uz izvades. Trasējamība starp objektiem tiek parādīta ar saiti „trasējas uz". Ieinteresētajām personām pieder noteikta loma objektu izveidē, izmantošanā un uzturēšanā. Personas operē gan ar pašiem objektiem, gan to savstarpējām saitēm. Objektus dokumentē dažādi avoti (projekta dokumenti u.c.), un šos avotus pārvalda ieinteresētās personas.



1.attēls. Trasējamības metamodelis.

Īpašu programminženierijas standartu, kas ir veltīti trasējamībai, nav. Tā tiek minēta kā viena no prasībām dažādos programminženierijā izmantotajos standartos, piemēram, ISO/IEC 12207, ISO 9001:2000, IEEE J-STD-016 un TickIT Guide Issue 5.0. Trasējamības forma var izrietēt no programmatūras procesu modeļu lietojuma, kuri var veidoties konkrētu standartu izmantošanas ietekmē, piemēram, ISO 9001, SEI CMM, Bootstrap u.c..

Jaunā definīcija

Ņemot vērā iepriekš minēto terminoloģisko daudzveidību un to, ka šajā darbā trasējamība tiek aplūkota tieši programmatūras projektu kontekstā, autors šī darba ietvaros aplūko jēdzienu „projekta vienums". Projekta vienuma jēdziens ietver visus tos projektā izmantojamos un radušos informācijas tipus, kas attiecas uz produkta īpašībām, saistās ar projekta notikumiem un satur projekta norisei nepieciešamo informāciju. Projekta vienums ir jebkas, ko projekts savā skatījumā uztver kā atdalāmu vienumu – prasība, modelis, modeļa elements, funkcija, klase, dokuments, dokumenta nodaļa, problēmziņojums, izmaiņu pieprasījums, intervijas protokols, apskates protokols utt.

Starp projekta vienumiem eksistē trasējamība, ja tie ir tieši vai pastarpināti saistīti ar saitēm. Atkarībā no tā, kāda tipa vienumi ir savstarpēji saistīti un kāds bija sasaistes konteksts, var iezīmēt divu galveno tipu saites: evolūcijas un notikuma.

Starp diviem projekta vienumiem eksistē evolūcijas saite, ja viens no vienumiem ir veidots, balstoties uz otra vienuma eksistences faktu, saturu vai īpašībām un ja šāda atkarība ir reģistrēta projekta pierakstos.

Evolūcijas saites parāda stingru saiti starp diviem vienumiem. Šo saišu gadījumā izmaiņas vienumā A, izsauks izmaiņas vienumā B, ja tas būs produkta attīstības (jeb evolūcijas) laikā atvasināts no A. Būtiski ir tas, ka saitēm ir virziens: iepriekšminētajā situācijā izmaiņas vienumā B nenozīmē, ka notiek izmaiņas arī vienumā A. Piemēram, izmaiņas prasībās izsauks izmaiņas tām piesaistītajās funkcijās, bet izmaiņas funkcijās prasības neiespaidos.

Starp diviem projekta vienumiem eksistē notikuma saite, ja viens no vienumiem satur norādi uz otru vienumu un ja šī norāde ir reģistrēta projekta pierakstos.

Notikuma saites saista divus vienumus nestingrā formā. Šādas saites veidojas tad, ja eksistē vienumi A un B, kur B nav projekta attīstības gaitā iegūta vienuma A transformācija, bet kas ir radies projekta notikuma rezultātā, satur norādi uz A. Šajā gadījumā neviena no izmaiņām jebkurā no saistītajiem vienumiem neizraisa izmaiņas otrā. Saite ir kā apliecinājums tam, ka konkrētā notikuma laikā abi vienumi bija saistīti, viens atsaucās uz otru u.tml. Notikuma saitēm ir tā specifika, ka tās raksturo saiti konkrētā notikuma kontekstā. Piemēram, testžurnāla ieraksts satur norādi uz testu – tā ir notikuma saite, kas parāda, ka konkrētajā brīdī notika testēšana, un ir iegūts kāds rezultāts. Projekta notikumi ir, piemēram, projekta vienumu apskate, testu izpilde, problēmas konstatēšana utt. Notikuma saitēm ir virziens, bet tas neiespaido iespēju veikt trasējamību. Virziens raksturo to, ka viens no vienumiem ir veidojies hronoloģiski agrāk par otru.

Darbā izmantotā trasējamības definīcija:

Trasējamība – Projekta vienumu trasējamība – spēja izsekot no viena projekta vienuma pie cita ar evolūcijas vai notikuma saišu palīdzību.

Saskaņā ar šo definīciju, projekta vienumu trasējamība neaplūko tādas patvaļīgi definētas saites starp vienumiem, kas parāda līdzību („ir līdzīgs") vai kādas citas subjektīvi piešķirtas attiecības, kuras neizriet no vienumu attiecībām projektā.

Starp projekta vienumiem var eksistēt šādas galvenās attiecības (kas ir aktuālas arī asociācijās UML klašu un objektu modelēšanas ietvaros [112]):

- vienkārša saite starp diviem vienumiem,
- vienums ir daļa no cita vienuma (agregācija),
- vienums ir speciāls gadījums no cita vienuma (specializācija).

Vienkāršā saite var būt evolucionāra vai notikuma saite, bet agregācijas un specializācijas gadījumā saites ir evolucionāras.

Lai arī kāds vārds („atkarīgs no", „seko no", „pieder pie" utt.) tiek piešķirts saitēm, ir būtisks ir tieši saites tips, jo tas nosaka trasējamības raksturu – izmaiņu nepieciešamība no viena vienuma pie cita vienuma var nonākt tikai ar evolūcijas saišu starpniecību, ja tās ir orientētas virzienā uz izmaiņām pakļauto vienumu.

Projekta vienumu trasējamības pieeja neuzliek ierobežojumus uz to, kāda var eksistēt saišu kardinalitāte (1:1, 1:n, m:n) starp divām projekta vienumu kopām, piemēram, prasībām un funkcijām.

3.4. Trasējamības klasifikācija

Trasējamībai nav veltīti atsevišķi standarti vai starptautiski akceptētas programmatūras projektu vadlīnijas, tomēr publikācijās ir definēti vairāki veidi, kā būtu lietderīgi klasificēt novērojamo trasējamību. Visplašāk izplatītais ir pēc trasējamības piederības attiecībā pret prasību specifikāciju. Retāk tiek aplūkots trasējamības virziens un ieviests divu dimensiju jēdziens. Autors saskata nozīmi aplūkot arī to, cik plaša ir trasējamība vairāku vienumu kontekstā.

Trasējamības novietojums attiecībā pret prasību specifikāciju

Pēc trasējamības novietojuma iedalījums ir šāds [59]:

- Pre-RS – trasējamība, kas saistās ar prasībām pirms to iekļaušanas prasību specifikācijā;
- Post-RS – trasējamība, kas veidojas kā rezultāts no prasību specifikācijā iekļautām prasībām.

Galvenā motivācija šādam dalījumam veidojas no tā, ka prasības ir programmatūras projekta pamats. Ir projekti, kuri var sākt darbu saskaņā ar nodefinētajām prasībām. Šādās situācijās nav būtiski, kā ir veidojusies specifikācija, un atliek tikai uz tās balstīt pārējo dokumentu un projekta vienumu izveidi – notiek Post-RS trasējamība. Savukārt, ja projekta ietvaros ir jāapkopo un jānodefinē prasības, tad ir būtiski arī saprast, no kurienes ir radušās atbilstošās prasības – no standartiem, potenciālajiem lietotājiem, likumdošanas, iepriekšējās izstrādes pieredzes utt. – šādos gadījumos ir aktuāla Pre-RS trasējamība. Programmatūras projektā var pastāvēt viena no minētajām, vai arī abas trasējamības. Galvenā būtiskā atšķirība ir tā, ka Pre-RS nespēlē būtisku lomu produkta implementācijas laikā, jo tajā brīdī prasības netiek apšaubītas, bet gan pārvēstas citā, darbināmā, formā. Pastāv iespēja, ka testēšanas laikā var pacelties jautājums par prasību izcelsmi. Analoģiska interese var parādīties produkta eksperimentālās ekspluatācijas vai akcepttestēšanas laikā.

Pre-RS trasējamība ļauj prasību definēšanas laikā pieņemt lēmumus par to prasību būtiskumu un nepieciešamību tās iekļaut specifikācijā, identificēt prasības, kuras var neprognozējami mainīties un ietekmēt produkta lietošanas iespēju (piem.,

likumdošanas izmaiņas). Raksturīga projektu situācija ir tāda, ka prasību specifikācija eksistē, bet implementācijas laikā kļūst skaidrs, ka tā ir nepilnīga. Tipiska rīcība ir iekļaut papildus noskaidroto informāciju nākošajos veidojamajos dokumentos. Ja projektā eksistētu atbilstošs mehānisms prasību izcelsmes fiksēšanai, būtu iespēja novērtēt pirmavotu un precizēt izmaiņu nepieciešamību un atbilstību.

Pēc trasējamības virziena

Nosacīti var uzskatīt, ka ir iespējams izsekot tam, kā viens projekta vienums ir iegūts no cita, vai arī – kā no viena vienuma varētu iegūt otru. Pirmajā gadījumā ir atpakaļejoša trasējamība, lai saprastu projekta vienuma izcelsmi, eksistences pamatojumu, kā arī novērtētu iespējamos gadījumus, kad vienums būtu jāmaina. Otrs virziens, turpejošais, raksturo iespēju saprast, kas ir atkarīgs no attiecīgā vienuma un kādas varētu būt šī vienuma mainīšanas sekas. Sasaistes virziens raksturo atkarības virzienu, kā arī vairumā gadījumu raksturo trasējamībā iesaistīto vienumu attiecības no hronoloģiskā viedokļa. Projektā iesaistītās personas interesējošais virziens var diktēt arī to, kā veikt trasējamības informācijas pierakstu. Ja tiek izmantots vienkāršas tabulas paņēmiens, jābūt iespējai to atbilstoši pārkārtot pēc katras no iesaistītajām vienumu grupām. Teksta dokumentos varētu veikt brīvā teksta meklēšanu, bet hipersaišu dokumentos būtu jāparūpējas par atgriezeniskām saitēm. Ja informācija tiek uzkrāta datubāzē, tad saišu virziena jautājums kļūst tehniski vienkārši risināms.

Trasējamības dimensija

Tradicionāli tiek aplūkota tā trasējamība, kas aplūko saites starp dažādiem projekta vienumiem, bet retāk – kā viens vienums attīstās laika gaitā. Viena dimensija ir vienuma saistība ar citiem vienumiem, bet otra – vienuma versijas. Faktiski runa varētu būt par to, ka katram vienumam projekta laikā var eksistēt vairākas versijas. Jāņem vērā tas, ka ne vienmēr objektam versijas attīstās lineāri. Vispārīgā gadījumā var veidoties koks – ja kāda versija ir izrādījusies nepiemērota, tad no kāda iepriekš eksistējoša varianta paralēli var tikt atvasināta jauna versija, kas atsevišķos gadījumos var veidoties no nepieciešamības uzturēt paralēli vienu konkrētu komponenti vecajam produktam un šo pašu komponenti attīstīt arī jaunajam produktam. Literatūrā faktiski jautājums par trasējamību un attīstību laikā netiek analizēts. Ir minēts konceptuālā līmenī [46], ka var trasējamību var aplūkot kā vertikālo (starp artefaktiem) un horizontālo (vienu un to pašu artefaktu attīstība laikā).

Projektā esošo dokumentu vai koda versiju uzturēšana parasti ir konfigurācijas pārvaldības procesa sastāvdaļa. Vienumiem, kas ietilpst dokumentos, versijas tiek uzturētas ar dokumentēšanas vides līdzekļiem.

Plašums

Mazāk būtiska, bet tomēr raksturojoša trasējamības īpašība ir tā, cik plaši konkrētā projektā tiek veikta trasējamība. Pastāv iespēja, ka labi tiek uzturētas trasējamības saites viena vienuma tipa ietvaros, bet pārējie projekta vienumi nav

sasaistīti. Piemēram, varētu būt definēta sasaiste tikai testēšanas procesa ietvaros, vai arī tikai prasību analīzes ietvaros. Ja ir reģistrēta trasējamība starp dažāda tipa vienumiem, tad plašuma jēdzienu var aplūkot kā vienumu tipu skaitu, kādus maksimāli var sasniegt no jebkura brīvi izvēlēta vienuma.

3.5. Pētījumu virzieni

Vērtējot trasējamības virzienus, tika izmantota fiziskajās bibliotēkās pieejamā literatūra, IEEE elektroniskā bibliotēka [71], CiteSeer [45] zinātnisko rakstu bibliotēka, kā arī citi zinātniskās informācijas avoti.

Lai arī pats par sevi trasējamības jēdziens nav jauns, līdzīgi kā daudzi citi kvalitātes principi, tas programminženierijā nav parādījies uzreiz. Kā jau bija minēts iepriekš, citās nozarēs trasējamība ir obligāta prasība, kur tas ir iestrādāts dažāda veida procesos. Ņemot vērā to, ka programminženierijā kā jaunā nozarē daudzas cilvēces mantojuma lietas tiek pielāgotas programmu formā (matemātika, vizuālā māksla) un programmatūras projektu norisei (plānošana, darba grupu pārvaldība), arī trasējamība tiek aplūkota tieši programmatūras kontekstā. Šis konteksts izvirza vairākas jaunas problēmas, kurām tiek meklēti risinājumi.

Ir sastopama virkne rakstu, kas aplūko trasējamības nozīmi programmizstrādes kontekstā [23], [47], [86], vai arī vienkārši deklarē, ka tas ir būtiski [133], [141]. [52] ir secināts, ka trasējamību ir jāsekmē un pat jāuzspiež prasību specificēšanas procesa laikā.

Trasējamības jautājumu izpēte aktualizējās pirms mazliet vairāk kā desmit gadiem, kad pētnieku grupa O.Gēteles un A.Finkelsteina [59] vadībā sāka analizēt kopīgo un atšķirīgo dažāda tipa projektos - kas īsti ir prasību trasējamība un kādus galvenos veidus varētu iezīmēt. 1994.gadā veiktais pētījums atspoguļoja literatūras analīzi, diskusijas ar programminženierijas praktiķiem, aptaujas un intervijas. Viena no būtiskām atziņām bija tā, ka trasējamības problēma eksistē, bet katrā projektā tā parādās atšķirīgā formā, un tas izriet no katra projekta individuālām darba metodēm un vajadzībām. Kā galvenā problēma tika identificēts pre-RS trasējamības vājais nodrošinājums, kā arī informācijas pieejamība un apmaiņa starp iesaistītajām pusēm.

Ir publikācijas, kas iezīmē potenciālus pētniecības virzienus trasējamībā. Piemēram, [30] aplūko to, kādas problēmas ir identificētas reālos programmizstrādes uzņēmumos 2001.gadā. Tiek secināts, ka kopš pirmās sistemātiskās problēmu eksponēšanas Gotel un Finkelstein darbā ir notikušas vairākas izmaiņas. Proti, ir uzlabojušies komunikāciju līdzekļi, tajā skaitā attīstījusies interneta vide un parādījušās dažādu prasību pārvaldības rīku iespējas, bet tajā pat laikā parādās jaunas informācijas pārvaldīšanas problēmas - ar rīkiem ir sarežģīti strādāt, un šī iemesla dēļ visa komunikācija var iet caur vienu atbildīgo personu. Īpaši ir aktualizējies dalītās (*distributed*) trasējamības jautājums - kā vairāku sadarbības partneru gadījumā nodrošināt efektīvu komunikāciju. Jauna problēma iezīmējas saistībā ar

ārpakalpojumu sniegšanu - kā veikt prasību un citas projekta informācijas apmaiņu ar pasūtītāju. Kā potenciālie pētījumu virzieni tiek iezīmēti: projektējuma risinājumu bibliotēkas veidošana, dalītās trasējamības atbalsts, uz izmaiņām operatīvi reaģējoša trasējamības repozitorija, kā arī neformālas informācijas reģistrēšanas un klasifikācijas atbalsts, jo tā ir cilvēkam raksturīga pieeja jaunu etapu sākuma posmos. Jāsecina, ka šajā darbā risinātās problēmas virziens daļēji atbilst atsevišķiem iepriekšminētajiem virzieniem – projektiem tiek piedāvāta operatīvi atjaunojama repozitorija un pastāv iespēja reģistrēt informāciju dažādos formālisma līmeņos.

Kā pētījumu trasējamības īpašībām speciālos gadījumos var minēt darbu [114], kur trasējamība tiek aplūkota formālo metožu kontekstā. Šajā nolūkā tiek ieviests papildus dalījums trasējamībai - pre-Formal RS un post-Formal RS trasējamība, kas akcentē formālo metožu procesa specifiku.

Programmatūras projektos eksistējošās trasējamības pamatīpašības tiek detalizēti pētītas Ramesh un Jarke darbā [123]. Autori ir veikuši pētījumus projektos, programminženierijas profesionāļu vidū, analizējuši rīku iespējas, kā arī izstrādājuši modeļus, kas atbilst trasējamībai projektā - trasējamības modeļus.

Ir virkne potenciālo pētījumu virzienu, kurus autors vēlējās redzēt kā risinātus, bet tomēr šiem netika atrastas norādes par detalizētiem pētījumiem. Proti, trasējamībai ir trīs aspekti, kas varētu būt kā raksturotāji konkrētas sistēmas, projekta vai procesa ietvaros eksistējošajam trasējamības veidam. Tie ir:

- plašums – cik daudz un kāda tipa pazīmes tiek uzkrātas un būtu jāuzkrāj par vienumiem;
- dziļums - cik tālu ir iespējams un būtu jāspēj veikt izsekošanu no viena vienuma tipa līdz citam;
- precizitāte - kāds ir nepieciešamais detalizācijas līmenis, lai trasējamību izmantojošās personas vai procesi varētu pateikt, tieši kā viens vienums saistās citu – daļēji, pilnībā vai ar speciāliem nosacījumiem.

3.6. Trasējamība un objektorientācija

Ņemot vērā, ka pēdējā desmitgadē programmatūras izstrādē dominē objektorientācijas paradigmas jēdziens, ir veikta virkne pētījumu par trasējamību tieši objektorientācijas apstākļos. Vairāki principi ir izmantojami arī cita veida programmatūrā.

[22] risina trasējamības problēmu gadījumam, kad ir jāsasaista objektotientēts projektējums ar programmas kodu. Tiek izmantotas XML valodas iespējas, lai izveidotu saišu informāciju starp projektējuma un JavaML koda elementiem. [27] objektorientēta projektējuma un objektorientēta koda sasaistei izmanto AOL (Abstract Object Language). Autori risina problēmu, kad implementācijā kods atšķiras no projektējumā paredzētā. Risinājumā trasējamības saites veidojas koda un

projektējuma salīdzināšanas brīdī. Rezultāts ir līdzības novērtējums, lai programmētājs varētu novērtēt koda atbilstību projektējumam.

Objektorientētu programmu uzturēšanā būtiski ir saprast, vai izmaiņas kādā klasē neizraisīs neprognozētu funkcionalitātes izmaiņu citos programmatūras apgabalos. Ja ar uzturēšanu nodarbojas vairāku cilvēku komanda, ir svarīgi arī, lai veiktās izmaiņas nav pretrunīgas. [26] aplūko trasējamību starp dažādām objektorientētas programmas versijām objektu definīciju un saskarņu līmenī. Autori piedāvā metodi, kā novērtēt izmaiņu apjomu, noteikt atbilstību kādiem iepriekš izvirzītiem objektorientētā koda kritērijiem, kā arī nodrošinātu saskaņota koda uzturēšanu.

[32] ir konstatējuši, ka objektorientētā modelēšanā bieži netiek parādīti projektētāju lēmumi, kādēļ tiek definēti konkrētie objekti tieši minētajā veidā, bet ne citādi. Tas var aktualizēties gadījumos, ja pie vienas sistēmas specificēšanas strādā vairāki cilvēki vienlaicīgi. Svarīgi ir saglabāt izvēlēto risinājumu pamatojumu, lai to varētu sasaistīt ar sākotnējām prasībām. Minēto objektorientētas programmatūras izstrādes problēmu risināšanas kontekstā piedāvā izmantot izstrādes stadijā esošo rīku RARE.

Lai uzlabotu trasējamību objektorientētos projektos, tiek piedāvāts TOOQE (Traceability for Object-Oriented Quality Engineering) process [46]. Pieejas pamatā ir piedāvājums veikt izstrādi nelielu iterāciju formā, kur trasējamības nodrošināšanas nolūkā katras ietvaros jāintegrē dažāda tipa modeļi, kā arī jāveic testēšana.

[116] visu produkta informāciju uztver kā objektus, un trasējamības pamatā ir skatījums uz projekta informāciju kā savstarpēji objektiem. Pieejas realizācijai izmanto rīku TOOR.

Literatūrā sastopamie rezultāti liecina par to, ka bieži objektorientācija tiek minēta tā iemesla dēļ, ka šāda tipa pieeja tiek uzskatīta par modernu, bet tikai atsevišķos gadījumos pēc būtības lomu spēlē tas fakts, ka programmā kods strukturēts savstarpēji saistītās klasēs ar savām metodēm un atribūtiem.

3.7. Trasējamība starp programmatūras artefaktiem

Ja mēs izsekojam saitēm, kas saista, piemēram, sākotnējās prasības un kodu, tad būtībā koda elementiem vai moduļiem kā tādiem nav tiešas saistības ar lietotāju skatījumā formulētajām prasībām. Faktiski programmas kodā ir daudz tehnisku lietu, ar kuru palīdzību ir iespējams realizēt kādu prasību aspektu. Jo programmēšanas valoda ir augstāka līmeņa, jo kods būs tuvāks prasībām vai augsta līmeņa projektējumam.

Ja aplūko pētījumu virzienus, tad populārākie trasējamības veidi ir no prasībām vai projektējuma uz kodu.

[28] aplūko metodes, kā varētu nodrošināt trasējamību starp C++ kodu un dokumentāciju, kā arī starp Java kodu funkcionālajām prasībām (iegūst informāciju, to attēlo vektoru telpā un salīdzina).

[65] analizē situāciju, kad sistēmai ir definētas nefunkcionālās prasības (piemēram, drošība vai veiktspēja), un tās attiecas uz faktiski visu programmatūru kopumā, kā arī katru individuālo elementu (moduli, funkciju, klasi, utt.). Pārāk liela saišu uzturēšana nav auglīga, jo tās padara trasējamības informāciju nepārskatāmu. Risinājums ir speciālu trasējamības saišu ieviešana.

[90] apraksta konfigurācijas pārvaldības un versiju vadības prototipa sistēmu, kas nodrošina augstas detalizācijas trasējamību starp sistēmā uzturamajiem vienumiem (piem., kodā vai projektējumā līdz konkrētai funkcijai, vai arī prasību dokumentā konkrēta prasība, rindkopa). Trasējamība tiek aplūkota kā iespēja izsekot katras smalkākās aplūkojamās vienības evolūcijai abos virzienos – izmaiņas no versijas uz versiju.

[111] aplūko pieeju, kā programmatūras dokumentos var organizēt trasējamību ar hiperteksta palīdzību tā, lai varētu uzturēt informāciju par savstarpēji atbilstošiem programmatūras dokumentiem. Atbilstības uzturēšanai tiek izmantoti īpaši nodefinēti tagi.

Trasējamība prasību ietvaros

Lai arī projekta mērogā būtiski ir nodrošināt izsekošanu saitēm starp dažāda tipa programmatūras projekta artefaktiem (vienumu tipiem), ne mazāk būtiski ir pārvaldīt saites konkrēta vienumu tipa ietvaros.

Rakstā [53] tiek piedāvāts papildināt saišu informāciju ar noteikta veida likumiem vai ierobežojumiem. Tas ļautu precīzāk pārvaldīt savstarpējās ietekmes, sekot konkrēta vienuma izcelsmei, kā arī varētu palīdzēt pārklājuma analīzei.

[87] ierosina papildus tradicionālajai prasību definēšanai tās integrēt ar scenārijiem. Tas nozīmē dažāda tipa prasību savstarpējo sasaisti, nepazaudējot trasējamības informāciju. Trasējamības uzlabošanas nolūkos [88] iesaka uzturēt prasības scenāriju veidā. Konkrēti, tas ir devis uzlabojumus pre-RS trasējamībais un komunikācijai starp projektā iesaistītajām pusēm.

Par mūsdienu prasību modelēšanas standartu var uzskatīt UML valodu [112]. Viena no praktiskas lietošanas problēmām ir uzskatāmas trasējamības iegūšana starp dažādu UML modeļu elementiem. Ir veikti atsevišķi pētījumi šīs problēmas risinājumam. Piemēram, [137] aplūko trasējamību modeļu un modeļu elementu līmeņos, kā arī analizē izmaiņu ietekmi prasību ietvaros. Līdzīgu problēmu analizē [139], kur piedāvā risinājumu ar eksperimentāla rīka palīdzību.

[61] analizē iespēju, kā grupēt trasētās prasības atbilstoši kvalitātes kritērijiem tādiem, kā funkcionalitāte, drošība, uzticamība, lietojamība u.c. Ja apzinātās prasības

atbilstoši klasificē un tām nodefinē papildus saites, var iegūt papildus informāciju prasību analīzei.

[64] piedāvā risinājumu prasību apskašu procesa uzlabošanai, bagātinot jaunās sistēmas specifikācijas ar piemēriem, kas balstīti uz eksistējošo praksi (ja tiek izstrādāta jauna sistēma, kas aizvieto veco). Tas var ļaut apskates dalībniekiem labāk saprast, ar ko jaunā sistēma atšķiras, un mazinās risku, ka kādas prasības ir aizmirstas vai ir liekas. Rakstā parāda rīka PRO-ART lietojumu.

Trasējamības metožu pielietošanas prakse industrijā

Ir sastopama virkne materiālu, kas atspoguļo trasējamību konkrētu projektu, procesu vai programmatūras sistēmu pielietojumu gadījumos. Lielākajā daļā šādu pētījumu ir norādes uz to, kā varētu šo problēmu risināt ar metodiskiem līdzekļiem, bet ne vienmēr ir norādes uz veidiem, kā var izmantot konkrētus rīkus.

[21] apraksta konkrētu risinājumu, kur dzelzceļa vadības sistēmas projektā prasību specifikācija bija pamatā balstīta uz savstarpēji saistīties lietojumgadījumiem (*use cases*). Šī informācija tika glabāta prasību pārvaldības rīkā DOORS, kas nodrošināja iespēju norādīt prasību saites. Prasību lasāmības nolūkos šī informācija tika izeksportēta hiperteksta formātā, lai dokumenta lasītāji var ērti pārlēkt starp saistītajiem aprakstiem.

Piemērs ar trasējamības nodrošināšanu starp vairākiem sistēmu modelēšanas rīkiem ir parādīts aviācijas rūpniecības gadījumam [104]. Šajā nolūkā tika nodefinēti modelēšanas principi un modeļu informācijas apstrādes procedūras ar nolūku uzturēt saišu informāciju starp modeļiem un to elementiem. Viens no pamatelementiem ir definētais metamodelis, kas apraksta plānoto trasējamību.

Nozīmīgs pielietojums dažāda veida trasējamības risinājumiem ir programmatūras produktu vai pakalpojumu saimes uzturēšanā. Šīs saimes ir raksturīgas ar to, ka organizācija dažādiem mērķiem (vai pasūtītājiem) veido programmatūru ar mazliet atšķirīgām iespējām, saskarni, saderību utt. Lai arī cik neliela būtu katra atšķirība, izstrādātājam ir jārūpējas par katras variācijas uzturēšanu nākotnē. Ja tiek veidots jauns vai mainīts esošais elements, ir būtiski saprast, kuri lietotāji tiks iespaidoti. Šādas problēmas iespējamais risinājums ir aplūkots pētījumā [107]. Pētījuma autori izvirza tēzi, ka īpaša nozīme ir efektīvām trasējamības uzturēšanas metodēm.

Trasējamības modeļi

Projektā vienumi nav saistīti patvaļīgi. Ir saites, kas ir pieļaujamas, un ir saites, kas nav iespējamas. Katram projektam pamatā ir tipveida vienumi un tipveida atļautās saites. Projektā eksistējošās tipveida saites var raksturot ar metamodeļa palīdzību. Trasējamības kontekstā ir kļuvis par tradīciju to saukt par trasējamības modeli:

> *Trasējamības modelis – metamodelis, kas apraksta/reprezentē projektā esošos vienumus un saites starp tiem.*

Katram programmatūras procesam ir iespējams iezīmēt savu trasējamības modeli. Ja projektu uztveram kā procesu kopu, tad projekta trasējamības modelis ir procesu trasējamības modeļu apvienojums. Jāņem vērā, ka arī viena un tā paša standarta (organizācijas, projekta vai procesa līmenī) pielietošana var būt par pamatu dažāda paveida trasējamības modeļiem.

Konkrēts trasējamības modelis no militārās jomas visas organizācijas sistēmas līmenī ir aplūkots, piemēram, pētījumā [122]. Tas aptver 13 vienumu tipus un 20 saišu tipus. Autori izpētīja dažāda organizatoriskā līmeņa darbinieku viedokli par trasējamību. Augšējai vadībai tā šķita konceptuāli svarīga, bet reāli visvairāk to izmantoja projektētāji un testētāji.

Konkrēts trasējamības modeļa piemērs tiek aplūkots arī pētījumā par prasību izmaiņu pārvaldību iegultu sistēmu gadījumā [138]. Tiek identificēta virkne vienumu tipu, kas ir iedalāmi trijās grupās: konceptuālais sistēmas modelis, konceptuālais dokumentācijas modelis un sistēmas prasības. Piedāvātais risinājums palīdz analizēt nepieciešamās izmaiņas, kuras iniciē izmaiņas vienā no sistēmas konceptuālajiem modeļiem.

Praksē ir gadījumi, kad trasējamības modelis apzināti tiek veidots nepilnīgs. Ja aplūkojam ne pārāk laba parauga, bet tomēr reāli bieži sastopamu programmatūras projektu, kas laika gaitā mainās, tam ir neprecīzi definētas attiecības ar pasūtītāju, netiek ievēroti visi programminženierijas labās prakses principi, ir mainīgas prasības, tad šim projektam varētu nebūt vēlēšanās pakļaut trasējamībai tos projekta vienumus, kuru loma projektā nav stingri definēta (vai arī to dzīves laiks ir īslaicīgs, bez tālejošām sekām projekta norisei). Šādā veidā projekts varētu būt ieinteresēts pakļaut trasējamībai ne visus projektā eksistējošos informācijas tipus, kas nozīmē, ka trasējamības modelis būs mazāka apjoma nekā tas būtu vēlams.

3.8. Trasējamības informācijas iegūšana un reģistrēšana

No praktiskā lietojuma viedokļa noderīgas varētu būt tādas pieejas, kas ļautu automatizēti un precīzi restaurēt eksistējošās trasējamības saites no konkrētiem projekta vienumiem. Pēc būtības šāda pieeja ir sagatavošanās darbs tālākajām operācijām ar trasējamības informāciju. Ja trasējamības informācijas uzturēšana ir pilnībā atkarīga no kāda projekta darbinieka manuālā darba, tad pastāv risks, ka pienākumu nepienācīgas izpildes rezultātā nebūs nodrošināta trasējamība. Lai arī no šāda tipa problēmām nevar izvairīties praktiski ikvienā darbā, uzdevumos, kuri pakļaujas automatizēšanai ir iespēja samazināt nepilnīgas vai neprecīzas informācijas rašanās iespēju. Šajā nolūkā ir veikti pētījumi par to, ko no trasējamības informācijas iegūšanas un reģistrēšanas varētu veikt automātiski. Trasējamības informācijas iegūšana aktualizējas saistībā ar šādu projekta darbu realizēšanu:

- esošās situācijas novērtējums (piem., pārliecināšanās, vai saites joprojām ir aktuālas) un

- reversā inženierija.

Ir priekšlikumi veikt cilvēku dzīvajā valodā rakstīto tekstu analīzi [26], [28] Mērķis būtu noteikt šo aprakstu piederību pie koda elementiem. Protams, ir jāeksistē kaut kādām sasaistošām pazīmēm. Šajā gadījumā tie ir kopīgi atslēgvārdi koda objektiem, funkcijām un komentāriem ar dokumentācijā minētajiem. Līdzīga problēma ir risināta [102] izstrādātajā pieejā, kas veiktajos eksperimentos demonstrē tikpat augstas precizitātes trasējamības saišu kopas izveidi, kā [28]. Tās pamatā ir atrasto atslēgvārdu semantikas identificēšana atbilstoši vārdu lietošanas kontekstam.

Trasējamības nozīme reversajā inženierijā ir konstatēta, piem., [43], kur viena no atziņām postulē, ka ir nepieciešams saglabāt informāciju par iegūto vienumu (projektējuma u.c.) sasaisti ar oriģinālajiem avotiem (kodu, datubāzi).

[55] aplūko gadījumu, kad ir nepieciešams restaurēt atbilstību starp sistēmas novērojamo darbību un prasībās specificētajiem darbības scenārijiem (piem., UML lietojumgadījumi). Pieejas izmantošanas sfēra ietver reverso inženieriju. Šajā nolūkā sistēmas darbības novērošanai ir izmantojami testu scenāriji, kurus parasti sagatavo programmas izstrādes ietvaros. Faktiski šajā gadījumā notiek testēšana, kur testi var kalpot kā prasību specifikācijas elementi.

Kopumā var konstatēt, ka aktuāls jautājums ir informācijas analīze apstākļos, kad dokumenti nesatur formālus pierakstus, bet gan ir rakstīti dzīvā valodā. Šādas analīze trasējamības kontekstā netiek plaši praktizēta, un aprakstītie piemēri neizmanto smalkākas metodes par iepriekšminēto atslēgvārdu identificēšanas pieeju. Viens no ideāliem skatījumiem programminženierijā ir virzīties uz to, lai programmatūras prasību specifikācijas būtu veiktas stingri formālā veidā. Tomēr praksē formālās pieejas nav guvušas plašu atsaucību, jo sistēmanalītiķim ir nepieciešama atbilstoša kvalifikācija un spēja pārcelt reālās dzīves jēdzienus formalizētos jēdzienos. Šādu iemeslu dēļ arī tuvākā nākotnē saglabāsies kontrasts starp prasībās minēto dzīvo valodu un izpildāmo kodu – tiešu saistību automatizēti atrast visos gadījumos nebūs iespējams. Risinājumi ir iespējami, piemēram, ja izmanto domēna specifiskas specificēšanas valodas vai vides, ar kuru palīdzību var ģenerēt programmas kodu [126]. Šādā kontekstā veiktie eksperimenti liecina, ka var automātiski ģenerēt arī trasējamības informāciju vēlamajā formā, lai to vēlāk izmantotu atbildēšanai uz jautājumiem par to, kā kods saistās ar prasībām.

3.9. Trasējamības atbalsts ar rīku palīdzību

Īpašs pētījumu virziens ir trasējamības atbalsts ar atbalstošu programmu jeb rīku palīdzību. Ņemot vērā to, ka trasējamības nodrošināšanai ir jāuzkrāj un jāapstrādā liels apjoms informācijas, tīri dabiski veidojas vēlme šo procesu automatizēt. Rīka galvenie uzdevumi būtu iedalāmi šādās kategorijās:

- informācijas savākšana,
- informācijas glabāšana,

- informācijas apstrāde,
- informācijas analīze,
- vispārējs lietotāja atbalsts ar trasējamību saistītos darbos.

Analizēto trasējamībai veltīto publikāciju raksturs liecina, ka ir veikti teorētiski pētījumi un atsevišķi eksperimenti ar trasējamības rīku izstrādi, bet daudzos gadījumos nav informācijas par to izmantošanu reālos programmatūras projektos. Vairākos gadījumos netieši ir secināms, ka rīka izmantošanai varētu būt praktiskas dabas ierobežojumi attiecībā pret informācijas apjomiem un projekta iespējām veltīt cilvēka un tehniskos resursus attiecīgā rīka darbināšanai. Industriāliem projektiem interesē pielietojami rezultāti. Jāņem vērā, ka industriālos projektos ne vienmēr tiek pielietotas tīras programmizstrādes metodikas – ja rīks attiecīgajai metodikai ir stingri piesaistīts, var rasties tā izmantošanas problēmas.

Akadēmiska rakstura trasējamības rīki

Publikācijās norādes uz trasējamības rīkiem ir sastopamas vairākos atšķirīgos kontekstos – ietekmes analīzes veikšana, trasējamības saišu automātiska identificēšana, starp objektiem esošo saišu reģistrēšana, analīze utt.

Ir pieejams izstādes stadijā esoša rīka RARE pamatnostādņu apraksts [32], kas ir rīks objektorientētas programmatūras izstrādei, nodrošinot trasējamības informācijas uzturēšanu starp dokumentiem un prasībām. Risinājums balstās uz autoru līdzdalībā tapušu programmatūras sistēmu izstrādes metodiku SEPA. Lai izmantotu trasējamības nodrošināšanas iespējas, ir nepieciešams pieturēties pie autoru definētās objektorientētās izstrādes metodikas. RARE viena no būtiskākajām iezīmēm saistībā ar trasējamību ir nodrošinātā iespēja izsekot pamatojumam līdz prasībām, kādēļ tiek veidota katra konkrētā klase, ko vāji nodrošina citi objektorientētie rīki. Nav zināms par šī rīka izmantošanas rezultātiem reālos projektos, tomēr jāsecina, ka uzstādītie ierobežojumi padara šī rīka lietošanu iespējamu tikai saistībā ar speciālās metodikas izmantošanu.

Viens no publikācijās minētiem trasējamības rīkiem ir ANALYST. Tas ir raksturojams kā akadēmiska rakstura trasējamības saišu analīzes rīks. Viens no tā pielietojumiem ir ietekmes analīzes veikšana [57]. Rīka pamatā ir datubāze, kurā tiek glabāta informācija par programmatūras artefaktiem un saitēm starp tiem. Ar datubāzi sadarbojas programmas servisa slānis, kurā var nodefinēt to, kāds ir projekta MDD (Model Dependency Descriptor) – trasējamības modelis. Rīka atbalsts ietekmes analīzes veikšanai tika novērtēts ar eksperimenta palīdzību, iesaistot studentu grupu kā personas, kas veic šo uzdevumu ar un bez rīka. Eksperimenti ir veikti gan ar procedurāla stila programmēšanas valodu, gan objektorientētu sistēmas izstrādi. Tālāki eksperimenti ir veikti ar atšķirīgiem MDD [44], tomēr visi tie ir veikti akadēmiskos apstākļos bez informācijas par pielietojumu reālos projektos.

Ietekmes analīzes atbalstam ir veidots arī pusautomātiska pieeja QuaTrace [140]. Tās izveides pamatā ir bijusi ekonomiska rakstura motivācija – samazināt kļūdu izmaiņu apjomu novērtējumos. Pieeja ir realizēta kā integrācija starp prasību pārvaldības rīku RequisitePro un CASE rīku Rhapsody, bet var tikt izmantota arī integrācijai starp citiem rīkiem. Autori apraksta pieeju saišu reģistrēšanai un apstrādei, kā arī piedāvā uzlabojumus projekta darbam (piemēram, definē lomu Prasību pārvaldnieks), tomēr nesniedz informāciju, kā minētie risinājumi ir izmēģināti reālos projektos.

Tādos uzdevumos, kur ir mērķis atrast saites starp programmas kodu un lietošanas scenārijiem eksperimentāli ir izmantota Trace Analyzer pieeja [56], kuras pamatā ir rīks, kas programmas darbības laikā ļauj reģistrēt darbināto kodu. Ja tajā pašā laikā ir iespēja norādīt, kāda modeļa vai specifikācijas daļa/elementi attiecīgajā brīdī tiek darbināti jeb izmantoti, tad rīks izveido saišu tīklu starp programmas kodu un prasībām. Šī metode ir pielietojama gadījumos, kad ir gatava programma, un ir nepieciešams uzzināt, kādas saites reāli eksistē. Metodes izmantošana ir ilustrēta ar eksperimentālu piemēru.

Sistēmas prasību un arhitektūras informācijas pārvaldības rīks TRAM ir orientēts uz praktisku pielietošanu projektos [63], nodrošinot dokumentus ar nepieciešamajām sagatavēm un trasējamību starp dokumentiem. Rīks ir veidots kā komerciāli pieejama rīka DOORS paplašinājums.

[116] – TOOR ir objektorientētā vidē veidots rīks, kas nodrošina iespēju definēt dažāda tipa projektā objektus, starp kuriem veikt trasējamību. Uz definētās informācijas pamata ir iespēja veikt dažāda veida saišu analīzi. Kā tehnisks apgrūtinājums no lietošanas viedokļa ir jāmin tas, ka lietotājam ir jāprot strādāt ar FOOPS (Functional and Object-Oriented Sysytem) valodu, lai veiktu objektu tipu definēšanu un informācijas analīzi. Lai arī tiek demonstrētas trasējamības iespējas uz dažādu piemēru bāzes, nav sniegta informācija par to, kā rīku varētu pielietot reālos projektos. Autori min to, ka rīks ir veidots pētījumiem, lai izpētītu šāda tipa rīka prasības un iespēju to izmantot reālos izstrādes projektos.

PRO-ART rīks [117], [118] ir veidots kā pre-RS trasējamības atbalsta līdzeklis paralēli jeb līdzās strādājoši personu vai grupu darba informācijas saskaņošanai. Tā pamatā ir vienotas informācijas resursu vārdnīcas izmantošana, lai visi dalībnieki izmantotu vienotus jēdzienus informācijas pierakstīšanas un uzturēšanas laikā. Rīks ļauj prasībās izvairīties no nevajadzīgu sinonīmu un daudznozīmību lietošanas, kā arī ļauj identificēt savstarpēji saistītas prasības. Autoru mērķis ir panākt rīka izmantošanu tieši prasību specificēšanas laikā, jo tiek saskatīta iespēja, ka tas var uzlabot prasību kvalitāti, kā arī autori reāli novērtē to, ka pēc prasību specificēšanas projekts nebūs ieinteresēts tās atkārtoti apstrādāt. Ir sniegti rīka lietošanas piemēri.

[130] piedāvā koncepciju trasējamības rīkam, kas varētu funkcionēt par integrējošu mehānismu dažādos citos rīkos tapušiem programmatūras artefaktiem. Risinājumā uzsvars ir likts uz iespēju apvienot nesasaistītus artefaktus, kuriem jau var

eksistēt arī kādas savas saites ar citiem, kā arī autori piedāvā identificēt netiešās saites, tās padarot par tiešajām saitēm. Tiesa, autori pieeju ir ieskicējuši tikai kā koncepciju, un nav zināms, cik veiksmīgi šādu pieeju izdosies vai ir izdevies realizēt.

Komerciālie rīki programmatūras projektiem un trasējamība

Pastāv vairāki iespējamie skatījumi, kā iedalīt programminženierijā izmantojamos rīkus. Programmatūras inženieru profesionālā asociācija INCOSE (*International Council on Systems Engineering*) [73] piedāvā programmatūras sistēmu inženieru interešu lokā esošo rīku taksonomiju. Rīku veidi ir sagrupēti hierarhiskā formā:

1. Pārvaldība
   - 1.1. Konfigurācijas pārvaldība
   - 1.2. Darba plūsmas pārvaldība
   - 1.3. Riska pārvaldība
   - 1.4. Izmaksu novērtēšana un izsekošana
     - 1.4.1. Izmaksu novērtēšana
     - 1.4.2. Izmaksu izsekošana
   - 1.5. Problēmu izsekošana
2. Inženierija
   - 2.1. Sistēmas projektēšana
     - 2.1.1. Sistēmas modelēšana
       - 2.1.1.1. Strukturālā modelēšana
       - 2.1.1.2. Uzvedības modelēšana
         - 2.1.1.2.1. Statiska uzvedība
         - 2.1.1.2.2. Dinamiska uzvedība
       - 2.1.1.3. Cilvēka-datora saskarnes prototipēšana
     - 2.1.2. Projektējuma atbalsts
       - 2.1.2.1. Simulācija
       - 2.1.2.2. Skaitliskā analīze
       - 2.1.2.3. Domēna specifiskie rīki
       - 2.1.2.4. Efektivitātes mērījumi
   - 2.2. Prasību inženierijas rīki
     - 2.2.1. Prasību pārvaldība
       - 2.2.1.1. Prasību klasifikācija
       - 2.2.1.2. Prasību notveršana un reģistrēšana
         - 2.2.1.2.1. Tekstuāla prasību notveršana
         - 2.2.1.2.2. Prasību izdibināšana
       - 2.2.1.3. Prasību trasējamība
     - 2.2.2. Prasību ģenerēšana
   - 2.3. Projektējuma validācijas rīki
     - 2.3.1. Pavedienu analīze
     - 2.3.2. Testu validācijas plānošana
     - 2.3.3. Scenāriju validācija
     - 2.3.4. Atbilstības prasībām pārbaude
       - 2.3.4.1. Mērīšana
       - 2.3.4.2. Veiktspējas analīze
   - 2.4. Specializētie inženierijas rīki

3. Informācijas kopīgošanas rīki
   3.1. Komunikācija
      3.1.1. Komunikācija starp personām
      3.1.2. Tīkla informācijas iegūšana
   3.2. Datu analīze
      3.2.1. Elektroniskās tabulas
      3.2.2. Datu reducēšana
      3.2.3. Datu vizualizācija
   3.3. Elektroniskā publicēšana
      3.3.1. Automātiskā dokumentu ģenerēšana
      3.3.2. Tehniskās publicēšanas sistēmas
      3.3.3. Tehniskās ilustrēšanas
      3.3.4. Failu formātu pārveidotāji
   3.4. Elektroniskā skatīšanās
      3.4.1. Datubāzes satura skatītāji
      3.4.2. Formatētu dokumentu skatītāji
   3.5. Rīku integrācija
4. Infrastruktūras atbalsta rīki
   4.1. Sistēmas administrēšana
   4.2. Tīkla atbalsts
   4.3. Produkta datu pārvalde

Specializētos projektos var eksistēt vēl papildus arī citi rīku veidi, kā arī ir iespējama mazliet atšķirīga pieeja taksonomijai. Viens un tas pats rīks var piederēt vairākām kategorijām gadījumā, ja tas, piemēram, ir veidots kā visaptverošs un universāls CASE tipa rīks. Autors uzskata, ka minētais dalījums sniedz labu pārskatu par rīku veidiem, bet atsevišķos punktos ir neprecīzs. Piemēram, testēšanas rīki (kategorija „Atbilstības prasībām pārbaude") nav raksturoti to iespējamos veidos, jo pie tiem var piederēt rīki, ar ko var veikt datu sagatavošanu, koda statisko analīzi, lietotāja saskarnes testus, vai arī uzturēt testu kopu. Saskaņā ar INCOSE klasifikāciju eksistē prasību trasējamības rīki, kas ir daļa no prasību pārvaldības rīku grupas. Trasējamības rīki plašākā nozīmē vairāk ir akadēmisku pētījumu objekts. Tajā pašā laikā vairākos „ne-trasējamības" rīkos ir novērojamas iestrādātas trasējamības nodrošināšanas iespējas – galvenokārt, prasību pārvaldības rīkos.

Autors no vairākām IT organizācijām ir noskaidrojis, ka ne vienā vien projekta gadījumā ir bijuši pieejami kādi no prasību pārvaldības rīkiem, piemēram, IBM Rational RequisitePro [66], tomēr tie netika pilnībā izmantoti, vai arī to izmantošana tika pārtraukta pēc prasību analīzes veikšanas. Konkrētie iemesli varētu būt dažādi, bet viens no galvenajiem ir tas, ka attiecīgie rīki projektam liek pieturēties pie konkrēta procesa vai rezultātu formāta, kas, savukārt, neatbilst citu projektu darbu rezultātiem vai nepieciešamajām ievadēm. Kā liecina prakse, reāli projektā ir iespējams strādāt ar dažādiem rīkiem – liela nozīme ir projekta darbinieku prasmei tos atbilstoši izmantot. Tajā pašā laikā ir novērojami rīki, kas ir izmantojami vairākos projekta procesos, vai arī nav viennozīmīgi raksturojami kā piederīgi tieši vienai no iepriekšminētajām kategorijām. Piemēram, sistēma problēmziņojumu reģistrēšanai

var tikt attiecināta gan uz testēšanu, gan projekta pārvaldību, gan arī uz izmaiņu pārvaldību.

Kopā ar INCOSE rīku taksonomiju ir izveidots reģistrs ar konkrētiem rīkiem. Pie Prasību pārvaldības un trasējamības rīku kategorijas ir norādīti 22 rīki. Par 9 no tiem 2004.gadā vairs aktuāla papildus informācija nav atrodama. Divi no atlikušajiem (icCONCEPT un RTM) ir apvienoti to uzturošo kompāniju saplūšanas rezultātā. No 12 aktīvajiem rīkiem trīs ir viena rīka DOORS [136] paveidi, bet no pārējiem 9 trīs ir saistīti ar konkrētu prasību pārvaldības pieeju (CORE, CRADLE/REQ un IRqA).

Lai arī autoram neizdevās iegūt citu rīku pilnās versijas, tomēr bija iespēja iepazīties ar rīku aprakstiem un rīku demonstrācijām. Vieni no populārākajiem rīkiem ir DOORS un RequisitePro. Par DOORS [136] ir saņemtas atsauksmes kā labu rīku prasību pārvaldīšanai, bet tas nav parocīgs neformālas informācijas uzturēšanai. Šo rīku izmanto daudzos ar IT saistītos uzņēmumos (ražotājs, atsaucoties uz The Standish Group pētījumu, apgalvo, ka tā tirgus daļa prasību pārvaldības rīku vidū ir 32%), un vairāki autoru iegūtie trasējamības praktiskie risinājumi (piemēram, [21]), ir balstīti tieši uz DOORS iespējām. Neskatoties uz šī rīka it kā lielo popularitāti, autors nav guvis apstiprinājumu tam, ka šo rīku izmantotu kāda IT kompānija arī Latvijā, bet ir novērojis šī rīka izmantošanu lielā telekomunikāciju kompānijā ārpus Latvijas. Minētais rīks ir pateicīgs prasību specifikāciju gatavošanai, jo ar tā palīdzību var ne tikai strukturēt un sasaistīt prasības, bet arī ģenerēt dažāda tipa dokumentus un pārskatus. Nav šaubu, ka šo rīku var izmantot prasību uzkrāšanai un trasējamībai. Ņemot vērā to, ka šī rīka kontekstā tiek minēta tikai trasējamība starp dažāda tipa prasībām un tikai atsevišķos gadījumos ar piesaistītajiem vienumiem, pastāv šaubas par to, vai rīkā var pilnvērtīgi uzkrāt informāciju par citiem projekta vienumiem.

Cits populārs rīks, RequisitePro [66], ir labi izmantojams, kad projektā tiek pielietots Rational Unified Process [124]. Tas no projekta prasa pieturēšanos noteiktam darba procesam un izmantot arī citus specifiskus rīkus, kas varētu nesaskanēt ar projekta iecerēto pieeju. RequisitePro ir vairāku Latvijas IT kompāniju rīcībā, tomēr nav informācijas, ka tas būtu pilnvērtīgi izmantots kādā konkrētā projektā. Minētais produkts tiek attīstīts vairāku gadu garumā, un labi atbalsta projektam nepieciešamo trasējamību. Īpaša iespēja rīkā ir uzglabāt ne tikai prasības, bet arī testus, kas to var padarīt noderīgu testēšanas procesa dalībniekiem. RequisitePro rīku var izmantot ne tikai formālo prasību uzkrāšanai, bet tajā var glabāt arī projektā veicamo uzdevumu sarakstu [135]. Pieejas būtība ir tajā apstāklī, ka tos var uztvert kā prasības tipu, un katram no tiem var atzīmēt statusu par izpildi. Rezultātā ieinteresētās personas var sekot līdzi izpildei un viegli veidot statusa pārskatus. Šis ir viens no risinājumiem, kā izvairīties no papildus vides izmantošanas gadījumos, ja ir jāuztur tikai neliels apjoms papildus informācijas.

Ir atrodamas atsauksmes par prasību pārvaldības un trasējamības rīku RTM [79], [129], kuru ir iespējams integrēt ar versiju pārvaldības sistēmu, iegūstot labu vertikālo un horizontālo trasējamību. Vairāki rīki sola trasējamības atbalstu, bet

daudzos gadījumos tā ir tikai pievienota īpašība prasību pārvaldības rīkam, un šī pievienotā īpašība ne vienmēr ir veiksmīgi pielietojama daudzveidīgu projektu gadījumā (t.i. der konkrēti organizētiem projektiem). Jaunākajiem rīkiem veidojas kopīgas iezīmes. Komerciālo rīku pēdējā laika versijas ļauj konfigurēt trasējamības modeļus. Katrā produktā tas tiek nosaukts citādi, kā arī eksistē noteikti ierobežojumi trasējamības modeļa aptvērumam.

Būtisks aspekts rīka izmantošanā ir lietotāji, kas strādā ar minēto trasējamības informāciju. B.Ramešs un M.Jarke [123] iedala lietotāju divās kategorijās: augstākās un zemākās kategorijas lietotājos. Zemākās kategorijas lietotājiem nav liela trasējamības pieredze, un viņi to saskata kā piespiedu pasākumu, kuru vēlas projekta īpašnieki, lai būtu atbilstība standartiem. Augstākās kategorijas lietotāji ir tie, kas trasējamībā saskata iespēju uzlabot zināšanas par projektu un sasniegt labāku rezultātu. Pārsvarā augstākās kategorijas lietotāji veidojoties pēc piecu gadu pieredzes ar trasējamības saistītiem jautājumiem.

Viena no eksistējošām problēmām ir tā, ka neautomatizētā variantā par trasējamību pārsvarā domāja viens cilvēks un tikai tiktāl, cik tas formāli būtu nepieciešams. Ja izmanto rīku, tad negatīva situācija var veidoties ar to, ka arī šajā gadījumā no visa projekta dalībniekiem ar rīku strādā tikai viens. Komunikācija fokusējas uz vienu personu - šāda komunikācija būs vai nu nepilnīga, vai arī būs ar pārāk intensīvu informācijas plūsmu. Kā labu var vērtēt tādu situāciju, kad trasējamības informācijas uzturēšanā ir iesaistīti visi darbinieki (lai arī pat katrs individuāli to var neapzināties).

3.10. Trasējamības kvalitāte

Kā jau bija minēts iepriekš, ir pieejas, kad trasējamība tiek uzskatīta par vienu no projekta un produkta kvalitātes rādītājiem. Publikācijās par trasējamību netiek definēti kritēriji, kā vērtēt pašas trasējamības kvalitāti. Tomēr tiek norādītas īpašību klases, kuras var izmantot par pamatu kritēriju definēšanai. Ir atrodamas norādes par izmaiņu ietekmes analīzes kvalitātes novērtējumu, kas tiek iegūts no diviem rādītājiem – ietekmes pareizības (vai identificētie vienumi tiešām tiek ietekmēti) un ietekmes pilnīguma (vai ir identificēti visi ietekmētie vienumi) [138]. Trasējamībai ir aplūkojami plašāka mēroga kritēriji. Tās kvalitāte ir atkarīga no projektā iedibinātās pieejas, no tā, cik cieši projekts ievēro savas procedūras. Autors piedāvā trasējamības kvalitāti vērtēt pēc pieciem kritērijiem, kas aprakstīti 6.tabulā. Vērtējuma skalā katram kritērijam ir norādīts punktu skaits, ko tas piešķir trasējamības kvalitātes vērtējumam. Trasējamības vērtējums var būt robežās no 0 līdz 13.

6.tabula. trasējamības kvalitātes kritēriji.

Kritērijs	Izklāsts	Vērtējumu skala (punkti)
Vienumu aptvērums	Raksturojums tam, kādus vienumus aptver	Nekāds (0) Niecīgs (1) – tikai atsevišķi vienumi,

	trasējamība, cik daudz un cik detalizēti.	kas veido stipri fragmentāru skatu Vidējs (2) Plašs (3) – ietverti praktiski visi projekta vienumi
Iesaistītie darbinieki	Raksturojums tam, kuri projekta darbinieki strādā ar trasējamības informāciju.	Neviens (0) Viens (1) – ar trasējamību nodarbojas viena persona (no lielas projekta grupas) Daži (2) Visi (3)
Uzglabāšanas vieta un metode	Raksturojums tam, kur tiek uzglabāta trasējamības informācija un vai tiek izmantots rīks vai metodes	Nav (0) Teksta dokumenti (1) Dokumenti un rīks (2) Rīks (3)
Trasējamības mērķis	Trasējamības informācijas uzkrāšanas mērķa raksturojums	Nav (0) Izpildīt formālās prasības (1) Atvieglot darbu (2)
Informācijas reģistrēšanas regularitāte	Cik bieži informācija tiek reģistrēta attiecībā pret projekta notikumiem	Netiek reģistrēts (0) Ar laika intervālu (1) Vienlaicīgi ar vienuma izmaiņām (2)

Ja konkrētam projektam analizē trasējamības kvalitāti, ir jāņem vērā, ka eksistē divi trasējamības kvalitātes aspekti – iecerētā (plānotā) un reālā. Par iecerēto var spriest pēc trasējamības modeļa un papildus informācijas par plānot saturu. Bet reālā objektīvi raksturo saistību. Lai to noskaidrotu, ir jāveic detalizēti pētījumi par projekta darbu struktūru, ikdienas praksi utt. Varētu sastapt situāciju, kad plānotā trasējamība atšķiras no reāli lietojamās. Sagaidāms, ka pēc noteikta laika plāni tiek koriģēti atbilstoši praksei.

3.11. Nodaļas secinājumi

Trasējamība ir sastopama ne tikai programminženierijā, bet tai ir ilgstošas tradīcijas arī fizisko produktu ražošanā un apstrādē. Tā tiek izmantota ražošanas procesu uzlabošanai un patērētājiem kalpo kā garantija par solītā satura un īpašību esamību produktā. Tā ir cēlusies no cilvēka dabiskās intereses izzināt lieto patieso dabu, cēloņus un sekas.

Trasējamības var iezīmēt divas būtiskākās misijas:

- Padarīt kvalitatīvāku programmatūras izstrādes (vai testēšanas) procesu;
- Nodrošināties ar risinājumu problēmu notikšanas gadījumā.

Jo projekta vai procesa lietotāji labāk apzinās misiju, jo vairāk iegūtie rezultāti ļaus nodrošināt trasējamību. Būtisks atskaites punkts trasējamībai ir prasības. Izstrādātājiem ir svarīgs viss, kas ir atvasināms no prasībām, bet pasūtītājam svarīgi ir

būt pārliecinātam, ka viss nepieciešamais ir nonācis līdz minētajām prasībām (pēdējā minētajā gadījumā situācija praksē mēdz būt arī tāda, ka pasūtītājs par šo atbilstību tomēr neinteresējas, bet pievērš uzmanību tikai gala rezultātam).

Starp trasējamības jomā pētāmajiem jautājumiem ir trasējamības modeļu apzināšana konkrētās situācijās, trasējamības informācijas iegūšana, trasējamības īpašības specifiskos programmizstrādes apstākļos un tehniskais atbalsts informācijas apstrādei. Šajā nodaļā tika definēti kritēriji trasējamības kvalitātes raksturošanai, kurus var trasējamības salīdzināšanai projektos un procesos.

Gan publikācijās iezīmētās problēmas, gan autora veiktie novērojumu programmatūras projektos liecina par problēmām ar teorētiski iecerētās trasējamības realizēšanu dzīvos projektos – viens no būtiskākajiem iemesliem varētu būt atbilstošu tehnisko risinājumu trūkums. Lai arī ir sastopami rīki, kas tiešā veidā ir orientēti uz trasējamības nodrošināšanu, un ir tādi, kas to satur kā papildus īpašību, joprojām reālam programmatūras projektam ir problēmas atrast sev noderīgu rīku, kas atbalstītu projekta prasības pēc trasējamības. Šādā kontekstā ir iespēja veikt pētījumu ar jaunu eksperimentālu trasējamības rīku – vai tāds spēj projektiem sniegt atbalstu? Ja šāda rīka izstrāde tiek veikta, tam jābūt orientētam uz tipisko trasējamības problēmu risināšanu.

4. Testēšana un trasējamība - problēmas nostādne

4.1. Nodaļas mērķi

Iepriekšējās nodaļas raksturoja testēšanas un trasējamības problemātiku. Šajā kontekstā viens no jautājumiem, kas parādās kopā ar praktiskiem testēšanas darbiem ir par to, kā īsti veidojas testi, kā tos labāk izpildīt un ko darīt, ja projektā mainās prasības un testējamā programma. Noderīgs rezultāts būtu rast atbildi tam, vai trasējamība kā projekta vai procesa īpašība iespaido testēšanas procesu.

Šajā nodaļā autors izvirza apgalvojumu, kas darba tālākajās nodaļās tiek pamatots. Pamatojums balstās uz definīciju analīzes, programminženierijas tradicionālo pieeju analīzes, esošās situācijas novērtējuma programmatūras projektos, eksperimentāla rīka izstrādes un tā ieviešanas rezultātu novērtējuma.

4.2. Novērotās problēmas

Praktiskos testēšanas darbos ir novērojamas dažāda rakstura problēmas. Tās var radīt nelielus traucējumus darbā, prasīt papildus resursus uzdevuma izpildei vai neļaut sasniegt plānotos rezultātus. Tās var attiecināt uz testēšanas metodiku, plānošanu, automatizāciju, datu ģenerēšanu u.tml. Testēšanas procesa norisei ir šādas būtiskākās problēmas:

Te1. Tehniski sarežģīti ir rast atbildes uz jautājumiem par prasībām atbilstošo testu izpildes rezultātiem – zems informācijas pieejamības līmenis, zems pilnīguma līmenis.

Te2. Testēšanas rezultātu apkopojums bez tehniskiem atbalsta līdzekļiem ir laikietilpīgs process – zems operativitātes līmenis.

Te3. Problēmu nesavlaicīga reģistrēšana un nogādāšana pie izstrādātājiem izraisa kavēšanos – zems operativitātes līmenis.

Trasējamības nodrošināšanā projekta vai projekta procesa ietvaros ir novērojamas šādas būtiskākās problēmas:

Tr1. Pēc noklusēšanas pieejamās biroja programmatūras izmantošana nenodrošina efektīvu trasējamības informācijas uzturēšanu.

Tr2. Līdzšinējā organizāciju pieredze trasējamības informācijas uzturēšanā ir bijusi negatīva, tādēļ projekti no šīm organizācijām nesaņem ieteikumus par trasējamības nodrošināšanu.

Tr3. Pat ja tiek sagatavota informācija par projekta vienumu saitēm, tā visbiežāk netiek atjaunota līdz sasniedz stāvokli, kad to vairs nav vērts izmantot.

Tr4. Eksistējošie trasējamības rīki ir veidoti akadēmiskiem nolūkiem, un nav sastopamas norādes par to veiksmīgu izmantošanu industriālos projektos – projektiem ir nepieciešamība pēc trasējamību atbalstošiem rīkiem, kas neuzspiež konkrētu izstrādes vai testēšanas metodiku.

4.3. Apgalvojums

Autors uzskata, ka iepriekšminētās testēšanas un trasējamības problēmas daļēji ir saistītas, un tām ir iespējams kopīgs risinājums. Analizējot testēšanas praktiskās pielietošanas problēmas projektos, autors ir konstatējis testēšanas ciešu saistību ar citiem programmatūras projekta procesiem. Viens no saistošajiem elementiem ir trasējamība. Klasiskajās testēšanas problemātikai veltītajās grāmatās un konferenču materiālos trasējamība netiek minēta kā testēšanas sastāvdaļa vai priekšnosacījums. Tā tiek minēta vispārējo projekta kvalitātes principu kontekstā, tomēr ir pamats domāt, ka trasējamība ir viena no neatņemamām testēšanas sastāvdaļām.

Apgalvojums sastāv no trijām daļām:

Testēšana un trasējamība saistās šādi:

a) testēšanas process ir atkarīgs no trasējamības kvalitātes,

b) trasējamības kvalitāti var uzlabot ar trasējamību atbalstoša rīka palīdzību,

c) ar trasējamību atbalstošu rīku testēšanas process ir kvalitatīvāks nekā bez tā.

Minētais apgalvojums tiek pamatots šī darba tālākajās nodaļās.

Apgalvojuma pamatošanas pieeja

Apgalvojuma (a) daļa tiek pamatota ar nākošajā apakšnodaļā sniegto demonstrējumu par testēšanas atkarību no trasējamības. Apgalvojuma (b) daļa tiks pamatota pēc trasējamību atbalstoša rīka pamatnostādņu izklāsta, tā ieviešanas gaitas un ieviešanas rezultātu analīzes. Apgalvojuma (c) daļa tiek pamatota ar trasējamību atbalstoša rīka izmantošanas pieredzi testēšanas procesa kontekstā. Testēšanas procesa novērtējumam tiek izmantoti šī procesa kvalitātes kritēriji, kas norādīti 2.7.nodaļā.

4.4. Trasējamība kā testēšanas priekšnosacījums

Ja analizējam testēšanas definīcijas, tad var secināt, ka visos gadījumos ir nepieciešams testējamais objekts – programmatūra un:

- jāspēj to salīdzināt ar prasībām [89], [33]
- jāspēj to salīdzināt ar plānoto statusu [85], [33], [84], [67]
- jāspēj to salīdzināt lietotāja vēlmēm, lietošanas scenārijiem [103], [39]
- jāspēj to salīdzināt ar kvalitātes kritērijiem [49]
- jāspēj to salīdzināt ar testu sagaidāmajiem rezultātiem [81]
- jāspēj to atšķirt kļūdu no pareizas darbības [83], [84], [51]
- saskaņā ar specificētu procedūru jāspēj novērtēt īpašības [74].

Testēšana prasa veikt darbu, lai izzinātu, kādas ir programmas (sistēmas) īpašības, bet novērtējuma veikšanai ir nepieciešams, lai mēs zinātu, kādas ir saites

starp aplūkojamo programmas funkcionalitāti, prasībām, projektējumu un kodu (atkarībā no konkrētā testēšanas veida). Tātad sistemātiska testēšana balstās uz to, ka testētājs spēs izsekot attiecīgajām saitēm. Lai tas notiktu, ir jāpastāv trasējamībai starp pārbaudāmo objektu (programmu) un kādu dokumentu, kas nosaka sagaidāmo rezultātu. Pēdējais minētais var būt jebkura tipa prasību specifikācija, projektējuma dokuments, lietotāja rokasgrāmata, tests vai standarts. Jo trasējamība būs labāk nodrošināta, jo precīzākus, pilnīgākus un produkta īpašībām atbilstošākus testus būs iespējams sagatavot. Ja trasējamības nav vispār (t.i. testētājs par to nav informēts, un nav pieejamu līdzekļu to noskaidrot), tad testēšana notiks atbilstoši paša testētāja priekšstatiem par programmas darbību. Ja testētājs darba uzdevumu un programmas lietošanas apgabalu, tad pastāv maza varbūtība, ka minētais subjektīvais priekšstats sakritīs ar programmas prasībām, un testēšanai nebūs jēgas, jo tā nenodrošinās atbilstības pārbaudi sākotnējām prasībām. Tas nozīmē, ka bez trasējamības joprojām var pārbaudīt programmu, bet sistemātiska testēšana nav iespējama. Netiek apmierināti arī kvalitatīvas testēšanas kritēriji – nav nodrošināts pilnīgums. Dažkārt katru no projekta etapiem/procesiem veic citi cilvēki, un katrs operē ar saviem jēdzieniem – projektā netiek nodrošināta kopīga komunikācija, vienota vārdnīca, un katrs process kļūst informatīvi izolēts, paaugstinot kopējo risku tam, ka netiks sasniegti projekta mērķi.

No definīcijām izriet arī tas, ka var nebūt formālu prasību, bet testētājam ir atbilstošās iespējas atrast kļūdas (t.i. spēja atšķirt korektu no nekorekta rezultāta). Tas nozīmē, ka atsevišķi neatrunātā formā testētājs ir informēts par programmas iecerēto funkcionalitāti. Nedokumentētas lietas veidojas apstākļos, kad visu dara vieni un tie paši cilvēki, kas ļoti labi pārzina pilnīgi visas nianses. Šādā situācijā netiek apmierināts kvalitatīva testēšanas procesa kritērijs - informācijas pieejamība par testēšanā iesaistītajiem vienumiem, kā arī neeksistē procesam raksturīgās ievades.

Testēšanas atkarība no trasējamības ir viens no centrālajiem šī darba mērķiem. Šo abu programmatūras jēdzienu saistību autors ir novērojis arī vairākos testēšanas nozares apskates kontekstos [20], [18], lai arī uz to brīdi vēl tas bija nevis formāla pamatojuma formā, bet tikai kā notiekošo pētījumu provizorisko rezultātu izklāsts.

Trasējamības kvalitātes ietekmes uz testēšanu novērtējums:

- Vienumu aptvērums – aptvērumam jābūt pietiekamam, lai būtu ietverti visi testēšanas procesā iesaistītie projekta vienumi, konkrēti, saites starp dažāda veida testiem un prasībām, kas ļauj analizēt prasību pārklājumu.
- Iesaistītie darbinieki – informācijas uzturēšanā jāpiedalās vismaz tiem testēšanas procesa dalībniekiem, kuri gatavo testus.
- Uzglabāšanas vieta un metode – jo vieglāk būs nodrošināt pārskatāmību un informācijas apmaiņu par sagatavotajiem testiem un to saistību ar prasībām, jo labāk būs atbalstīts testēšanas process.

- Trasējamības mērķis – iegūt pārliecību par testu atbilstību prasībām un iegūt novērtējumu par prasību implementācijas korektumu.
- Informācijas reģistrēšanas regularitāte – jo savlaicīgāk tiks atjaunota informācija par testiem, to saistību ar prasībām, to izpildes rezultātiem un izraisītajām problēmām, jo labāk būs iespējams pārraudzīt testēšanas stāvokli.

No iepriekšminētā seko, ka testēšanas procesa kvalitāte (informācijas pieejamība, operativitāte un pilnīgums) cieši saistās ar trasējamības kvalitāti. Ja testēšanas procesam netiks nodrošināta trasējamība pret prasībām, tad testēšana nav veicama kvalitatīvi. Jo labāk ir apmierināti trasējamības kvalitātes kritēriji, jo kvalitatīvāks ir testēšanas process. Iepriekšminētais kalpo kā apgalvojuma (a) daļas pamatojums.

5. Industriālie programmatūras projekti un trasējamība

5.1. Nodaļas mērķi

Nodaļa aplūko teorētiski prasīto un programmatūras projektos praksē sastopamo trasējamību. Teorētiskās prasības ir parādītas standartu un projektu labās prakses kontekstā. Īpaša uzmanība ir pievērsta testēšanas procesa prasībām un tajā esošās trasējamības īpašībām. Veiktā analīze ļauj veikt secinājumus par testēšanas procesa atkarību no trasējamības kvalitātes.

Lai novērtētu projektu reālo situāciju ar trasējamību, autors 2004.gada ietvaros veica aptauju Latvijas IT uzņēmumos, kur projektu pārstāvji informēja autoru par to, kā ir noritējis projektu darbs un kā savā starpā tika sasaistīta projektā esošā informācija. Mērķis bija konstatēt, kādas metodes projekti izmanto trasējamības nodrošināšanai, kā arī saprast, cik liela daļa no projekta informācijas tiešām tiek pakļauta trasējamībai. Iegūtie rezultāti ļauj veikt secinājumus par būtiskākajām trasējamības problēmām, tipiskākajiem projekta vienumiem un saitēm, kā arī par tradicionālākajiem risinājumiem.

5.2. Programmizstrādes projektu prasības

Trasējamība nav tikai atsevišķu projektu labā griba, bet tam pamatā ir vairāku standartu prasības, kuras definē vēlamo trasējamības līmeni, bet projekti var izvēlēties arī vienkāršāku pieeju, ja tam ir atbilstošs pamatojums.

Standarta J-STD-016 sadaļas 4.2 "Programmatūras izstrādes vispārējās prasības" punktā 4.2.3. "Trasējamība" ir teikts: "Izstrādātājam ir jāveic trasējamība starp prasību līmeņiem, starp prasībām un projektējumu, starp projektējumu un programmatūru, kas to implementē, starp prasībām un kvalifikācijas testu informāciju un starp aparatūras resursu lietošanas prasībām un reģistrēto aparatūras resursu lietošanu." Dokumenti, no kuriem ņemt trasējamības vienumus ir minēti šādi: Sistēmas/apakšsistēmas specifikācija (SSS), Saskarnes prasību specifikācija (IRS), Programmatūras prasību specifikācija (SRS), Sistēmas/apakšsistēmas projektējuma apraksts (SSDD), Saskarnes projektējuma apraksts (IDD), Datubāzes projektējuma apraksts (DDD), Programmatūras projektējuma apraksts (SDD), Programmatūras testēšanas plāns (STP), Programmatūras testu apraksts (STD), Programmatūras produkta specifikācija (SPS). Standarts katram šim dokumentu tipam prasa atsevišķi aprakstīt nepieciešamo trasējamības informāciju. Analoģiski, vienīgi vispārinātā formā trasējamību prasa arī programmatūras izstrādes procesus aprakstošais standarts ISO/IEC 12207.

Arī ISO 9001:2000 standarta interpretācija programmatūras izstrādei TickIT Guide Issue 5.0 virknē prasību iekļauj trasējamības nepieciešamību. Piemēram, E daļas punktā 7.5.3. "Identifikācija un trasējamība" ir izvirzītas vispārējās prasības pret

trasējamību programmatūras procesos - piegādē, kodēšanā, integrācijā, testēšanā, uzturēšanā, dokumentēšanā u.c.

No vienas puses ir redzams, ka prasība pēc trasējamības ir vispārīgā formā, bet, piemēram, J-STD-016 dokumentu sagatavēs ir nodaļas ar nosaukumu "Prasību trasējamība". Tieši šis apstāklis daudzos gadījumos izsauca to, ka standarta formālās prasības ievērošanas nolūkos tur tiek ievietota kāda trasējamības tabula, bet to reāli neviens neizmanto. Īpaši lielas problēmas izceļas izmaiņu gadījumā - šāda tipa tabula kļūst neaktuāla, un no pirmo izmaiņu brīža to var uzskatīt par nederīgu.

Trasējamība organizācijas kvalitātes aktivitāšu kontekstā

Autors kopā ar līdzautoriem ir veicis novērtējumu tam, kādi ir būtiskākie kvalitātes nodrošināšanas darbi [1]. Novērojumi tika balstīti uz divu IT kompāniju pieredzes. Organizācijas līmenī viens no kvalitātes uzturēšanas un uzlabošanas mehānismiem ir procesa uzlabošanas pasākumu veikšana, tajā skaitā ietverot šādus darbus:

1. Kvalitātes pārvaldības sistēmas izveide organizācijas līmenī;
2. Programmatūras izstrādes procesa mērījumu veikšana projekta un organizācijas līmeņos;
3. Trasējamības jautājumu akcentēšana un tās atbalsts ar rīku palīdzību.

Programmatūras procesa uzlabošanas aktivitātes organizācijas līmenī iespaido visas organizācijas funkcijas. Viena no īpašībām šāda veida aktivitātēm ir tāda, ka uzlabošanas procesam ir jābūt pastāvīgam. Uzlabojumus var strukturēt pa nelieliem soļiem, kur viens no būtiskākajiem ir kvalitātes sistēmas izveides pasākums. IT uzņēmumos gūtā pieredze liecina, ka viens no optimāliem variantiem ir kvalitātes sistēma ieviešana kopā ar atbilstošu mērījumu programmu. Lai kvalitātes sistēmā ieviestu procesus, ir jāsaprot, kas no notiekošā ir tiešām atsevišķs process, bet kas būtu ierindojams tikai kā lielāka procesa sastāvdaļa. Kopā ar kvalitātes sistēmu tiek izstrādāta arī dokumentu pārvaldības sistēma. Lai to veiktu, ir jābūt skaidri izstrādātam priekšstatam, kādi dokumentu tipi tiks veidoti un kādas būs to funkcijas. Ir pieredzētas situācijas, ka organizācijai ir bijis liels dokumentu skaits, bet reālais darbs ar dokumentiem darbs dažu gadu laikā ļauj izanalizēt, kas patiešām ir nepieciešams, un tā rezultātā ir iespējams samazināt dokumentu apjomu, noņemt lieku slodzi no darbiniekiem. Šāda pieeja ļauj no darbinieku puses radīt pozitīvu attieksmi pret kvalitātes sistēmu. Kopumā ņemot, dokumentu pārvaldības sistēmas galvenais uzdevums ir ļaut vieglāk sagatavot projektā izmantojamos dokumentus, darīt tos pieejamus un saskaņotus, uzturēt vienotu dokumentu bibliotēku ar versiju kontroli.

Mērījumu process parasti ir orientēts uz projektu vai organizāciju. Abos gadījumos mērījumi tiek iegūti no notiekošajiem procesiem, un būtisks aspekts mērījumu veikšanai ir saprast mērķi, kam tiks izmantoti iegūtie rezultāti. Ja process ir labi organizēts, ir iespēja tam nomērīt daudz dažādus parametrus, bet reāli procesa

uzlabošanai tiek izmantoti tikai atsevišķi no tiem. Jāņem vērā, ka mērījumu veikšana arī prasa resursus, tādēļ ir jāveic tikai tie mērījumi, kas var uzlabot projekta procesu caurspīdīgumu un ļaut samazināt programmatūras izstrādes riskus. Ir veikti pētījumi par to, kādi riski no projekta viedokļa tiek uzskatīti par būtiskākajiem. Visbiežāk minētais ir risks, ka var rasties problēmas ar programmatūras prasībām. IT kompānijā veiktā mērījumu programma tika veidota ar mērķi novērtēt izstrādes procesu, lai iegūtu gatavību risku novēršanai. Būtiska ir regulāra mērījumu informācijas analīze un salīdzināšana ar labās prakses rezultātiem.

Iepriekšminētā kontekstā arī trasējamība konkrētajos uzņēmumos tiek identificēta kā viens no organizācijā notiekošo procesu uzlabošanas mehānismiem. Trasējamībai tiek izvirzīta pilnības prasība: starp jebkuriem diviem projekta vienumiem ir jābūt tiešai vai pastarpinātai saitei. Reālos projektos šī prasība ne vienmēr ir izpildāma, bet tā var kalpot kā mērķis, kuru projektam jācenšas sasniegt.

Trasējamības dzīves cikls

Projekta ietvaros trasējamība kā īpašība attīstās kopā ar notiekošajiem procesiem. Lai arī autors nav ne redzējis personīgi, ne arī sastapis norādes par to, ka kādos projektos tiktu īpaši aplūkots trasējamības cikls, tāds reāli eksistē. Tam ir šādi etapi:

1. Projekta vienumu tipu definēšana
2. Projekta vienumu attiecību definēšana
3. Vienumu izveide jeb apzināšana
4. Saišu izveide jeb iegūšana
5. Vienumu kopas analīze viena vienuma tipa ietvaros
6. Saistīto vienumu analīze
7. Vienumu un saišu modifikācijas

No jebkura etapa var tikt veikta pāreja uz jebkuru iepriekšējo etapu, iteratīvi paplašinot trasētās informācijas kopu. Šādi etapi seko no programmatūras projekta norises specifikas. Sākotnēji projekts izvēlas izstrādes metodiku, kas nosaka izmantojamos dokumentācijas veidus, uzkrājamo informāciju un informācijas apmaiņas formu. Kad projektā tiek uzsākti darbi, parādās saturs atbilstoši iepriekš noteiktajai formai, kā arī daļa vienumu veidojas, balstoties uz iepriekš projektā eksistējošiem vienumiem. Pie viena vai vairāku vienumu eksistences ir iespējama to satura un saišu analīze. Trasējamībai nav definēts gala produkts – tā eksistē kopā ar atbilstošajiem procesiem. Ikviena izmaiņa jebkurā vienumā var izraisīt izmaiņas trasējamības informācijā. Ja projekta trasējamības informāciju ir plānots atbalstīt ar rīka palīdzību, tad tam ir jābūt pieskaņotam trasējamības dzīves cikla etapiem.

5.3. Testēšanas grupas prasības

No organizatoriskā viedokļa testēšanas grupa projektā tipiski veidojas nevis vienkārši no ierindas testētājiem, bet šai grupai ir vadītājs. Atkarībā no izmēra tas var būt viens no testētājiem, vai arī atsevišķa persona. Tiesa, nav novērots, ka testētāju vadītājs būtu arī projekta pārvaldnieks (ja vien projekta galvenais mērķis nav tieši testēšana). Šādu dalījumu nenosaka standarti, bet tā ir izveidojusies prakse, kas izriet no projekta darbinieku specializēšanās – testētāju vadītājam ir pilnu laika apjomu jādomā par testēšanu nevis jāsadala savs laiks un intereses starp implementāciju un testēšanu, kurām atsevišķos projekta etapos ir pretēji mērķi.

Ja aplūko testēšanas grupas veicamos darbus, tad projekta testēšanas grupas būtiskākās aktivitātes ir:

- testu gatavošana,
- testu izpilde un izpildes rezultātu reģistrēšana,
- problēmu analīze un reģistrēšana un
- problēmu labojumu pārtestēšana.

Testēšana ir atkarīga no spējas izsekot saistībai starp testu un tā pamatojumu, kā arī no spējas izpētīt sakarības starp testiem un to izpildes rezultātiem. Bez iepriekšminētā zūd jēga gatavot testus un reģistrēt problēmas. Ierindas testētājiem un grupas vadītājam ir atšķirīgi uzdevumi, un tādēļ arī pastāv atšķirīgs veids, kā tiek izmantota trasējamības īpašība.

Testu gatavošana

Testu gatavošana sākas ar mērķa definēšanu – kas tieši un kādā līmenī tiks testēts. Testiem par pamatu tiek izmantoti pieejamie informācijas avoti atbilstoši testu veidiem. Testētājam ir svarīgi saglabāt katra testa izveides pamatojumu – kādēļ šāds tests ir nepieciešams? Testa nepieciešamība var izrietēt no tā, ka dokumentos ir minēta atbilstoša prasība, vai arī testēšanas vadlīnijās ir norādīts, ka attiecīgā veida programmatūrai ir nepieciešami šādi testi. Var eksistēt testi, kuriem jāpārbauda tādas prasības, kas atklātā veidā specifikācijas nav minētas, bet izriet no attiecīgās klases programmatūras lietošanas konteksta. Testētāja uzdevums ir sekot tam, lai katrai testēšanas mērķim atbilstošai prasībai būtu vismaz viens atbilstošs tests. Savukārt, grupas vadītājam ir svarīgi iegūt kopskatu par testu gatavošanas gaitu, par prasību pārklājuma līmeni ar testiem un par problemātiskajām vietām testu definēšanā – vai ir kādas prasības, kurām ir nepietiekams skaits atbilstošu testu, vai ir tādas, kas definētā mērķa ietvaros būtu jātestē, bet tām nav testu utt. Atbilstošajai informācijas kopai ir jāspēj atbildēt uz jautājumu – „kas tiks testēts?"

Testu izpilde un izpildes rezultātu reģistrēšana

Testu izpilde var tikt organizēta dažādos veidos. Vismazāk iespējama ir pieeja, kur testi tiek izpildīti mehāniski pēc kārtas secībā, kādā tie ir veidoti vai sakārtoti pēc alfabēta. Izpildes secība ir atkarīga no tā, kādas koda daļas ir pieejamas, vai arī kāda ir

loģiskā secība darbam ar programmu. Testu izpildes laikā ierindas testētājam primārais nav redzēt testa saistību ar citiem projekta vienumiem. Ja tests ir labi veidots, tad tas saturēs visu nepieciešamo testa izpildei. Grupas vadītājam ir jāanalizē, kuras prasības ir katrā konkrētajā brīdī pārbaidītas, kuras programmas daļas ir pārbaudītas, kādi testi ir izpildīti un kādi vēl būs jāizpilda. Jāanalizē arī tas, kādi testi būtu nepieciešami papildus. Izpildes rezultātu reģistrēšana var izpausties vienkārši kā konkrēta testa izpildes statusa (piemēram, „der" vai „neder") atzīmēšana, vai arī tā var saturēt informāciju par šī testa piederību noteiktai testu kopai, testēšanas sesijai, par saistību ar detalizētu programmas aprakstu. Atbilstošajai informācijas kopai jāspēj atbildēt uz jautājumu – „kāds ir rezultāts?"

Problēmu analīze un reģistrēšana

Problēmu konstatēšanas gadījumā testētājs veic tās cēloņu un blakus parādību izpēti. Standarta gadījumā tiek atzīmēts, kurš tieši tests izraisīja problēmu. Papildus situācijas parakstam nepieciešams precīzi atzīmēt, kurā programmas daļā tā tieši tika konstatēta. To būtu nepieciešams identificēt ar tādu precizitāti un detalizāciju, kādu testētājs var nodrošināt ar savā rīcībā esošo informāciju (funkcija, modulis, ekrāns, apakšsistēma). Ja testētājs ir ar pietiekami labām zināšanām par testu saturu, tad testētājs var identificēt tos testus, kuri arī varētu saturēt līdzīga rakstura problēmas. Tas varētu būt iespējams, ja testētāja rīcībā būtu viegli pārraugāma informācija par testu saistību ar prasībām, ekrāniem, moduļiem.

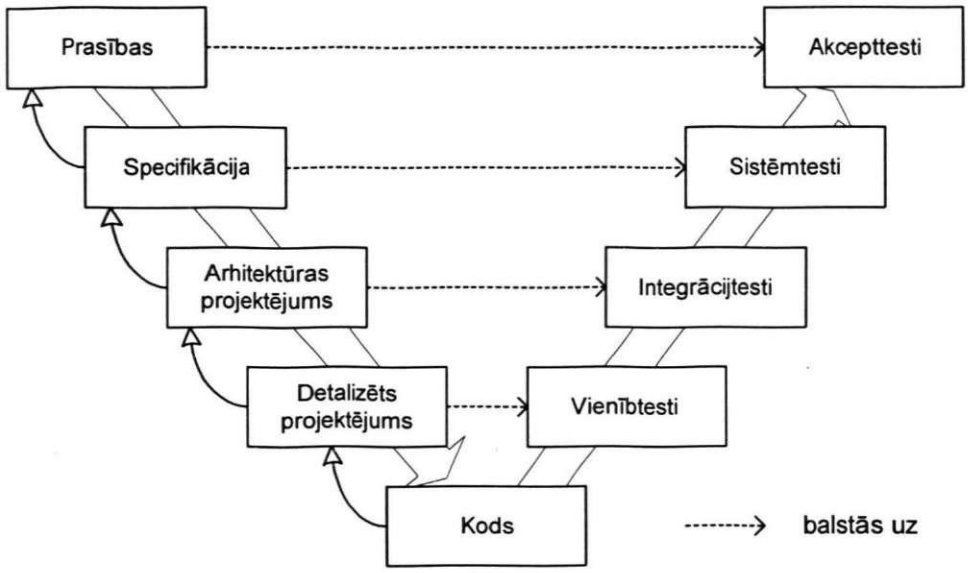
Problēmu labojumu pārtestēšana

Pārtestēšana ir aktuāla situācijās, kad iepriekš ir bijusi konstatēta problēma, tā ir izlabota, un ir nepieciešams pārliecināties par labojumu atbilstību. Testētājs var rīkoties vairākos veidos: pārbaudīt problēmziņojumā reģistrēto konkrēto situāciju, atkārtot visu testu no jauna, vai arī saprast problēmas būtību un izveidot jaunu papildus testu, kas pārbauda konkrēto īpašību detalizētā veidā. Pēdējā pieeja ir no testēšanas viedokļa viskvalitatīvākā, jo tā orientējas uz jaunu problēmu atrašanu un programmas korektas darbības gadījumā palielina mūsu pārliecību par tās atbilstošo kvalitāti. Lai realizētu minētā veida pieeju, ir nepieciešami līdzekļi analīzei, kā problēmziņojums saistās ar testu, prasībām un programmatūras elementiem. Jāizpēta, kādi eksistējošie testi attiecas uz minētajiem projekta vienumiem un kādi būtu nepieciešami kā papildinājums.

5.4. Trasējamības īpašības testēšanas procesā

Lai arī testēšana nav formāli ierindota standartā ISO/IEC 12207 kā viens no programmatūras projekta procesiem, to var atvasināt kā pielāgojamu procesu. Testēšana tipiski tiek strukturēta vairākos līmeņos. Kā jau bija minēts iepriekš, tas uzskatāmi parādās V-modelī, kas ir veidojies no Programmatūras dzīves cikla modeļa

(Software Lifecycle Process Model) [54] un kas bieži vien tiek attēlots vienkāršākā variantā [119]. Modeļa tradicionālais variants ir parādīts 2.attēlā.



2.attēls

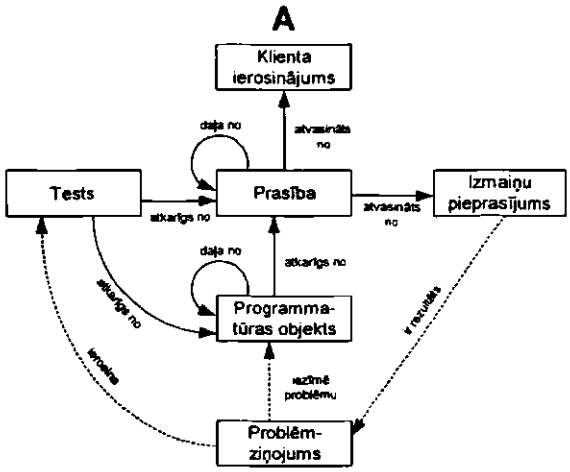
Eksistē arī šī modeļa variācijas, piemēram, W modelis [132]. Katram testēšanas līmenim ir savs testējamais objekts, un testi balstās uz sava veida projekta vienumiem (artefaktiem).

Katram projektam ir savs testēšanas process, un ne visiem projektiem ir uzdevumi, kas atbilstu katram testēšanas līmenim. Būtiskas atšķirības var parādīties atšķirīgi lietotos aktivitāšu vai dokumentu nosaukumos, lai arī pēc būtības tas ir viens un tas pats darbs. Piemēram, vienībtests var saukties komponenta tests, vai arī akcepttestēšana var sadalīties sīkāk ražotāja akcepttestēšanā un lietotāja akcepttestēšanā, vai arī šī līmeņa vietā var būt kvalifikācijas testēšana. Tam iemesls ir tas, ka ne vienmēr komerciāli projekti izmanto tieši kādu IEEE vai ISO grupas standartu, bet gan tas varētu būt kādas starptautiskas kompānijas iekšējais standarts, ko izmanto visi saistītie uzņēmumi un projekti. Piemēram, starptautiskā kompānijas IBM vairākās sfērās ir izstrādājusi savu iekšējo standartu kopas.

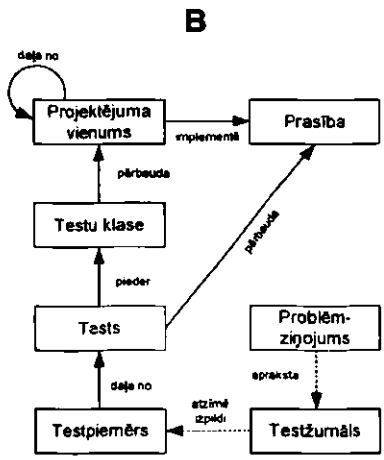
Autors ir konstatējis, ka tikai retos gadījumos projekti spēj pateikt, kāds dzīves cikls eksistē projektā. Pārsvarā kāds konkrēts modelis eksistē, bet tas nav aprakstīts kā dzīves cikla modelis, bet gan jāizsecina no definētajiem procesiem, vai arī reālās prakses. Jāpiezīmē, ka lielākā daļa modeļu, piemēram, tipiski parādītais V-modelis, neparāda cikliskumu starp dažādajām programmatūras izstrādes aktivitātēm. Parasti nav mazāk par diviem testēšanas līmeņiem, jo lielas sistēmas parasti veido vairāki cilvēki – katrs veido kādu nelielu daļu un pārbauda to savās iespēju robežās, un pēc tam tiek testēta sistēmas daļa vai visa sistēma.

Lai analizētu testēšanas procesa vienumus, autors sākotnēji apkopoja informāciju no 14 projektiem, kā arī pēc tam pārliecinājās par tās atbilstību vēl piecu konkrētajā brīdī aktīvu projektu gadījumā. Katram projektam bija mazliet atšķirīga izstrādes pieeja, bet tajā pašā laikā katru projektu var attiecināt uz standartā J-STD-

016 definētajiem dzīves cikla elementiem. Kā tipiskie tolaik identificētie piemēri tiek parādīti četri modeļi (A, B, C un D). Trīs no tiem atbilst specializētas informācijas sistēmas izstrādi un viens atbilst utilītprogrammas izstrādei.



3.attēls

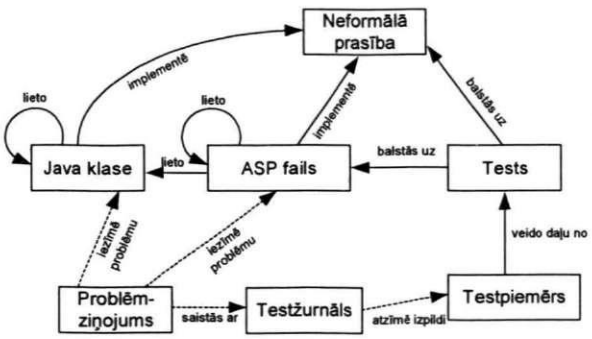


4.attēls

Projekts A (3.attēls) pārstāv vienu no populārākajiem veidiem, kā ir savstarpēji saistīti projektu vienumi. Formālās prasības ir atvasinātas no klientu piedāvājumiem, testi ir bāzēti uz šīm prasībām, un problēmas ir dokumentētas problēmziņojumos. Testu izpildes rezultāti tiek glabāti kopā ar testu informāciju. Vienumi, kas tiešā veidā saistīti ar testēšanu, ir testi un problēmziņojumi.

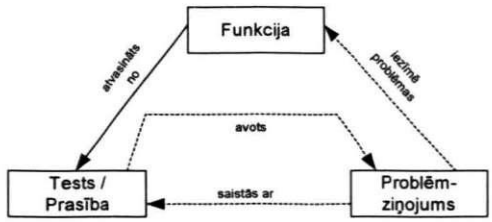
Projekts B (4.attēls) parāda projektu, kur ir stipri detalizēta testēšana. Testēšanas ietvaros tiek uzturēta informācija par testa klasi, testu, testpiemēru testžurnālu un problēmziņojumu. Šāds projekts vairāk atbilst situācijai, kad prasības tiek pakļautas minimālām izmaiņām un projekta galvenais uzdevums ir implementācija un testēšana nevis, piemēram, projektēšana.

C



5.attēls

D



6.attēls

Projekti C un D (5. un 6.attēli) atbilst situācijai, kad prasības var tikt pakļautas izmaiņām. Projekts C atbilst situācijai, kad tiek detalizēti aprakstīta implementācijas informācija. Šāda situācija bija novērojama, konkrēti tīmekļa programmatūras izstrādes projektā. Testēšanas procesa vienumi ir: tests, testpiemērs, testžurnāls un problēmziņojums. Testēšana ir raksturojama kā labi organizēta, un konkrētajā gadījumā bija tika pārsvarā veikti vienībtesti un integrācijas testi.

Savukārt, D projekts atspoguļo mazliet atšķirīgu situāciju – projekta pamatā nebija konkrēti definētu prasību, bet gan produkts tika izstrādāts iteratīvi. Lai arī šāda stila projekti nav dominējošie, tie raksturo to daļu projektu, kuros valda neformāla pieeja, un notiek ātra un evolucionāra izstrāde.

Autors konstatēja, ka ir iespējams visos 14 projektu trasējamību piekārtot četriem iepriekš minētajiem trasējamības modeļiem. Galvenā atšķirība parādās terminoloģijā, kas izriet no pielietotās metodikas. Piemēram, prasības var būt gan konkrēta diagramma, lietojumgadījums (*use case*), scenārijs, instrukcija, iezīme utt.

Testēšanas process un trasējamības modeļa īpašības

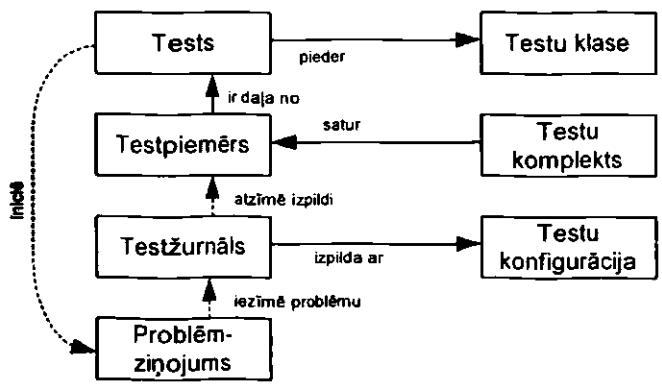
Ar testēšanu saistīti projekta vienumi ir uzskaitīti 7.tabulā.

7.tabula. Ar testēšanu saistīti projekta vienumi.

Tests	Kopa ar mērķiem, metodēm un novērtēšanas kritērijiem, ko izmanto konkrēta programmatūras vienuma pārbaudes nolūkos.

Testpiemērs	Konkrētam testēšanas mērķim sagatavota kopa ar ievaddatiem, izpildes priekšnosacījumiem un sagaidāmajiem rezultātiem.
Testu klase	Konteineris (ietvars) testiem ar līdzīgiem mērķiem, metodēm vai testēšanas līmeņiem.
Testu konfigurācija	Kopa ar papildus atribūtiem un nosacījumiem, kas būtu pielietojami testpiemēru izpildei.
Testu komplekts	Testpiemēru konteineris (ietvars), ko izpilda konkrētas testu sesijas vai scenārija ietvaros, vai arī ko izpilda noteikts personāls. Testu komplekts var būt implementēts automatizēto testu programmas (skripta) veidā.
Testžurnāla ieraksts	Informācija par testa izpildes statusu
Problēmziņojums	Testēšanas incidenta detalizēts apraksts, papildus izpēti prasošas situācijas fiksējums.

Trasējamības saišu analīze liecināja par to, ka saites nav gadījuma, bet gan eksistē tipiski virzieni un atkarības. Katra atsevišķa projekta gadījumā veidojas testēšanas procesa trasējamības modeļa apakškopa. 7. attēlā ir parādīts vispārīgais testēšanas procesa trasējamības modelis. Variācijas, kas var būt individuāla projekta ietvaros, var ietvert arī papildinājumus. Piemēram, kā atsevišķs vienuma tips varētu tikt pievienots automātisko testu skripts. Vēl viena pieļaujamā variācija attiecas uz to, ka ne vienmēr problēmziņojums ir saistīts tieši ar testēšanas žurnālu. Saite var būt pretējā virzienā, vai arī problēmziņojums var norādīt pat tiešo uz testu vai testpiemēru. Arī testu komplekts var būt saistīts ar saiti pretējā virzienā un attiekties ne tikai uz testpiemēriem, bet gan testiem.



7.attēls. Vispārināts trasējamības modelis testēšanas procesam.

Vienumi, kas ir 7.attēla kreisajā pusē, pārstāv tipiskos informācijas tipus, kas ir definēti standartā IEEE J-STD-016. Testēšanas procesa informācijas kopai ir jāsniedz atbildes uz šādiem trim jautājumu apgabaliem, kas ir svarīgi projekta vadībai un produkta kvalitātei:

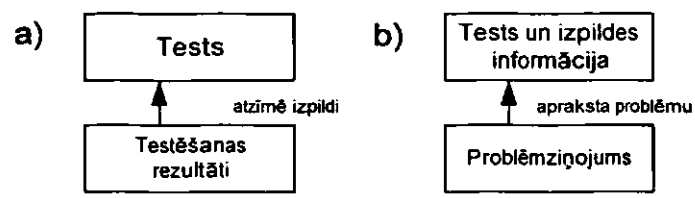
- kas un kā tiks (vai tika) testēts?
- kad un kā testi tika izpildīti?
- kādas bija konstatētās problēmas?

7.attēla labajā pusē esošie vienumi pilda klasificējošu lomu – dažos gadījumos tie ir nevis kā atsevišķi vienumi, bet gan atribūti vienumiem kreisajā pusē.

Industriālos projektos vistipiskākā pieeja ir atsevišķi neizdalīt testus un testpiemērus. Visus sauc par testiem, un vienums var būt vai nu ļoti detalizēts (kā testpiemērs), vai tas var aprakstīt tikai galveno virzienu (kā tests). Ņemot vērā to, ka praktiski ikviens projekta lēmums tiek pieņemts, izvērtējot arī ekonomisko izdevīgumu, projekta darbos, tajā skaitā testēšanā tiek meklēta optimizācija – samazināta dokumentu kopa, optimāla testu izvēle utt. Autors novēroja šādas optimizācijas divus variantus līdz pat diviem projekta vienumiem (8.attēls):

Ir definēti testi atsevišķi, bet visi testēšanas rezultāti tiek uzturēti kā viens testžurnāls (tajā skaitā ietverot problēmu aprakstus). Tas tiek praktizēts nelielos projektos, kur nav būtiski nodibināt formalizētu komunikāciju starp izstrādātājiem un testētājiem, bet tajā pašā laikā testētāji nodrošina testēšanas gaitas un problēmu reģistrēšanu (gadījums a).

Testi, kas definēti mērķu un galveno soļu līmenī ar klāt uzturēto izpildes rezultātu informāciju, un problēmziņojumi. Tā ir raksturīgāka situācija lielākiem projektiem, kur ir būtiska formalizēta informācijas apmaiņa starp izstrādātājiem un testētājiem (gadījums b).



8.attēls. Iespējamās testēšanas procesa trasējamības modeļa optimizācijas.

Daudzos projektos ir novērojama iniciatīva atcelt testu aprakstīšanu, vai arī to darīt tikai tādā līmenī, kā prasības formulējuma kopijai pievienot sākumā frāzes „pārbaudīt, vai", „pārliecināties, ka" utt. Lai arī sākotnēji šāda pieeja var nodrošināt labu trasējamību un visu prasību pārklājumu, tā nenodrošina labu testēšanu [31]. Tests nedrīkst palikt tikai vienas prasības pārbaude vienā konkrētā veidā.

Testēšanā bāzes elements ir tests – šim vienuma tipam ir jābūt projektā, ja tiek uzskatīts, ka projektā notiek testēšana. Visi citi ar testēšanu saistītie projekta vienumi ir atkarīgi no testa. Testa atkarība no problēmziņojuma nespēlē būtisku lomu – tā parāda, ka testēšanas rezultātā var paplašināties testu kopa.

Testēšanas procesa ārējās saites

Testēšanas procesa trasējamības modelis nav izolēts no citiem projekta vienumiem. Konkrēti, testam un problēmziņojumam ir saites ar vienumiem ārpus testēšanas procesa.

„Ārējie" vienumi, kas ir saistīti ar testu, ir iedalāmi trijās grupās:

- programmatūras sistēmu definējošie vienumi (prasības, projektējuma vienumi),
- kods,
- blakusprodukti (piemēram, lietotāja rokasgrāmata).

Tests parasti ir stingri atkarīgs no šiem vienumiem. Eksistē sakarība starp testēšanas līmeņiem un testa atkarības no vienumiem. Tests var būt atkarīgs arī no problēmziņojumiem, bet to ietekmē nevis testēšanas līmenis, bet testēšanas procesa organizēšana.

Saites ar sākotnējām vai precizētajām prasībām funkcionālajai testēšanai ir uzskatāmas par normu, jo tas sniedz vistiešāko atbildi uz jautājumu par to, vai klienta prasības ir ievērotas. Projektējuma vienumi un koda elementi par pamatu testiem tiek izmantoti parasti izstrādātāju veiktajā testēšanā, jo, saskaņā ar tradicionālo situāciju, ne lietotājiem, ne neatkarīgajiem testētājiem nav iespējas izmantot iekšējo izstrādes informāciju.

8.tabula. Vienumi, kas saistīti ar testu un tiem atbilstošiem testēšanas līmeņi.

Projekta vienums	Testēšanas līmenis
Sākotnējās prasības	Akcepttestēšana
Prasības	Sistēmtestēšana
Projektējuma vienums	Integrācijtestēšana
Koda elements	Vienībtestēšana
Lietotāja rokasgrāmata	Sistēm- un akcepttestēšana
Problēmziņojums	Vienīb-, integrācijas, sistēm- un akcepttestēšana

Saite ar lietotāja rokasgrāmatu var parādīties divos gadījumos – vai nu testējamais objekts ir dokumentācija, vai arī gadījumos, ja testējamai programmatūrai nav formāli specificētu prasību. Pēdējo minēto gadījumu var uzskatīt arī par īpašu pieeju programmatūras izstrādei, jo lietotāja rokasgrāmatas izmantošana ļauj vieglāk, lētāk un precīzāk uzturēt vienotu dokumentu, kas apraksta programmatūru [37]. Jāpiebilst, ka pasūtītāji ne vienmēr prasa prasību specifikāciju, bet lietotāja dokumentāciju gan. Testa saite ar problēmziņojumiem ir tipiska procesa pieejai, ka katras jaunas problēmas gadījumā tiek nodefinēts tests, kas turpmāk ļaus pārbaudīt, vai iepriekš konstatētā problēmsituācija joprojām ir spēkā.

Problēmziņojums ir notikuma izraisīts projekta vienums, tādēļ ar citiem vienumiem tas ir saistīts ar nestingrajām saitēm - nekādas izmaiņas citos vienumos neizraisa izmaiņas problēmziņojumā. Problēmziņojums var būt saistīts praktiski ar jebkuru vienumu projektā, bet vistipiskākās ir tiem pašiem vienumiem, ar ko ir saistīts tests. Atkarībā no trasējamības modeļa problēmziņojumam var būt dažādas netiešās ar citiem vienumiem, piemēram, caur saskarnes elementu ar prasībām. Konkrēto saišu esamība ir atkarīga no testēšanas līmeņa. Katrā konkrētajā projektā tiek izmantota tikai daļa no augstākminētā. Problēmziņojuma saites ar testu vai testžurnālu parāda, kāda aktivitāte ir bijusi pamatā problēmziņojumam. Projektējums, kods, saskarnes

elementi vai lietotāja rokasgrāmata parāda testētāja novērto problēmas atrašanās vietu. Atsauce uz prasībām var tikt lietota, lai norādītu uz problēmu prasībās vai no tām atvasinātos vienumos.

9.tabula. Vienumi, kas saistīti ar problēmziņojumu un atbilstošajiem testu līmeņiem.

Projekta vienums	Testēšanas līmeņi
Tests	Vienības, integrācijas, sistēmas, akcepttestēšanas
Testžurnāla ieraksts	Vienības, integrācijas, sistēmas, akcepttestēšanas
Projektējuma elements	Integrācijas, sistēmtestēšanas
Saskarnes elements	Sistēmas, akcepttestēšanas
Koda elements	Vienības, integrācijas
Prasības	Akcepttestēšanas
Lietotāja rokasgrāmata	Sistēmtestēšanas

Papildus iepriekš minētajam, ir saite, kur ārējs vienums ir atkarīgs no problēmziņojuma – izmaiņu pieprasījums. Parasti tas tālāk ir saistīts ar prasībām (9.attēls).

9.attēls.

Programmatūras testēšanas procesa trasējamības modeļi neatkarīgi no piederības dzīves cikla modelim parāda kopīgas īpašības. Ir ievērojama līdzība starp testēšanas procesiem, un nav konstatēti pretrunīgi principi. Katrā projektā novērojamais ar testēšanu saistītais trasējamības modelis ir apakškopa no vispārīgā testēšanas procesa trasējamības modeļa. Būtiskākās atšķirības ir katrā projektā vai organizācijā lietotajā terminoloģijā.

Interesanti pētāmie jautājumi būtu ar testēšanu saistītās trasējamības atkarība no projekta lieluma, ilguma un darba uzdevuma.

No iepriekšminētās testēšanas procesa un trasējamības saistības analīzes var gūt papildus pamatojumu apgalvojuma (a) daļai - testēšanas process ir atkarīgs no trasējamības kvalitātes.

5.5. Situācija Latvijas IT projektos

Aptaujas mērķi

Autors bija iepazinies ar citu autoru ([30], [59], [123]) veiktajiem pētījumiem par trasējamību, un intuitīvs novērtējums liecināja, ka arī autora apkārtnē (kas vienlaikus ir Latvijas vadošā IT uzņēmuma situācija) sastaptajos projektos ir līdzīga rakstura problēmas – nepietiekama trasējamības praktizēšana, jo ir grūti pārvaldīt lielo informācijas apjomu. Tomēr savas pieredzes, publikācijās minēto problēmu un citos

Latvijas IT uzņēmumos esošās situācijas salīdzināšanas nolūkos autors rīkoja aptauju par trasējamības praksi dažādos Latvijas IT projektos.

Autors bija veicis iepriekšējus pētījumus par testēšanas procesā esošajām tipiskajām informatīvajām saitēm. Aptauja tika veikta Latvijas IT uzņēmumos ar mērķi iegūt priekšstatu par reālos programmatūras projektos pastāvošo informācijas trasējamības praksi. Jaunā pētījuma mērķi bija:

- apzināt projektu pieredzi vairākos Latvijas IT uzņēmumos;
- uzzināt projektu dalībnieku viedokli par trasējamību;
- iegūt informāciju par informatīvajām saitēm ārpus testēšanas procesa;
- iegūt priekšstatu par reāli pastāvošo atšķirīgo trasējamības pieeju amplitūdu;
- novērtēt rezultātus līdzīgu pētījumu kontekstā;
- ielikt pamatu trasējamības modeļu sagatavēm, kuras varētu izmantot trasējamības rīka konfigurēšanai.

Ņemot vērā to, ka katrā uzņēmumā ir mazliet atšķirīga tipa projekti, ir bijusi atšķirīga attīstības vēsture un ir nostiprinājusies sava darba kultūra, autors bija gatavs sagaidīt dažādību. Galvenais jautājums, uz kuru tika meklēta atbilde - vai dažādu uzņēmumu projektos pastāv vienāda tipa saites ap konkrētiem kodola vienumu tipiem?

Rezultātu aktualitātes kontekstā mērķis bija iegūt priekšstatu par projektiem, kas ir aktīvi aptaujas norises laikā, vai arī ir neilgi pirms tam noslēgušies. Autors apzinājās tādu problēmu, ka praktiski nav iespējams atlasīt reprezentatīvu projektu kopu. Tam ir šādi iemesli:

- komerciālu apsvērumu dēļ nav iespējams noskaidrot, kuros Latvijas uzņēmumos šobrīd norit vai ir noritējuši projekti,
- komerciālu apsvērumu dēļ nav iespējams noskaidrot, cik un kādi projekti norit katrā konkrētā IT vai ar IT saistītā uzņēmumā,
- jēdziens „projekts" dažādos uzņēmumos var tikt interpretēts atšķirīgi atkarībā no uzdevuma, norises laika un iesaistīto cilvēku skaita.

Iepriekšminēto iemeslu dēļ autors izvēlējās pieeju apzināt Latvijas lielāko IT uzņēmumu pieredzi, ļaujot pašu uzņēmumu pārstāvjiem piedāvāt stāstīt par sevis izvēlētiem projektiem. Autora lūgums šādas pieejas kontekstā bija uzņēmumiem stāstīt par dažāda tipa projektiem (t.i., lai nav informācija par atšķirīgiem projektiem, kuri tomēr izmanto vienu un to pašu darba pieeju), kas ir tipiski attiecīgajam uzņēmumam. Pie šādas pieejas nav iespējams apgalvot, ka ir iegūts reprezentatīvs skatījums par uzņēmumos esošo projektu kopu, bet autors uzskata, ka aptauja:

- sniedz informāciju par tendencēm trasējamības kultūras attīstībā,
- raksturo tipiskākos trasējamības modeļus projektos,
- var kalpot par pamatu ieteikumu sagatavošanai trasējamības uzlabošanai.

Aptaujas norise

Aptauja tika veikta laika periodā no 2003.gada decembra līdz 2004.gada novembrim. Intervējamās personas bija projekta pārvaldnieki vai arī tādi projekta pārstāvji, kuri var kompetenti pastāstīt par projektā eksistējošo informāciju, tās saitēm un darba procesu. Ņemot vērā intervējamo personu noslodzi, iecere bija pēc iespējas minimāli patērēt viņu laiku. Dalībnieks anketu aizpildīja klātienē ar aptaujas vadītāju. Pirms konkrētās tikšanās reizes aptaujas vadītājs sagatavošanas nolūkos nosūtīja dalībniekiem elektroniskā formā aptaujas anketu un pavadvēstuli par aptaujas norises formu (skat. Pielikumu), bet neprasīja šo anketu aizpildīt. Klātienes tikšanās reizē daļa dalībnieku bija mēģinājuši aizpildīt anketu, daļa bija īpaši padomājuši par trasējamības tēmu, bet citi tam nebija nemaz veltījuši laiku. Lielākā daļa nebija pārliecināti, ka autoru var interesēt tāds specifisks jautājums kā trasējamība, jo tas lielākajā tiesā gadījumu projekta pārstāvjiem nelikās kā īpaši būtisks projekta norisei. Daži uztvēra aptauju piesardzīgi, jo tās ietvaros tiek iegūts noteikts priekšstats par uzņēmuma projektiem, un IT kompānijas nevar būt drošas par šīs informācijas izmantošanas mērķiem. Tomēr autors atteikumu saņēma tikai atsevišķos gadījumos, kur tikai neliela daļa konkrēti minēja kā iemeslu konfidencialitāti, bet pārējos gadījumos bija konkrēto personu aizņemtība. Autoram no netiešām pazīmēm ir pamats uzskatīt, ka daļā atteikumu iemesls varētu būt arī kompāniju pārstāvju apzināšanās par slikti organizētu trasējamību un nevēlēšanās to plaši afišēt.

Ja viena un tā pati persona bija gatava stāstīt par vairākiem projektiem, noruna bija, ka viņa izvēlas dažāda tipa projektus. Zināms ierobežojums informācijas izpaušanai ir projektu komerciālais raksturs, un šī iemesla dēļ intervijas laikā tika minēti abstrakti projekta nosaukumi tādi, kā Lielais projekts, Otrais projekts, A projekts utt. Kā jau bija minēts iepriekš, projektu pārstāvjiem svarīga nianse bija arī tāda, ka iespējama konkrēto rezultātu publiskošana par konkrēto uzņēmumu var potenciāli negatīvi ietekmēt uzņēmuma tēlu. Tādēļ jau pirms aptaujas norises autors deklarēja, ka rezultātu atspoguļojumā nevienā vietā nefigurēs rezultāti, ka saista konkrēta uzņēmuma vai aptaujas dalībnieka vārdu ar kādu no atbilžu variantiem. Anketās aptaujas dalībnieka vārds tiek reģistrēts tikai ar mērķi, lai nepieciešamības gadījumā varētu sazināties ar dalībnieku. Aptaujā netika iekļauti komerciālas dabas vai projekta konkrētā darba jautājumi, lai arī tas varētu dot kādu interesantu informāciju.

Klātienes tikšanās ilgums nepārsniedza 40 minūšu sarunu par katru projektu. Sarunas laikā, papildus iepriekš sagatavotās anketas aizpildīšanai, autors no respondentiem uzzināja dažādus projekta norises faktorus, kas ietekmē trasējamību. Šīs atziņas ir atspoguļotas nākošā nodaļā.

Aptaujas rezultāti

Aptaujas ietvaros tika apzināti 32 projekti no 6 Latvijas IT uzņēmumiem. Visos gadījumos projektu pārstāvji zināja, kas ir trasējamība, tomēr tikai atsevišķi bija tādi, kur pārstāvji paši norādīja uz trasējamību kā projekta īpašību, kas ļāva labi veikt

darbus. Arī tajos gadījumos, kad projekta pārstāvis skeptiski vērtēja trasējamības nozīmi savā konkrētajā projektā, autors pēc tālākās sarunas un uzzīmētā trasējamības konstatēja, ka trasējamību projekti tomēr ir praktizējuši. Lai arī dažos projektos trasējamība nebija nekādi īpaši nodrošināta, projekti arī īpašas problēmas nav izjutuši. Iespējams, tam pamatā ir tas, ka minētie projekti ir nelieli, kur atsevišķas personas zināja visu, un dokumentēšana tajā brīdī nedotu reālu labumu. Trasējamību novērtē kā labu īpašību, bet tajā pašā laikā dārgu – prasa papildus darbu.

Projekti atzina, ka viena no galvenajām problēmām trasējamības uzturēšanā ir nepieciešamība pēc tehniski laba programmatūras nodrošinājuma. Ja būtu tehniskas metodes, kas ļautu to visu atvieglot, tad nebūtu nekādu iebildumu. Nebija neviena, kas noliegtu trasējamības nepieciešamību, bet iebildumi bija pret to, ka tā varētu prasīt papildus laika un tehniskos resursus. Vairāki uzsvēra trasējamību kā vienu no akadēmiskās aprindās minētām lietām, kas līdz praksei nonāk ar grūtībām, jo nav labu veidu, kā projektiem to nodrošināt.

Visos gadījumos, kad ir bijusi negatīva pieredze, tas saistījās ar to, ka kādā no iepriekšējiem projektiem ir bijis jāveido trasējamības tabulas, kuru noderīgums nebija zināms (un statiskā formā tās ir grūti lietojamas). Slikta sākotnējā pieredze nebija radījusi vēlēšanos meklēt labāku pieeju, bet gan vienkārši atteikties vai arī formāli paciest attiecīgās tabulas izveidi. Tas izraisīja arī principu to projekta gaitā konsekventi turpmāk vairs neizmantot (jo tajā netiek ieviestas aktuālās izmaiņas, un tā vairs nav noderīga reālam darbam).

Projekta raksturlielumi

Pēc darbu tipa lielākā daļa aptverto projektu bija pilnas izstrādes projekti (71,9%). Pārējie ir ierindojami pie uzturēšanas (15,6%), prasību specificēšanas vai projektējuma sagatavošanas (9,4%) un reinženierēšanas (3,1%) projektiem. Aptaujas ietvaros netika aplūkots sīkāks dalījums, jo pie aptaujāto projektu skaita tas nevarētu kalpot par pamatu tālākiem secinājumiem. Pēc projekta grupas izmēra dalījums veidojās šāds:

- no 1 līdz 4 cilvēki - 53,1%,
- no 5 līdz 15 cilvēki - 15,6%,
- no 16 līdz 40 cilvēki - 18,8%,
- 41 un vairāk cilvēku - 12,5%.

No aptaujātajiem projektiem lielākā daļa bija ilgtermiņa projekti: virs viena gada ilga 50,0%, starp 3 mēnešiem un 1 gadu - 40,6%, bet mazāk par 3 mēnešiem - 9,4%. Arī autora pieredze ārpus šīs aptaujas liecina, ka aptuveni puse projektu ir ilgāki par gadu, un pārējie ir līdz gadam, kā arī lielākā daļa projektu iesaista samērā nelielu cilvēku skaitu.

Lai arī projektu skaits nav pietiekami liels, lai raksturotu likumsakarības starp izmēru un ilgumu, aptaujātās grupas ietvaros var novērot, ka līdz 3 mēnešiem ir tikai tādi projekti, kuros ir 1-4 cilvēki. Savukārt, to projektu vidū, kas ilgst no 3 mēnešiem

līdz 1 gadam, mazie projekti (1-4) veido 53,8%. Neviens pilnas izstrādes projekts nebija ar ilgumu līdz trīs mēnešiem, bet sadalījās aptuveni vienādi sadalījās starp no 3 mēnešiem līdz 1 gadam un virs 1 gada ilgušajiem. Kopumā var secināt, ka aptaujas ietvaros ir izdevies iegūt tādu projektu kopu, kas atbilst autora priekšstatiem par projektu sadalījumu tipiskā Latvijas IT uzņēmumā.

Vispārējā prakse

Viens no jautājumiem projekta pārstāvjiem bija par to, vai projektā īpaši tika domāts par trasējamību. Pozitīvi atbildēja 53,1%, noliedzoši - 21,9%, bet daļēji par trasējamību ir domājuši 25,0%. Šis bija tiešā veidā formulēts jautājums, un uz tā sniegtās atbildes var liecināt par respondentu viedokli attiecībā pret trasējamību, viņu izpratni saistībā ar jēdzienu „trasējamība", gan arī vēlmi apliecināt formālu kvalitātes prasību ievērošanu. Objektīvāks trasējamības raksturojums ir iespējams pēc trasējamības modeļa. Jo vairāk trasējamības modelī ir saistītu vienumu tādā apjomā, ka ar to palīdzību ir iespējams nodrošināt procesus, jo vairāk trasējamību var uzskatīt par labi iztrādātu. Ja ir zināms, ka tiek veidoti testi, kuriem būtu jāpārklāj visas prasības, bet trasējamības modelis neuzrāda tiešu vai netiešu testu saistību ar prasībām, tad tas ir indikators par sliktu trasējamības praksi. Ja, savukārt, modelī šādu saišu nav, jo konkrētajā projektā testi ir balstīti uz kādiem likumdošanas dokumentiem, tad šāda situācija varētu kļūt arī pieņemama.

Lielākajā daļā gadījumu projektu atspoguļotais trasējamības modelis atbilst pēdējam vai aktuālākajam variantam - 78,1%, bet daļēji - 15,6%. Pēdējais minētais nozīmē, ka modelis tika vai tiks mainīts projekta izmaiņu dēļ, un beigu variants vairāk raksturo projekta reālās vajadzības. Atbildot uz jautājumu par to, vai projekta metodika tika pārņemta no kāda veiksmīga iepriekšēja piemēra, atbildes vienādi sadalījās starp tiem, kas ir un tiem, kas nav pārņēmuši metodiku (46,9%). Ja projekta pārstāvji tiek lūgti novērtēt, kā projekta pieredze ir vai tiks izmantota arī citos projektos, tad lielākā daļa (71,9%) atbilde pozitīvi, bet 15,6% atbild noraidoši, jo ir saskatījuši problēmas attiecīgajā modelī. Liela daļa respondentu piebilda, ka arī no neveiksmēm var mācīties, tādēļ arī trasējamības pieejas, kurās bija problēmas, tiks izmantotas kā pozitīvi vai negatīvi piemēri.

Jāsecina, ka lielākā daļa projektu trasējamības tipa informāciju neuztur ar rīku palīdzību. Uztur 18,8 %, daļēji uztur – 37,5%, bet neuztur - 43,8%. Autors nenovēroja korelāciju starp projekta kvantitatīvajiem raksturlielumiem (izmērs, ilgums) un trasējamības praksi. Vienīgi var atzīmēt, ka proporcionāli vairāk par trasējamību ir domājuši projekti, kuru ilgums ir virs 1 gada vai grupas izmērs ir 1-4 cilvēki.

Vienumi un saites

Anketas aizpildīšanas process ietvēra individuālo projekta vienumu identificēšanu un trasējamības modeļa uzzīmēšanu. Nebija divu absolūti vienādu trasējamības modeļu. Pielikumā ir sniegti visi aptaujas laikā iegūtie trasējamības

modeļi. Vienumu tipu skaits modelī bija robežās no 2 līdz 13 vienumiem. To var iedalīt šādās grupās:

- no 2 līdz 4 (6 gab.)
- no 5 līdz 6 (8 gab.)
- no 7 līdz 8 (6 gab.)
- no 9 līdz 10 (6 gab.)
- no 11 līdz 13 (6.gab.)

Vidēji modelī bija 7,3 vienumu tipi. Saišu skaits modeļos bija robežās no 1 līdz 16. Tās var iedalīt šādās grupās:

- no 1 līdz 4 (5 gab.)
- no 4 līdz 6 (8 gab.)
- no 7 līdz 9 (7 gab.)
- no 10 līdz 13 (6 gab.)
- no 14 līdz 16 (6 gab.)

Vidēji modelī bija 8,2 dažāda veida saites. Attiecība „saites pret vienumiem" ir robežās no 0,5 līdz 1,4. Vidēji minētajiem modeļiem šī vērtība ir 1,04.

Tālāk seko biežāk sastopamo vienumu un to saišu uzskaitījums. Vairākos gadījumos bija redzams, ka projektā konkrēts vienumu tips tika saukts vienā vārdā, bet citā projektā saturiski tāds pats vienuma tips bija ar citu vārdu. Šādās situācijās autors vienumu nosaukumus uztvēra kā sinonīmus, un tālāk ir norādīts to aptverošais nosaukums. Terminoloģiskā daudzveidība vienumu tipu nosaukumos izriet no IT uzņēmumu atšķirīgās vēsturiskās pieredzes un iestrādātajām tradīcijām. Būtiskākais ir saturiski noskaidrot, kas ir attiecīgais vienuma tips.

Trasējamības modeļu analīzes process bija šāds:

1. Vienumu tipu sastopamības identificēšana. Attiecīgā pētījuma ietvaros par biežāk sastopamajiem vienumiem tiek uzskatīti tie, kas ir vismaz 25% modeļu (jeb vismaz 8 no 32). Biežāk sastopamie vienumi ir parādīti 10.tabulā.
2. Biežāk sastopamajiem vienumiem nosaka saites ar citiem vienumiem (gan biežāk sastopamajiem, gan pārējiem). Rezultāti ir parādīti 11.-20.tabulās.
3. No identificētajām saitēm atlasa tās, kuras ir aptaujā biežāk sastopamās. Saite tiek ierindota pie biežāk sastopamajām pēc šādiem kritērijiem: vienumiem ar sastopamību 61 vai vairāk procenti – jābūt saitēm vismaz 10% gadījumu; ar sastopamību 40-60% - saites 15% gadījumu; zem 40% - saites vismaz 20% gadījumu.

10. tabula.

Vienuma tips	sastopams modeļos	%

Koda elements (klase, funkcija)	26	81,3
Problēmziņojums	24	75,0
Prasība (detalizētā, formālā)	22	68,8
Projektējuma vienums	20	62,5
Izmaiņu pieprasījums	19	59,4
Neformālās (sākotnējās, pamata, augsta līmeņa, piedāvājuma) prasības	15	46,9
Tests	12	37,5
Testpiemērs	12	37,5
Testžurnāls, testu rezultāts	11	34,4
Modelis (datu, ER, ...)	8	25,0

No vienumu sastopamības biežuma var secināt, ka nav tādu vienumu tipu, kuri būtu visos projektu trasējamības modeļos. Jāņem vērā, ka katram projektam ir savi uzdevumi un notiekošie procesi. Tas arī nosaka projektā apstrādājamos informācijas tipus un attiecīgi arī vienumu tipus trasējamības modelī. Tālākajās tabulās procenti apzīmē vienumu sastopamību tajos modeļos, kur attiecīgais vienums vispār bija (t.i. netiek aplūkoti tie gadījumi, kur šāda vienuma nebija). Tabulās ir norādīts saites tips (e – evolūcijas, n – notikuma), bet pēc tā nav veikta papildus grupēšana.

11. tabula.

Koda elements			
Virziens	Saite	Vienuma tips	Biežums (%)
Vecāki	e	Projektējuma vienums	**42,3**
	e	Problēmziņojums	**15,4**
	e	Koda elements	**11,5**
	e	Formālā prasība	**11,5**
	e	Modelis	**11,5**
	e	Modulis	**11,5**
	e	Izmaiņu pieprasījums	3,8
	n	Piegāde	3,8
	e	Biznesa funkcija	3,8
	e	Ārējās sistēmas projektējums	3,8
Bērni	n	Problēmziņojums	**30,8**
	e	Koda elements	**11,5**
	n	Izmaiņu pieprasījums	3,8
	e	Testpiemērs	3,8
	n	Testžurnāls	3,8
	n	Modelis	3,8
	n	Piegāde	3,8
	n	Moduļa izmaiņas	3,8

Iegūtie rezultāti liecina, ka koda elementam vistipiskākie vecāki ir projektējuma vienums, problēmziņojums, cits koda elements, formālā prasība, modelis un modulis. Koda elementa tipiskākie bērni ir probēmziņojums vai cits koda elements. Koda

elements kā vienuma tips trasējamības modelī nav sastopams gadījumos, ja projekts ir fokusēts uz testēšanu vai projektējuma/specifikācijas izstrādi. Koda elements var saistīties ar citu koda elementu gadījumos, kad ir izdevīgi uzturēt informāciju par koda struktūru, tā hierarhiju. Apzinātajos projektos tās ir, piemēram, koda funkcijas, metodes un klases.

12. tabula.

Problēmziņojums

Virziens	Saite	Vienuma tips	Biežums (%)
Vecāki	n	Koda elements	**29,2**
	n	Testpiemērs	**29,2**
	n	Tests	**25,0**
	n	Projektējuma vienums	**20,8**
	n	Formālā prasība	**12,5**
	n	Modulis	8,3
	n	Izmaiņu pieprasījums	4,2
	n	Testžurnāls	4,2
	n	Ekrāns	4,2
	n	Piegāde	4,2
	n	Biznesa funkcija	4,2
	n	Programmatūras objekts	4,2
	n	Uzdevums projekta plānā	4,2
Bērni	e	Koda elements	**16,7**
	n	Testžurnāls	**12,5**
	e	Formālā prasība	4,2
	n	Izmaiņu pieprasījums	4,2
	e	Testpiemērs	4,2
	n	Modulis	4,2
	n	Moduļa izmaiņas	4,2

Problēmziņojuma tipiskie vecāki ir koda elements, testpiemērs, tests, projektējuma vienums un formālā prasība. Tipiskākie bērni - koda elements un testžurnāls. Problēmziņojums eksistē visos tajos gadījumos, kad projektā notiek testēšana. Atsevišķos gadījumos nelieli projekti bez tā ir iztikuši, tomēr nav novērots, ka mūsdienās kāds šaubās par problēmu reģistrēšanas lietderību. Jautājums ir tikai par to, kā to dara un ko tajā ietver. Parasti problēma nav vienkārši tāpat, bet gan attiecas uz kādu programmas koda daļu vai dokumentu. Ja ir reģistrēta problēma, atbildē uz to seko darbība - izmaiņas, kas nodrošina problēmas novēršanu. Tie ir problēmziņojuma bērni.

13. tabula.

Prasība (detalizētā, formālā)

Virziens	Saite	Vienuma tips	Biežums (%)

Vecāki	e	Sākotnējās, neformālās prasības	**54,5**
	e	Izmaiņu pieprasījums	**36,4**
	e	Scenārijs, biznesa process	9,1
	e	Standarts	4,5
	e	Piedāvātais risinājums	4,5
	e	Problēmziņojums	4,5
	e	Likumdošana	4,5
	e	Intervijas piezīmes	4,5
	e	Biznesa funkcija	4,5
	e	Līgums	4,5
	e	Koncepcija	4,5
Bērni	e	Projektējuma vienums	**59,1**
	e	Tests	**27,3**
	e	Koda elements	**13,6**
	n	Problēmziņojums	**13,6**
	e	Testpiemērs	**13,6**
	e	Izmaiņu pieprasījums	9,1
	e	Ekrāns	9,1
	n	Piegāde	9,1
	e	Modelis	4,5
	e	Koncepcija	4,5
	e	Modulis	4,5
	e	Scenārijs	4,5
	n	Testu komplekts	4,5
	e	Biznesa funkcija	4,5
	e	Testa plāns	4,5
	e	Apakšistēmas elements	4,5
	e	Lietotāja dokumentācija	4,5

Detalizētajai vai formālajai prasībai tipiskie vecāki ir sākotnējās jeb neformālās prasības un izmaiņu pieprasījums. Vecāki ir tie, kas attiecas uz pre-RS trasējamību – prasību izcelsmi. Tipiskākie bērni ir projektējuma vienums, tests, koda elements, problēmziņojums un testpiemērs. Prasība ir vienuma tips, kam visvairāk dažādu saistīto bērnu, jo tieši no prasībām seko tālākais programmatūras saturs. Papildus aptaujā identificētajām ir iespējami arī citi bērni.

14. tabula.

Projektējuma vienums			
Virziens	Saite	Vienuma tips	Biežums (%)
Vecāki	e	Formālā prasība	**60,0**
	e	Izmaiņu pieprasījums	**40,0**
	e	Sākotnējās, neformālās prasības	10,0
	e	Modelis	5,0
	e	DB entītija	5,0
	e	Eksistējošas sistēmas īpašība	5,0

	e	Ārējās sistēmas projektējums	5,0
Bērni	e	Koda elements	**50,0**
	e	Tests	**25,0**
	n	Problēmziņojums	**20,0**
	e	Modulis	**15,0**
	e	Modelis	10,0
	e	DB entītija	10,0
	e	Testpiemērs	5,0
	e	Ekrāns	5,0
	n	Testu komplekts	5,0
	e	Testa plāns	5,0
	e	Apakšistēmas elements	5,0

Projektējuma vienuma tipiskākie vecāki ir formālā prasība, izmaiņu pieprasījums. Tipiskākie bērni ir koda elements, tests, problēmziņojums un modulis. Projektējuma vienums var būt tuvināts prasībām, vai arī būt tuvs kodam. Ir dažādas formas, kā var pierakstīt projektējuma vienumus.

15. tabula.

Izmaiņu pieprasījums

Virziens	Saite	Vienuma tips	Biežums (%)
Vecāki	n	Problēmziņojums	10,5
	n	Formālā prasība	10,5
	n	Koda elements	5,3
	n	Likumdošana	5,3
	n	Apakšsistēmas elements	5,3
Bērni	e	Projektējuma vienums	**42,1**
	e	Formālā prasība	**36,8**
	e	Sākotnējās, neformālās prasības	10,5
	e	Koda elements	5,3
	e	Tests	5,3
	e	Modelis	5,3
	n	Piegāde	5,3
	e	Biznesa funkcija	5,3
	e	Apakšsistēmas elements	5,3
	e	Uzdevums projekta plānā	5,3

Izmaiņu pieprasījuma tipiskākie bērni ir projektējuma vienums un formālās prasības. Var secināt, ka tipiskas situācijas gadījumā izmaiņu pieprasījums nav atkarīgs no kādiem citiem vienumiem, bet gan tas pats kalpo par avotu jauniem vienumiem vai izmaiņām.

16. tabula.

Sākotnējās, neformālās prasības

Virziens	Saite	Vienuma tips	Biežums (%)
Vecāki	e	Izmaiņu pieprasījums	13,3
	e	Likumdošana	6,7
Bērni	e	Formālā prasība	**66,7**
	e	Lietojumgadījums	13,3
	e	Projektējuma vienums	6,7
	e	Testpiemērs	6,7
	n	Līgums	6,7
	e	Piedāvātais risinājums	6,7

Sākotnējo prasību tipiskākie bērni ir formālās prasības. Šis vienums ir tipiska pre-RS trasējamības sastāvdaļa, un parasti ir būtiski reģistrēt no dažādiem avotiem ienākošās prasības, jo uz to pamata tiek veidotas formalizētas prasības. Tikai atsevišķos gadījumos no neformālām prasībām var tikt atvasināti testi, projektējums vai citi vienumi. Ņemot vērā to, ka 46,9% no aptaujātajiem projektiem tika uzturēta informācija par sākotnējām jeb neformālajām prasībām, var secināt, ka pre-RS trasējamība tiek samērā plaši praktizēta. Šīs aptaujas ietvaros netika izpētīts, kāda tieši informācija tika uzturēta par šiem vienumiem un saitēm.

17. tabula.

Tests

Virziens	Saite	Vienuma tips	Biežums (%)
Vecāki	e	Formālā prasība	**50,0**
	e	Projektējuma vienums	**41,7**
	e	Testu komplekts	**25,0**
	e	Problēmziņojums	8,3
	e	Izmaiņu pieprasījums	8,3
	e	Modelis	8,3
Bērni	n	Problēmziņojums	**58,3**
	e	Testpiemērs	**41,7**
	n	Testžurnāls	16,7

Testa tipiskākie vecāki ir formālā prasība, projektējuma vienums un testu komplekts. Tipiskākie bērni ir problēmziņojums un testpiemērs. Kā bija minēts 5.4.nodaļā, testa un testpiemēra jēdziens bieži tiek lietots līdzīgās nozīmēs, lai gan testu parasti var uztvert kā testpiemēru apvienojumu. Ne mazums projektos testēšana nav pietiekami labi detalizēta, lai būtu minētā divu līmeņu struktūra – testi ir aprakstīti mērķu līmenī, ir norāde uz prasībām, bet nekas precīzāk nav aprakstīts. Tests balstās uz tiem informācijas avotiem, kas nosaka programmas saturu. No testa ir atkarīga vai nu tālāka tā detalizācija, vai arī pieraksti, kas saistās ar tā izpildes rezultātiem.

18. tabula.

Testpiemērs			
Virziens	Saite	Vienuma tips	Biežums (%)
Vecāki	e	Tests	**41,7**
	e	Formālā prasība	**25,0**
	e	Koda elements	16,7
	n	Testu komplekts	16,7
	e	Projektējuma vienums	8,3
	e	Sākotnējās, neformālās prasības	8,3
	e	Biznesa funkcija	8,3
	e	Testa plāns	8,3
Bērni	n	Problēmziņojums	**66,7**
	n	Testžurnāls	**66,7**
	e	Testa plāns	16,7

Testpiemēra tipiskākie vecāki ir tests un formālā prasība. Bērni – problēmziņojums un testžurnāls. Testpiemērs var tikt veidots kā testa atvasinājums, vai būt neatkarīgi balstīts uz prasībām. Līdzīgi kā testam, arī uz testpiemēru norāda testēšanas rezultātā iegūtie pieraksti.

19. tabula.

Testžurnāls			
Virziens	Saite	Vienuma tips	Biežums (%)
Vecāki	n	Testpiemērs	**72,7**
	n	Problēmziņojums	**27,3**
	n	Tests	18,2
	n	Koda elements	9,1
	n	Testa plāns	9,1
Bērni	e	Testēšanas pārskats	18,2
	n	Problēmziņojums	9,1

Testžurnāla funkcija ir reģistrēt testēšanas rezultātus. Testžurnāla tipiskākie vecāki ir testpiemērs un problēmziņojums, un tikai atsevišķos gadījumos uz to atsaucas kādi projekta vienumi. Saitei starp problēmziņojumu un testžurnālu ir uztverama kā simetriska saite – vispārējā gadījumā nav būtiskas nozīmes virzienam – tas ir atkarīgs no konkrētā projekta metodikas.

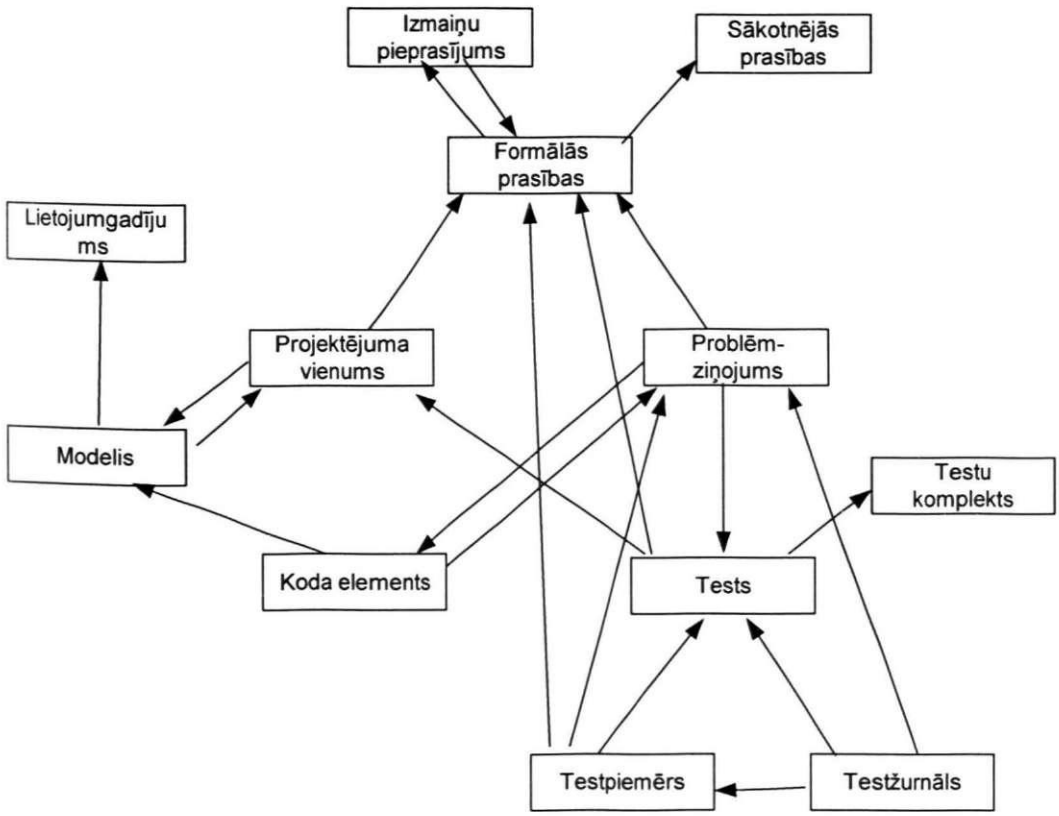
20. tabula.

Modelis			
Virziens	Saite	Vienuma tips	Biežums (%)
Vecāki	e	Projektējuma vienums	**25,0**
	e	Lietojumgadījums	**25,0**
	e	Koda elements	12,5

	e	Formālā prasība	12,5
	e	Modelis	12,5
	e	Izmaiņu pieprasījums	12,5
Bērni	e	Koda elements	**37,5**
	e	Projektējuma vienums	**25,0**
	e	Tests	12,5
	e	Modelis	12,5
	n	Piegāde	12,5
	e	Biznesa funkcija	12,5

Projektā modelis var būt robežās no prasībām līdz projektējumam – tas ir atkarīgs no projektā izmantotās metodikas. Modeļa tipiskie vecāki ir projektējuma vienums un lietojumgadījums, bet bērni – koda elements un projektējuma vienums. Modelis var būt saistīts arī ar citu modeli.

Apvienojot vienā modelī visas tipiskākās saites starp vienumiem, var iegūt nosacītu summāro trasējamības modeli (10.attēls), kas raksturo tipiskāko veidu, kā savstarpēji projektos ir saistīti vienumi. Šis modelis ir uztverams tikai kā viens skatījums, lai sniegtu priekšstatu par trasējamības veidiem projektos. Ņemot vērā to, ka tika iekļautas tikai tipiskākās saites, nav ietverti vairāki retāk sastopami vienumu tipi. No otras puses, katrā konkrētā projektā ir tikai daļa no minētajām saitēm.



10.attēls. Trasējamības modelis ar projekta vienumu tipiskākajām saitēm.

Ja konkrēti aplūkojam vienumus, kas attiecas uz testēšanas procesu, tad apvienotajā modelī ir iekļauti pieci: tests, testpiemērs, problēmziņojums, testžurnāls

un testu komplekts. Šo vienumu veidotais modelis atšķiras no 5.4.nodaļā parādītā vispārinātā testēšanas procesa trasējamības modeļa. Atšķirības nav principiālas. Pie tipiskajiem vienumiem nav tādu vienumu, kā testu klase, testu komplekts un testu konfigurācija. Ir variācijas ar notikumu saitēm, kurās ir iesaistīts testžurnāls un problēmziņojums: var atšķirties saišu virzieni un vienumi, ar kuriem tie ir saistīti. Lietas būtību tas nemaina, jo projektam tiek nodrošināta iespēja izsekot to saistībai un atkarībām.

Atgriežoties pie iepriekš minētajiem citu autoru pētījumiem ([59], [30], [123]) par trasējamības situāciju, var konstatēt, ka Latvijas IT projektos pastāvošās trasējamības problēmas ir līdzīgas: minimāla rīku izmantošana trasējamībai saistīta ar to, ka pastāv priekšstati, ka tas var prasīt papildus darbu. Tiešām, ir trasējamības risinājumi, kas prasa lieku darbu. Projektiem nav skaidrs, cik detalizēti uzturēt trasējamības informāciju. Ir liela daļa projekta informācijas, ko darbinieki tik un tā zina, bet, savukārt, no fragmentāra pieraksta nav lielas jēgas. Rezultātā nav nekādu pierakstu.

No aptaujas var secināt, ka trasējamības nodrošināšanai ir nepieciešams rīku atbalsts. Pretējā gadījumā tā būs nepilnīga un grūti pārvaldāma. Arī aptauja parāda, ka projektos, kur tika lietoti rīki, bija lielāka apjoma trasējamības modeļi nekā tajos, kur nebija izmantoti rīki (vidēji 8,6 vienumu tipi un 10 saišu tipi projektos ar rīkiem pret 5 vienumu tipiem un 4,8 saišu tipiem projektos, kuros netika izmantots trasējamības rīks). Otrs secinājums saistās ar to, ka trasējamību nav nepieciešams afišēt kā kaut ko īpašu un atdalītu no projekta darbiem. Tai ir jābūt integrētai projekta procesos tā, lai tā palīdz veikt primāros uzdevumus, bet nepatērētu resursus sevis uzturēšanai.

Aptaujas pētījumu būtu noderīgi detalizēt divos virzienos:

- padziļināta situācijas un procesu izpēte gadījumos, kur pret trasējamību bija vērojama atturīga nostāja;
- papildus pētījumi par rīku izmantošanas veidiem - saņemto atbalstu un novērojamajām problēmām.

Šādu pētījumu būtu nozīme veikt šobrīd darbojošos projektos (jo ir svarīgi redzēt, kā tiek izmantota un uzturēta ar trasējamību saistītā informācija). Tas nozīmē, ka vairākos gadījumos nebūtu iespējams izmantot šeit aprakstītajā aptaujā apzinātos projektus. Ņemot vērā to, ka līdzšinējais pētījums prasīja darbu aptuveni viena cilvēkmēneša apjomā, piedāvātie tālākie pētījumi prasītu lielāku darba apjomu, un tā rezultātiem vajadzētu būt ne tikai ar akadēmisku vērtību, bet jāsniedz projektu praktiskajam darbam noderīgi secinājumi.

5.6. Nodaļas secinājumi

Šajā nodaļā tika aplūkotas trasējamības problēmas konkrētās programmatūras projektu situācijās. Vairākos jautājumos tiek pretstatīta industriālo projektu prakse standartos izvirzītajām teorētiskajām prasībām. Galvenie iemesli, kādēļ projekti

izvēlas risinājumus, kas nepilnīgi apmierina tādu trasējamības pilnības prasību, ka ikvienai projektā esošajai ar produktu saistītai informācijai ir jābūt sasaistāmai ar citu informāciju (t.i. starp jebkuriem diviem projekta vienumiem ir jābūt tiešai vai pastarpinātai saitei), ir saistīti ar nepietiekamie projekta kvalitātes kritērijiem un ierobežotām tehniskajām iespējām. Īpaši tika analizēta trasējamība testēšanas procesa ietvaros, kur ir novērojamas tipveida saites starp testēšanas procesā iesaistītajiem projekta vienumiem. Lai arī katrā projektā testēšanas process var atšķirties, ir minimālā informācijas kopa (var būt ietverta atšķirīgos projekta vienumu tipos), kas raksturo testēšanas procesu – plānotie testi un izpildes laikā iegūtie rezultāti.

No reālos Latvijas IT kompāniju projektos iegūtās informācijas var secināt, ka nepietiekami liela daļa projektu pievērš uzmanību trasējamībai. Lai arī daudzos gadījumos tomēr tiek uzturēta trasējamības informācija, būtiskas problēmas ir ar tehnisko risinājumu – informācija ir nepilnīga, tā ne vienmēr ir aktuāla un ir jāvelta īpašs darbs tās uzturēšanai, kas reālos projektus komerciālos apstākļos attur pievērst atbilstošu uzmanību tādam kvalitātes aspektam kā trasējamībai. Rezultātu salīdzinošs novērtējums ar citiem pētījumiem par trasējamības praksi projektos parāda līdzīgas problēmas, un ļauj secināt par pētījumu šādos virzienos:

- trasējamības atbalsts ar rīku palīdzību,
- trasējamības īpašības integrēšana programmatūras procesos.

Šajā nodaļā tika pamatota darbā izvirzītā apgalvojuma (a) daļa - testēšanas process ir atkarīgs no trasējamības kvalitātes.

6. Jauna rīka pamatnostādnes

6.1. Nodaļas mērķi

Nodaļā ir aprakstīti apsvērumi, kas tika ņemti vērā, uzsākot darbu pie trasējamības rīka TraceIt izstrādes. Lai arī iepriekš tika aplūkots, ka eksistē rīki, kas nodrošina trasējamību, šeit parādītas reālo projektu prasības, kas liecināja par tāda risinājuma nepieciešamību, kurš nebūtu pretrunā ar projekta pamatuzdevumu, tradīcijām un uzņēmuma metodiku. Jaunajam rīkam netika plānoti ierobežojumi attiecībā uz programmatūras projekta procesiem. Prasības pēc trasējamības atbalsta rīka formā pamazām evolucionēja, un vairāku mēnešu laikā parādījās iespēja sākt darbus pie eksperimentāla rīka izstrādes.

Kā redzams no pētījuma par IT uzņēmumu praksi, trasējamībai ir nepieciešams atbalsts, un acīmredzot eksistējošie komerciālie rīki nav bijuši piemēroti, vai arī nav radusies nepieciešamība pēc tiem. Tādēļ autors uzskata, ka jauns trasējamības rīks varētu sniegt sagaidāmo atbalstu, vai arī vismaz sniegt projektiem pirmo uzskatāmo priekšstatu par trasējamības praktisko pusi – noderīgumu un saistītajām projekta aktivitātēm.

TraceIt pamatā ir dažas prasības pret informācijas saglabāšanas formu – katram vienumam tiek piešķirts unikāls identifikators, vienumi tiek grupēti pēc to tipa, starp vienumiem var nodibināt iepriekš definēta tipa saites saskaņā ar trasējamības modeli. Nodaļā iepriekšminētie jautājumi ir aplūkoti detalizētāk kopā ar projektā veicamo darbu mērķiem. TraceIt konceptuālās nostādnes ir atspoguļotas rakstā [6].

6.2. Jauna rīka ieviešanas motivācija

Laikā, kad autors bija vairāku gadu garumā nostrādājis RITI Kvalitātes nodrošināšanas un testēšanas laboratorijā, uzņēmumā tika nodibināta Procesa uzlabošanas grupa, un autors tika iekļauts tās sastāvā. Šīs grupas darba mērķi bija analizēt projektu pieredzi un nozares standartu prasības, lai IT uzņēmumā sagatavotu principus, kā programmatūras projektiem kopā ar pamatmērķa izpildi nodrošināt kvalitatīvu pieeju darbam – tas uzlabo gan produkta kvalitāti, gan arī padara veicamo darbu pārskatāmāku. No vairāku gadu attāluma var labi redzēt, ka kvalitātes sistēmas ieviešanas pirmsākumos tā pamatā kalpoja diviem mērķiem – (1) lai uzņēmums varētu pats sev un citiem apliecināt, ka kvalitāte veicamajam darbam ir un ka uzņēmums ir gatavs jebkuram eksāmenam; (2) lai uzņēmumam būtu dokuments, kas paaugstina tā konkurētspēju Latvijas un starptautiskā programmatūras tirgū. Par kvalitātes sistēmas reālajām sekām katram procesā iesaistītajam bija savas domas. Ja šāds process ir kaut kas jauns, tad ir svarīgi, lai katrs tā dalībnieks saskatītu ieguvumus sava darba uzlabošanai. Pretējā situācijā vadība un tās nozīmētās personas strādā pie kvalitātes sistēmas ieviešanas, bet pārējiem šis process šķiet vienaldzīgs vai pat naidīgs – kādēļ kādam no malas būtu jāiejaucas projekta darbos, ja šie cilvēki no malas nemaz

nepārzina projekta specifiku. Sākotnēji piekoptā burtiskā sekošana kvalitātes standarta prasībām izraisīja lielu skaitu jaunu dokumentu rašanos, kas ne viena vien darbinieka apziņā ieviesa asociāciju par kvalitāti kā par birokrātiju. Šajā kontekstā Procesa uzlabošanas grupas uzdevums bija izanalizēt pēc būtības, kas ir nepieciešams projektiem, lai bez lieka darba izpildītu standartos minētās prasības, kā arī uzlabotu sava darba kvalitāti.

Pamata problēma izrietēja no tā, ka praksē ir grūti ievērot visas standartu prasības. Vienas un tās pašas prasības var realizēt atšķirīgos veidos, un neveiksmīgs risinājums ātri kļūst par apgrūtinājumu projekta mērķa sasniegšanai. Ja veidojas tāda situācija, tad pilnīgi noteikti ir jāmeklē citi realizēšanas varianti vai arī no minētās prasības ieviešanas jāatsakās vispār. Tiesa, šādi mēs riskējam atkāpties no standarta.

Viena no neatbilstībām starp standarta prasībām un to, ko praksē sagaidītu projekti, ir trasējamība. Standartos, tādos, kā J-STD-016 ir minēta prasība par trasējamības eksistenci, un arī dokumentu sagatavēs ir norādīta vieta trasējamības tabulai. Ja nav paskaidrojošu materiālu un atbalstošu līdzekļu, minētā prasība tiek uztverta kā lieka. Programmatūras projekti vadās pēc principa: „kā palīdzēs man sasniegt mērķi tas, ko no manis prasa standarts vai kvalitātes sistēma?". Pasūtītājs tiešā veidā nemaksā par kvalitātes aktivitātēm. Samaksa ir par konkrēti iegūstamo rezultātu – specifikāciju, darbināmu programmatūru u.tml. Kvalitātes aktivitātēm ir jābūt daļai no pamatdarba, tām nav jātraucē, bet jāatbalsta pamatmērķa sasniegšana.

Autors pievērsās trasējamības tēmai, balstoties uz vairāku gadu apjomā iegūto pieredzi reālos testēšanas projektos. Tieši vadoties no iepriekšminētajiem projektu praktiskajiem apsvērumiem, autors saskatīja nepieciešamību uzlabot trasējamības īpašību projektos ar kāda atbalstoša mehānisma palīdzību. Ideja bija izveidot tādu projekta ietvaros izmantojamu datubāzi, kur glabātos kaut kas līdzīgs projekta dokumentu satura rādītājiem. Tajā būtu iespēja atrast, ko satur dokumenti, kā arī tajos aprakstīto objektu savstarpējo saistību – t.i. saturētu informāciju par saitēm starp vienumiem. Pirmie priekšstati bija aptuveni, un pirms tālākas idejas attīšanas tika veikti iekšēji neformāli pētījumi par uzņēmumā pielietotajiem tradicionālajiem risinājumiem trasējamības saišu uzturēšanai. Tika konstatēti šādi risinājumi:

- Dokumentiem, prasībām, funkcijām un testiem ir vienota stila vairāku pozīciju identifikatora forma, kas, ļauj izmantot atsevišķas pozīcijas saistības raksturošanai;

- Tiek uzturētas īpašas tabulas biroja programmatūras dokumentu formā;

- Vairāki dokumenti un dokumentu daļas savstarpēji tiek sasaistītas ar hipersaitēm.

Ne vienā vien gadījumā vienkāršās metodes apmierināja projekta vajadzības. Tiesa, tipiskā veidā tas der tikai nelieliem projektiem - vienumu un saišu skaits nav pārāk liels. Otra lieta ir tā, ka ne visi projekti bija sev izvirzījuši kvalitatīvus jautājumus par implementācijas vai testēšanas pilnību, izmaiņu pārvaldību u.c. Salīdzinošs novērtējums pa programmatūras procesiem liecināja, ka lielākā daļa izstrādes projektu ir maksimālu uzmanību veltījuši projektēšanai un implementācijai.

Arī gadījumos, kad formālā dokumentācija bija nepilnīga (pārsvarā neliela izmēra projektos), sistēmanalītiķi un programmētāji lielu daļu nianšu pārzināja tik labi, ka dokumentālas trasējamības informācija uzturēšana nebija nepieciešama. Tomēr tajā brīdī, kad projektā iesaistījās testētāji (parasti tas notiek implementācijas laikā vai līdz ar implementācijas noslēgumu), viņiem parādās vitāli svarīga nepieciešamība būt lietas kursā par prasību, funkciju un programmas ekrānu savstarpējām atkarībām, ir nepieciešams gūt pārliecību par testēšanas pārklājumu un ir jāspēj izsekot problēmu risināšanas gaitai.

Jāsecina, ka motivācija pēc jauna rīka izveides nāca pamatā no testēšanas un no tā, ka projektā parādās vairāku tipu intereses – implementēt un testēt. Vienai personu grupai ir labas zināšanas, pārzinot programmu, bet viņi nespēj tās efektīvi tās nodot. Ja būtu pieraksti, kurus kolēģi varētu analizēt, tad komanda nedraud būt sašķelta. Lai arī ir zināms, ka jaunu metožu pielietošana parasti var izraisīt traucējumus, tika nolemts veikt eksperimentu tā, lai tas pēc iespējas minimāli mainītu no projekta puses ieguldāmo darba apjomu. Darbi pie jauna rīka pamatiem tika uzsākti hronoloģiski agrāk nekā tika veikts pētījums par trasējamības praksi Latvijas IT projektos. Iepriekšminētais pētījums bija kā papildus apstiprinājums rīka izstrādes nepieciešamībai.

Ja aplūkojam tādu potenciālo rīka lietošanas aspektu, kā izmaiņu vadība, tad būtisks elements ir informācijas apmaiņas par programmā konstatētajām problēmām. Praktiski ikvienā projektā bija kāda metode problēmziņojumu reģistrēšanai un apstrādei. Tehniskie risinājumi ietvēra e-pastu, MS Excel failus, rīku PVCS Tracker [106]. E-pasta apmaiņa nodrošina labu informācijas apriti, bet tā nenodrošina problēmu statusa izsekošanu – ja arī sūta kopijas visiem darbiniekiem pēc katra labojuma vai papildinājuma, pazūd iespēja pārskatāmā veidā noskaidrot aktuālo problēmu sarakstu, problēmu sadalījumu pēc to prioritātes utt. MS Excel risinājuma būtiskākais mīnuss ir apgrūtinātais darbs daudzu lietotāju gadījumā – kā vislabāk sinhronizēt informāciju, ja vairāki cilvēki piesaka problēmas un vairāki veic programmā labojumus (t.i. aktualizē ierakstus). Vairākos projektos bija izstrādātas metodes, kas balstījās uz administratīvi noteiktu projekta darbinieku darbu izpildes kārtību. PVCS Tracker vislabāk atbilda komandas darbam, bet no praktiskā lietojuma viedokļa bija novērojamas gan veiktspējas problēmas, gan arī ierobežotās iespējas darbā ar ziņojumiem. Tas bija orientēts tikai uz problēmu apstrādi, un nebija iespēju, piemēram, apstrādāt arī testu izpildes rezultātus un sasaistīt tos ar problēmziņojumiem.

6.3. Darbību veidu struktūra

Autors ir novērojis, ka reālos programmatūras projektos darbu intensitāte nav sadalīta vienmērīgi. Parasti sākuma etaps ir relatīvi mierīgs – notiek situācijas izpēte, plānošana, pirmie darbi un eksperimenti ar pieejām. Savukārt, vidus posms un

noslēguma etapi parasti tiek pavadīti ar maksimālu darba noslodzi. Nereti ir gadījumi, kad darbiniekiem rodas nepieciešamība strādāt virsstundas, kā arī projektam nobīdās izpildes termiņi. Nemēģināsim analizēt visus iemeslus, bet ir vērts tikai minēt, ka daļēji laika trūkuma problēmas saistās ar projekta sākumā novērtā darba apjoma neprecizitāti - parasti novērtējums ir par zemu, jo intuitīvi viss sākumā ir šķitis daudz vieglāk paveicams. Nobīdītie termiņi visbiežāk sevī ietver papildus iterācijas ciklā implementēšana-testēšana, vai arī projektēšana-implementēšana-testēšana. Starp šiem procesiem informācijai ir jābūt sinhronizētai – trasējamai starp dažādiem projekta informācijas tipiem – tas ir būtisks kvalitātes faktos. Papildus nosacījums ir informācijas apmaiņas operativitāte. Projektam interesē, lai iterāciju cikls būtu īsāks, un to skaits būtu mazāks.

Iepriekšminēto iemeslu dēļ, aplūkojot iespējamos jauna rīka uzdevumus un ieviešanas variantus, parādās praktiskas dabas nosacījums, ka jaunais rīks nedrīkst prasīt no lietotājiem papildus darbu salīdzinājumā ar eksistējošo stāvokli (teksta dokumenti, e-pasts, elektroniskās tabulas). Ja izdodas panākt darba apjomu samazināšanos, tad rezultāts būtu īpaši pozitīvs. Ņemot vērā to, ka trasējamība tomēr ir informācijas satura kvalitātes īpašība, tad īpaši šim nolūkam uzturēt kādu atsevišķu informācijas kopu nav lietderīgi. Ir jāpievieno trasējamības iespējas pie eksistējošajiem risinājumiem. Ja tādu nav, vai arī tie nepakļautos papildināšanai, ir jāveido jauna vieta trasējamības informācijas un kādas daļas no projekta informācijas uzturēšanai. Piemēram, cikla implementācija-testēšana gadījumā ar jauno rīku būtu jāspēj uzkrāt un apstrādāt: informāciju par kodu, testus, testu izpildes rezultātus, problēmziņojumus. Ja eksistē risinājums, kā tas tiek veikts šobrīd, jaunajam risinājumam ir jāspēj piedāvāt vismaz tikpat lielas iespējas.

Katrs rīks prasa tā sagatavošanu un konfigurēšanu. Projektā katram darbiniekam ir sava loma, un atbilstoši šai lomai nepieciešams sadalīt rīka konfigurēšanu un informācijas uzturēšanu. Aplūkojot, jaunu trasējamības rīku, darbs ar to varētu būt iedalāms vairākās grupās:

- administrēšana (sistēmas parametru un lietotāju definēšana),
- tipiskie ikdienas darbi, kas ietver darbus ar vienumu pamatdatiem un vienumu saitēm,
- izmaiņu apskates aktivitātes un
- informācijas importēšanas un eksportēšanas uzdevumi.

Administrēšanas funkciju var veikt projekta pārvaldnieks vai viņa pilnvarota persona. Savukārt, izmaiņu apskates, importēšanas un eksportēšanas funkcijas var veikt sistēmanalītiķi vai citi darbinieki. Ikdienas darbi ar vienumu informāciju būtu veicami ikvienam projekta darbiniekam.

6.4. Vienumi

Jaunā trasējamības rīka kontekstā vienums ir nodalāma projekta informācija, kas saturiski projekta vai procesa ietvaros veic kādu iepriekš apzinātu tipveida uzdevumu. Vienumu jāspēj aprakstīt ar strukturētas tekstuālas informācijas palīdzību. Ja pats vienums pēc būtības sastāv no strukturētas tekstuālas informācijas, tad rīkā ir uzturams ne tikai vienuma apraksts, bet arī pats vienums. Piemēram, prasības visbiežāk ir cilvēka valodā formulēts teksts, un viena prasība parasti tiek aprakstīta ar vienu vai dažiem teikumiem. Projekta vienumi ir jādefinē tādi, kas reāli projektam ir nepieciešami. Tipisks piemērs ir iepriekš aprakstītajā IT projektu aptaujā minētie vienumu tipi. Rīkā netiek veidoti ierobežojumi attiecībā pret nosaukuma formu.

Ikvienam vienumam ir vairākas raksturojošās īpašības:

- izcelsme,
- lietotāju loks,
- pielietojums,
- atkarība no citiem.

Rīks tiešā veidā neoperē ar iepriekšminētajām īpašībām, bet netieši tās tiek uzturētas. Izcelsme un pielietojums ir atkarīga no piederības procesam, kuros tiek izmantots rīks. Lietotāju loks tiek noteikts ar lietotāju lomu palīdzību – katram lietotājam ir viena vai vairākas lomas, bet katrai lomai – individuālās tiesības, ko lietotājs var redzēt un ar ko var operēt. Vienuma atkarība no citiem raksturojama ar tam esošajām saitēm. Vispārējā gadījumā vienumam var būt viens vai vairāki vecāki, kas nosaka tā izmaiņu nosacījumus. Pats vienums var atrasties vairākos stāvokļos. Sākumā tas ir definēts. Pēc tam to var sasaistīt ar citiem vienumiem vai labot. Ja vienums ir ar saitēm, tad to var atsaistīt. Vienumu pārstāj izmantot projekta informācijas apritē, ja tas tiek dzēsts.

Viena no informācijas kvalitātes īpašībām ir tā, ka konkrētā projekta ietvaros katrs vienums ir viennozīmīgi atšķirams no cita vienuma. Šajā nolūkā rīkā ir iestrādāts princips, ka katram vienumam ir tiek piešķirts automātiski (vai arī to norāda lietotājs) unikāls identifikators.

Vienuma tipa definīciju var mainīt projekta norises laikā. Tas nozīmē, ka sākotnēji, piemēram, testus, var uzkrāt tikai identifikatora un nosaukuma līmenī, norādot tā saistību ar citiem vienumiem, bet projekta vēlākā gaitā to var papildināt ar klasificējoša tipa pazīmēm un konkrēto saturisko informāciju. Tas notiek, papildinot vienuma definīciju ar papildus laukiem. Līdzīgi var arī likvidēt nevajadzīgos laukus. To var darīt gadījumos, ja projektam nav nepieciešamības un iespēju uzkrāt atbilstoša tipa informāciju. Ja tomēr informācija ir pieejama, tad nebūtu ieteicams dzēst uzkrātos datus, jo tie projekta vēlākā gaitā var noderēt.

Vienums (konkrēta identificēta prasība, funkcija, tests, problēmziņojums) ir savstarpēji saistīts ar citu - kādas ir atkarības un kādas varētu būt iespējamo izmaiņu

sekas. Raksturīgi ir tas, ka parasti atkarība starp projekta vienumu tipiem ir orientēta, t.i. tā iet vienā virzienā, bet ne otrādi, kā arī saitēm ir noteikts tips.

6.5. Saites

Ir lietderīgi aplūkot, kā veidojas trasējamības saites. Ja izslēdzam triviālos gadījumus, kad objekts var būt saistīts pats ar sevi, saiti var nodefinēt jau tajā brīdī, kad eksistē vismaz divi objekti. Jautājums ir par saišu nozīmi un lietderību. No praktiskiem apsvērumiem nebūtu nozīmes uzturēt saites, kas nedod praktisku labumu kvalitātes uzlabošanai un darba efektivitātes palielināšanai. Ir saites, kas sasaista viena objekta dažādas versijas to hronoloģiskā attīstībā. Tipiski tā ir savstarpēji saistītu objektu virkne, bet vispārīgā gadījumā var veidoties versiju koks – ja kāda versija tiek tālāk attīstīta vairākos atšķirīgos veidos.

Otra veida saites ir tās, kas parāda, ka objekts ir veidots, lai atbilstu minētajai prasībai, tā ir tā pati funkcija, bet implementēta programmēšanas valodā. Tas raksturo evolūciju, kad prasība vai ideja tiek pārcelta un transformēta no viena tipa objekta uz citu. Šādu saišu gadījumā būtiska ir saistība, ka viens objekts parasti ir atkarīgs no otra. Ja izmaiņas notiek vienā, tad tām būtu jāizpaužas arī otrā.

Trešā veida saites ir tādas, ka kalpo kā informācija par to, ka divi objekti kaut kā ir saistīti, bet tas nenozīmē, ka viena objekta mainīšana iespaido otru.

Katrai saitei var būt vēsturiskais konteksts – kad veidojusies, vai ir spēkā un cik ilgi ir bijusi spēkā. Saitēm ir noderīgi definēt tipus tā, ka starp noteiktiem vienumu tipiem var eksistēt tikai noteikta tipa saites. No lietotāja viedokļa saites parasti ērti ir nosaukt kādos darbības vārdos „atvasināts no", „pieder pie", „ir daļa no".

Starp projekta vienumiem rīkā atbilstoši 3.3.nodaļā sniegtajai definīcijai tiek uzturētas evolucionārās un notikumu saites. Rīka ietvaros evolucionārās saites tiek sauktas par stingrajām saitēm, notikumu saites – par nestingrajām.

Katrai saitei ir virziens. Tas atbilst vienumu veidošanās secībai vai hierarhijai. Ņemot vērā to, ka vēlāk veidojies vienums atsaucas uz agrāk eksistējošu vienumu, saites virziens tiek veidots kā atsauces virziens – no jaunākā vienuma uz vecāko.

Saišu virziena kontekstā katram vienumam saites var iedalīt divās grupās:

- „uz" saites – no vienuma izejošās saites un
- „no" saites – vienumā ienākošās saites.

Ņemot vērā to, ka attiecībā pret vienumu atkarībā no saišu virziena veidojas divas saistīto vienumu grupas, tām tiek doti nosaukumi „vecāki" (ar „uz" saitēm saistītie) un „bērni" (ar „no" saitēm saistītie).

Nav noteikti ierobežojumi tam, vai vienumam ir jābūt obligāti ar citiem piesaistītiem vienumiem. Rīks neizvirza nosacījumus pret to, ar cik citiem vienumiem var eksistēt saites. Notikumu saitēm netiek ierobežots to eksistences ilgums. Pēc notikuma nosacītām beigām saite netiek dzēsta, ja vien netiek dzēsts viens no iesaistītajiem vienumiem.

Lai arī rīkā iestrādātajos principos ir definēts, ka netiek aplūkotas tādas saites, kas ir simetriskas un saista divus vai vairāk vienumus pēc kādām nestingri noteiktām pazīmēm, sistēma neaizliedz lietotājam definēt nestingru saiti ar atbilstošu nosaukumu un izmantošanas mērķi.

Saitēm var mainīt vārdu un piezīmju informāciju, bet nav iespējams mainīt tipu. Ja saiti atsaista vai maina jebkuru no diviem tās vienumiem, tad saite faktiski tiek dzēsta. Tās vietā var definēt jaunu. Ja dzēš vienu no vienumiem, pie kā pieder saite, tad tiek dzēsta arī pati saite. Ja tiek dzēsts vienums, kuram ir vairākas saites, tad tiek dzēstas visas šīs saites.

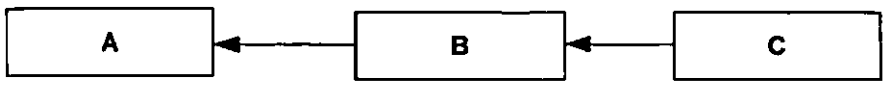
6.6. Trasējamības modelis

Trasējamības modelis ir metamodelis visām novērojamām saitēm starp projekta vienumiem. Visērtākais tā attēlošanas veids ir grafiskas diagrammas veidā - vienumu tipus apzīmē ar taisnstūriem, bet saites ar bultām. Šī notācija atbilst ierobežotai un pielāgotai UML klašu diagrammu notācijai. Saišu bulta ir orientēta virzienā no bērna uz vecāku.

Trasējamības modelim ir jāatspoguļo projektā plānotā informācijas struktūra, un projekta norises laikā pie šīs struktūras ir jāpieturas. Rīka uzdevums ir nepieļaut saišu veidošanu, kuras neatbilst trasējamības modelim. Saišu esamība konkrētam vienumam nav obligāta. Rīks pieļauj situāciju, ka tiek nodefinēts vienums, un tas tiek sasaistīts ar vecāku (ja to prasa konteksts) vēlāk. Starp jebkuriem diviem vienumiem var eksistēt ne vairāk kā viena tiešā saite (katram saites tipam). Sasaistes kontrolei lietotājs var izmantot rīka līdzekļus. Ja projekta norises laikā atklājas, ka ir jāuztur papildus vienumi un saites, rīks nodrošina iespēju modeli modificēt. Pēc modifikācijas vecā modeļa variants vairs netiek atbalstīts (iepriekš definētie vecie vienumi un saites pēc apstiprināšanas tiek likvidētas). Rīka ietvaros metamodeļa definēšana ir formulēta kā pieļaujamo saišu noteikšana starp definētajiem vienumu tipiem.

Projekta norises gaitā trasējamības modelis var mainīties. Šādā situācijā var tikt veidoti jauni vienumu tipi un jaunas pieļaujamās saites. Īpašs gadījums ir pieļautās saites vai vienumu tipa dzēšana. Šajā gadījumā būtu jādzēš arī visas līdz šim izveidotās attiecīgās saites, bet vienuma tipa gadījumā – visi atbilstošie vienumi. Implementācijā ir iestrādāts drošības mehānisms, kas neļauj lietotājam dzēst vienumu tipu vienlaikus ar visiem ierakstiem, kā arī pieļautās saites dzēšanas gadījumā tiek atstātas iepriekš nodefinētās saites. Tālākā saišu likvidēšana jāveic konkrētajām saitēm individuāli.

Ar trasējamības modeļa starpniecību var uzskatāmi novērot ne tikai tiešās, bet arī saites, kas veidotas pastarpināti – caur citiem vienumiem. Kā jau bija minēts iepriekš, stingro saišu gadījumā eksistē transitīva īpašība – ja vienums B ir atkarīgs no A, bet C no B, tad C ir atkarīgs no A (11.attēls). Tiesa, ne katra izmaiņa vecākā nozīmē izmaiņu tā bērnos.



11.attēls

Autora iecerētais trasējamības modeļa princips ir tāds, ka katram projekta vienuma tipam ir jābūt saistītam ir kādu citu projekta vienumu tipu. Tajā pašā laikā ir pieļaujams, ka eksistē konkrētie vienumi, kuriem nav saites ar citu vienumu (tādi ir vecāki tajā brīdī, kad tie ir tikko definēti). Bērni nedrīkst būt bez saites ar vecāku. Tikai pagaidu variantā tie varētu būt bez saitēm. Tomēr varētu būt arī projekta situācijas, kad trasējamības modelī apzināti kāds vienumu tips netiek sasaistīts ar citu vienumu tipu. Šādā gadījumā, protams, trasējamība netiks nodrošināta starp šo individuālo tipu un pārējiem tipiem, tomēr, iespējams, rīka lietotāji ir iecerējuši vēlāk šo trasējamības modeli mainīt, vai arī vēlas vienkārši šo rīku izmantot kā parocīgu veidu informācijas uzkrāšanai un apstrādei.

Trasējamības modelī ir pieļaujama situācija, kad vienuma tipam ir saite pašam ar sevi. Šāda tipa saites ir iespējams izmantot vienumu hierarhijas organizēšanai. Trasējamības modelī ir pieļaujama arī cikla veidošanās (izsekojot līdzi saišu virzieniem). Tomēr jāņem vērā, ka no praktiskā viedokļa cikliska informācija projektā nevar veidoties. Visticamāk šāda situācija var būt nekorektas informācijas reģistrācijas rezultāts. Cikls var eksistēt vienumu tipu līmenī, bet konkrētie individuālie vienumi būs atšķirīgi.

Viens no noderīga trasējamības modeļa virsuzdevumiem ir tas, ka tam būtu jābūt atkārtoti izmantojamam citos projektos.

6.7. Izmaiņu pārbaude

Individuālās saites un koks

Lai rīks atbalstītu lietotāju trasējamības analīzes veikšanā, lietotājam tiek piedāvātas vairākas iespējas, kā aplūkot vienumu saišu informāciju. Pamata iespēja ir aplūkot katram vienumam individuālas saites ar citiem vienumiem. Šīs saites uztveramības nolūkos tiek grupētas pēc virziena (vienuma vecāki un bērni), un katra virziena ietvaros piesaistītie vienumi ir sagrupēti pa tipiem. Katram piesaistītajam vienumam var aplūkot saites tipu, nosaukumu, saites komentārus, kā arī lietotāja izvēlētu informāciju par pašu vienumu. Katram piesaistītajam vienumam ir iespēja izsekot tā saitēm. Tādējādi lietotājs var pārvietoties jeb navigēt pa saitēm no viena vienuma pie cita, lai novērtētu savstarpējās atkarības. Lietotājs var arī aplūkot vienuma vēsturi, lai novērtētu, kādas izmaiņas ar šo vienumu jau ir notikušas.

Lietotājam ietekmes analīzes procesā ir jāņem vērā tas, ka eksistē divu veidu saites, kur potenciālas izmaiņas bērnos var rasties tikai stingras saites gadījumā ar mainīto un uz to norādošo vienumu.

Lai arī individuālo saišu aplūkošana sniedz visu vajadzīgo informāciju par saitēm, no programmas lietojamības viedokļa noteikt viena mainītā vienuma potenciālo ietekmi uz citiem vienumiem var izrādīties apgrūtinoši. Tas ir īpaši aktuāli pastarpinātas atkarības gadījumā. Piemēram, ja kodu ietekmē prasības maiņa, bet kods un prasības pa tiešo nav saistītas, tad ir nepieciešams līdzeklis, lai lietotājs varētu uzskatāmi redzēt, cik moduļi vai koda funkcijas potenciāli tiek skartas. Šajā nolūkā rīkā ir iebūvēts mehānisms, kas ļauj attēlot informāciju saišu koka veidā. Koks sniedz skatījumu uz to, kā no vienuma var izsekot uz citiem vienumiem vecāku virzienā (skats atpakaļ uz agrākiem notikumiem un vienumiem) vai bērnu virzienā (skats uz priekšu un sekojošiem notikumiem un vienumiem). Šī ir metode, kā pirms izmaiņu veikšanas vienā no vienumiem lietotājs var novērtēt ietekmes apjomu (cik daudz ir atkarīgo vienumu), kā arī ietekmes dziļumu (cik tālu pastarpinātā formā tiek ietekmēti vienumi). Koka attēlošanā var parādīties tādas situācijas, ka viens un tas pats vienums atkārtoti tiek attēlots kā atšķirīgam zaram piederošs elements. Tas notiek gadījumos, kad no viena vienuma līdz citam var nonākt pa diviem atšķirīgiem ceļiem. Piemēram, no vienas konkrētas prasības ir atkarīgi vairāki projektējuma vienumi (ekrāni), bet koda implementācijā tiek izmantota vienota funkcija.

Ja vienums ir vienlaikus stingri atkarīgs no vairākiem vienumiem, tad ir jāņem vērā visu vecāku ietekme. Tas nozīmē, ka vienums ir ietekmēts, ja kaut viens no vecākiem tiek mainīts. Ja tiek aplūkotas vairāku vecāku ietekmes, tad vienums ir uzskatāms par atbilstošu aktuālajai situācijai tikai pēc visu ietekmju novērtējuma un ņemšanas vērā.

Lai arī no savstarpējo atkarību viedokļa cikliem nevajadzētu būt, trasējamības modelis neaizliedz ciklu pastāvēšanu. Cikli var veidoties ar nestingro saišu starpniecību, kad tiek pievienotas saites ar informatīvu nozīmi, bet dziļākas semantiskās slodzes. Rīkā realizētais koks attēlo ciklu līdz elementam, kas attiecīgajā trasējamības ceļā jau ir bijis sastopams. Īpaši tiek izcelta cikla esamība, lai informāciju analizējošā persona varētu novērtēt cikla esamības pamatotību un nepieciešamības gadījumā pārstrukturētu informācijas saites.

Izmaiņu analīzes process

Iepriekš aprakstītās metodes izmaiņu ietekmes analīzei ļauj apzināt ietekmētos vienumus, tomēr pati pieeja balstās uz to, ka lietotājam ir sistemātiski jāpārbauda katra mainītā vienuma saites. Lai risinātu šo problēmu, rīkā ir ietverts mehānisms ietekmes automātiskai novērtēšanai. Šajā nolūkā katrs vienums var būt vienā no trijiem minētajiem statusiem:

A (*Actual*) – vienuma pašreizējā vērtība (t.i. tā aprakstošie lauki, saturs), kas atbilst reālajam stāvoklim;

P (*Pending*) – vienums ir jāpārbauda, jo tas ar stingru saiti ir atkarīgs no cita vienuma, un vecāka mainīšanas rezultātā, iespējams, ir jāmaina;

C (*Candidate for Pending*) – vienums, kas ar stingru saiti ir atkarīgs no Pending vienuma un kas tiešā veidā nav ietekmēts.

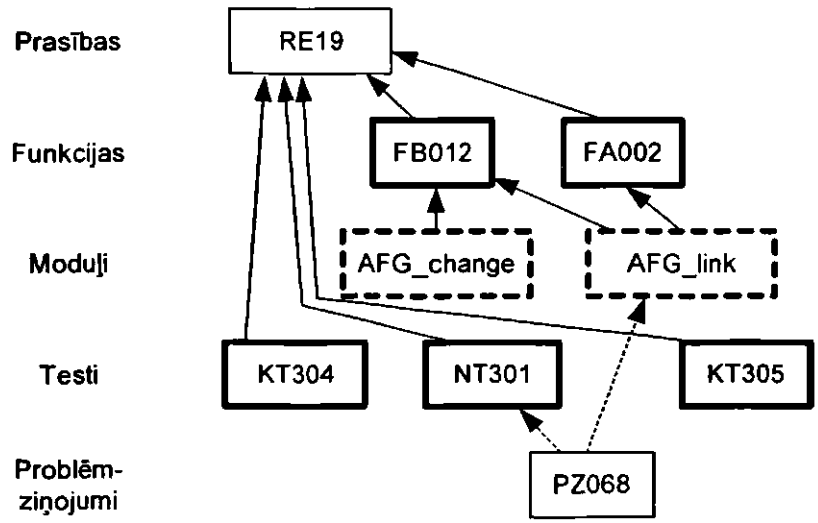
Jaunizveidotam vienumam statuss ir A. Ja tiek mainīts vienums ar statusu A, tas saglabā statusu A. Visi A ar stingro saiti saistītie bērni iegūst statusu P. Ņemot vērā to, ka vienums var iegūt statusu P vairāk kā viena mainīta vecāka rezultātā, katram vienumam tiek reģistrēts, kuri vienumi bija tie, kas izsauca statusa nomaiņu uz P. Statusa maiņas likumi ir parādīti 21.tabulā.

21.tabula

Vienuma statuss pirms maiņas	Tiek mainīts vienuma saturs	Tiek mainīts vecāka saturs	Tiek mainīts vecāka statuss	Vienuma statuss pēc maiņas
A	x			A
A		x		P
A			no A uz P	C
A			no A uz C	C
P	x			A
P		x		P
P			no A uz P	P
P			no A uz C	P
P			no C uz P	P
P			no C uz A	P
C	x			A
C		x		P
C			no P uz A	P vai A
C			no C uz A	C vai A
C			no C uz P	C
C			no A uz C	C
C			no A uz P	C

Stāvokļa maiņa no P uz C nav iespējama – vienums pēc izmaiņām var palikt vai nu ar statusu P, vai arī iegūt statusu A. Ja vienumam ir vairāki vecāki, no kuriem var veidoties atšķirīgi stāvokļi no dažādiem trasējamības ceļiem (skat., piemēru situācijas 12.attēlā), tad summārais stāvoklis tiek iegūts saskaņā ar 21.tabulā norādīto likumu. 12. un 13.attēlos ir parādīti piemēri stāvokļu maiņām pēc izmaiņām atsevišķos vienumos. Pirmajā situācijā (12.attēls) tiek mainīta prasība RE19. Tas izsauc statusa

izmaiņas no A uz P funkcijās FB012 un FA002, kā arī testos KT304, NT301 un KT305. Moduļiem AFG_change un AFG_link tiek piešķirts statuss C. Tajā pašā laikā statuss nemainās problēmziņojumam PZ068, jo tas ir saistīts ar notikuma (nestingro) saiti.



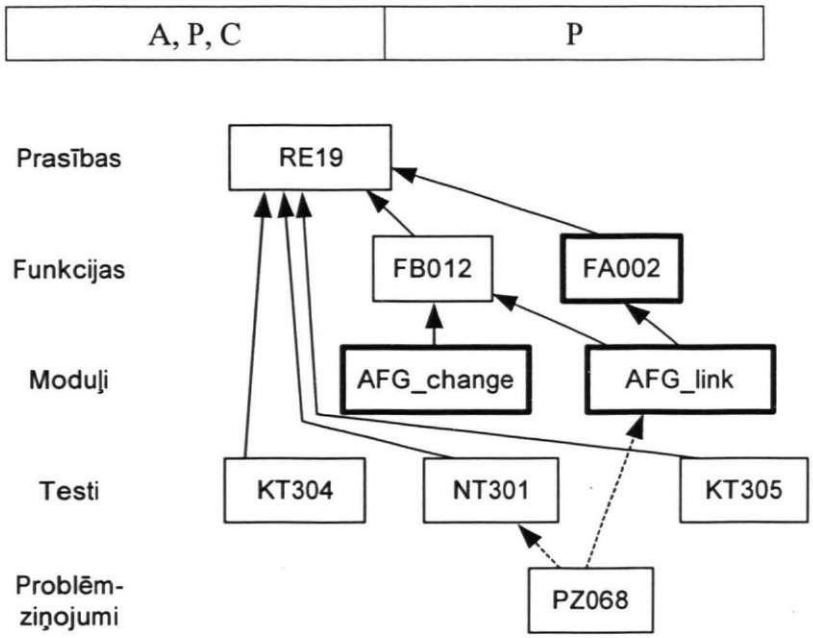
12.attēls. Piemēra situācija ar izmaiņām un stāvokļiem (1.).

Piemērā parādītajā situācijā lietotāja pienākums ir apskatīt minētās izmaiņas, un nepieciešamības gadījumā veikt izmaiņas vienumos. Ja izmaiņas nav nepieciešamas, tad lietotājam ir jāapstiprina, ka ir ņemtas vērā vecāka izmaiņas (t.i. veic tikai apstiprinājumu). Speciāla situācija ir ar moduli AFG_link. Tam vienlaikus ir divi vecāki – funkcijas FB012 un FA002. Ja lietotājs veic izmaiņas vai apstiprinājumus visos vienumos ar statusu P, izņemot FA002, tad tiek iegūta 13.attēlā parādītā situācija. Abiem attēlotajiem moduļiem statuss ir nomainījies no C uz P. Lai arī AFG_link attiecībā pret FA002 it kā varētu būt ar statusu C, noteicošais ir no cita vecāka iegūtais statuss P. Ja FA002 tiks mainīts pirms AFG_link, tad AFG_link būs ar statusu P, un pietiks ar vienu izmaiņu vai apstiprinājumu, lai tam būtu statuss A. Savukārt, ja tiks mainīts AFG_link pirms izmaiņām funkcijā FA002, tas joprojām saglabās statusu C, jo potenciāli ir iespējams, ka FA002 izmaiņas var izsaukt nepieciešamību mainīt arī AGF_link.

Gadījumi ar vairāku vecāku statusiem ir attēloti 22.tabulā.

22.tabula.

Stāvokļi, kas izriet no dažādiem trasējamības ceļiem	Vienuma summārais stāvoklis
A, P	P
C, P	P
A, C	C



13.attēls. Piemēra situācija ar izmaiņām un stāvokļiem (2.).

Rīkā lietotājs pēc noklusēšanas vienumam redz summāro stāvokli, bet vienlaikus tiek uzturēta informācija arī par saitēm un vienumiem, kas ir izraisījušas attiecīgā vienuma stāvokļa maiņu. Ja mainās viena no vecākiem stāvoklis, tad attiecīgi tiek atsvaidzināts arī paša vienuma stāvoklis.

Lietotājs ikvienam vienumam var aplūkot izmaiņas vecākos, kas izsauca attiecīgā vienuma izmaiņas. Iespējamie varianti tālākajai attīstībai:

- Vecāka izraisītās izmaiņas tiek ņemtas vērā, tiek izmanīts konkrētais aplūkojamais vienums, un šī vienuma bērni iegūst statusu P.
- Vecāka izraisītās izmaiņas tiek ņemtas vērā, tiek izmainīts konkrētais vienums, bet tālāk izmaiņas netiek izplatītas – visi bērni attiecīgā trasējamības ceļā ir ar statusu A (t.i. tālākā izmaiņu izplatīšanās tiek apturēta, notiek statusa nomaiņa no C uz A, ja vien C neizriet arī no cita vienuma izmaiņām).
- Vecāka izraisītās izmaiņas netiek ņemtas vērā (t.i. vecākam izmaiņas neizraisa izmaiņas bērnos), un attiecīgais vienums un visi bērni tā trasējamības ceļā ir ar statusu A.

Papildus – nerealizēta ir iespēja lietotājam īpaši norādīt, ka attiecīgā vienuma izmaiņas neizsauc nekādas izmaiņas bērnos (t.i. preventīvi novērts to statusa maiņu no A uz P).

Paplašināta analīze

Nereti rodas nepieciešamība operēt nevis ar individuālu vienumu izmaiņām, bet noskaidrot, kā viena vienumu kopa, piemēram, vairākas prasības, izmaiņu gadījumā

ietekmēs citu, piemēram, ar klientu saskaņoto testu kopu. Šāds paplašinātās analīzes gadījums ir plānots kā trasējamības rīka funkciju paplašinājums.

Minētās iespējas šobrīd rīkā TraceIt vēl nav implementētas, bet ir ietvertas nākotnes attīstības plānos. Paplašinājums ļaus izvēlēties apakškopu ar vienumiem, un iegūt apvienotu skatu (sarakstu) ar vienumiem, kurus ietekmētu izvēlētās vienumu kopas vienumu izmaiņas. Šobrīd šāda veida atbildes ir iegūstamas pēc izvēlēto vienumu individuālas analīzes.

6.8. Tīmekļa programmatūra ar datubāzi

Jaunam rīkam kā viens no priekšnosacījumiem tehniskajai realizācijai bija ideja par efektīvu informācijas apmaiņu starp visiem projekta dalībniekiem un nepieciešamības gadījumā arī pasūtītāju un projektu uzraugošām struktūrvienībām. Vairākos projektos praktizētā pieeja ar elektroniskām tabulām kā vidi informācijas reģistrēšanai un apstrādei prasa papildus darbu, lai sinhronizētu dažādās vietās esošu informāciju, kā arī apvienotu vairāku personu darba rezultātus. Tas parasti ir kontekstā ar prasību ievērot organizatorisku disciplīnu aizpildīšanas formāta, datu atbilstības un savlaicīguma ziņā. Risinājums tam ir informācijas uzkrāšana centralizētā vietā. Piemēram, centralizēti var tikt uzturēta dokumentu kopa ar hipersaitēm. Cits risinājums, kas ir elastīgāks no informācijas atlases un uzturēšanas viedokļa, ir atbilstošās informācijas glabāšana datubāzē. Datubāzes risinājums ļauj arī elastīgāk nodrošināt informācijas apmaiņu un integrāciju.

6.9. Rīka izstrādes tehnoloģiskie aspekti

Autors nodefinēja tehnoloģiskos pamatprincipus, bet turpmākā arhitektūra tapa sadarbībā ar kolēģi Mihailu Bogdanovu. Šobrīd Mihails Bogdanovs ir arī galvenais implementācijas darbu veicējs.

Rīks TraceIt ir implementēts kā interneta pārlūkā darbināma programmatūra, kas balstās uz HTTP servera (piemēram, Apache [29]), PHP valodas [115] un MySQL datubāzu vadības sistēmas [110] sadarbību. Tīmekļa bāzēts risinājums tika izvēlēts, balstoties uz šādiem apsvērumiem [12]:

- plaša saderība ar servera puses un klienta puses programmatūrām,
- tīmekļa popularitāte un daudzi pieejamie standarti,
- klients var izmantot programmu bez tās instalēšanas uz sava datora,
- iespēja to padarīt pieejamu interneta vidē,
- daudzveidīgie pieejamie informācijas šifrēšanas mehānismi.

No tehniskā viedokļa tika izvēlēts risinājums, ka visa informācijas apstrāde notiek servera pusē, un klientam ir tikai pieejama saskarne, kas ļauj izsaukt un apskatīt informāciju, izmantojot standarta pārlūkus tādus kā MS Internet Explorer,

Netscape Navigator, Opera vai Mozilla. Ņemot vērā rīka TraceIt projekta pētniecisko un eksperimentālo raksturu, kā pievilcīgi faktori bija arī tādi, ka izvēlētajiem risinājumiem ir elastīga licencēšanas politika, kā arī tas ir guvis interneta lietotāju vērtējumu kā stabils un no uzturēšanas viedokļa perspektīvs risinājums – PHP un MySQL risinājums tiek izmantots daudzu interneta portālu veidošanā.

Rīka izstrādes ietvaros bija jārisina divi globāli aspekti, kas izriet no paredzētā tipa programmatūras prasībām:

- lietotāju autorizācija – katram lietotājam jābūt pieejamām tikai tām lapām un ekrāniem, kas atbilst viņa tiesībām un veicamā darba kontekstam;
- daudzlietotāju režīms – katram lietotājam jāstrādā ar savu skatu uz datubāzē esošo informāciju, kā arī jārisina vairāku lietotāju potenciāli konfliktējošas darbības attiecībā pret vienu un to pašu objektu.

Lai arī lietotāja autorizācijas klasiskais risinājums ir katram lietotājam nodefinēt lietotāja vārdu un paroli, kas glabājas datubāzē, un darba uzsākšanas brīdī tiek salīdzināta ar lietotāja ievadītajiem datiem, tas neatrisina jautājumu par to, kā sistēmai pareizi vadīt katru nākošo darba soli. Šajā nolūkā tika izmantots sīkdatņu (*cookie*) mehānisms. Brīdī, kad lietotājs ir atpazīts kā legāls lietotājs, servera pusē tiek uzģenerēta sīkdatne ar unikālu sesijas numuru, kas tiek saglabāta uz lietotāja datora, lai to izmantotu tālāk pie katras nākošās tīmekļa lapas izsaukšanas. Katrā PHP modulī ir iestrādāts autorizācijas funkcijas izsaukums, kur sīkdatnē esošais sesijas numurs tiek salīdzināts ar datubāzē saglabāto. Ja tas nav derīgs, tad lietotājam tiek atvērta nevis izsauktā, bet gan pieslēgšanās lapa. Tīmekļa vides īpatnība ir tāda, ka nav iespējams zināt, vai lietotājs vēl strādā (skatās) ar sistēmu, vai ne. Lai uzlabotu drošību un samazinātu neaktīvu sesiju skaitu, katrai sesijai tiek uzskaitīts tās neaktivitātes periods. Ja tas pārsniedz kādu iepriekš nokonfigurētu vērtību, piemēram, stundu, tad mēģinājums izmantot sīkdatni ar izbeigušās sesijas numuru tiks interpretēts kā neveiksmīgs pieslēgšanās mēģinājums. Tas, ka sesijas ilguma kontrole ir servera pusē, ļauj izvairīties no ļaunprātīgiem pieslēgšanās mēģinājumiem, piemēram, izlabojot sīkdatni.

Līdzīgi risinājumi ar sīkdatņu izmantošanu šobrīd ir lielai daļai tīmekļa sistēmu, kur ir darbs ar reģistrētiem lietotājiem, tomēr TraceIt izstrādes laikā šāda pieeja tika oriģināli izstrādāta projekta ietvaros.

Pati par sevi tīmekļa vide ir labi piemērota daudzlietotāju režīmam. Lapu gadījumā bez lietotāju autorizācijas tiek apstrādāti visi lietotāju pieprasījumi – tiktāl der standarta pieeja. Ja tiek izmantota lietotāju autentifikācija, sistēmai ir jāpārliecinās, kura informācija ir sniedzama lietotājam. Taču gadījumā, ja divi vai vairāk lietotāji mēģina modificēt vienu un to pašu ierakstu, tad ir nepieciešams nodefinēt likumu, kā tiks apstrādātas šādas konfliktējošas situācijas. Izvēlētajiem izstrādes līdzekļiem PHP un MySQL nebija standarta mehānisma realizācijai, izņemot tabulu bloķēšanu datubāzē, kas traucētu darbu arī tiem lietotājiem, kuri neapstrādā

konfliktējošo objektu. Tādēļ tika ieviests savs bloķēšanas mehānisms – katrā situācijā, kur tiek izsaukta kāda vienuma rediģēšana, notiek šī ieraksta bloķēšana tā, lai citi lietotāji to nevarētu arī atvērt rediģēšanai, bet tikai skatīšanās režīmā. Problēma var rasties tajā brīdī, kad kāds ir atvēris objektu rediģēšanai, bet pēc tam neveic izmaiņu saglabāšanu – objekts var mūžīgi palikt bloķētā stāvoklī. Lai tas nenotiktu, bloķēšanai tiek uzturēts derīguma termiņš, pēc kura beigām citi lietotāji var iegūt rediģēšanas tiesības, bet sākotnējam lietotājam tās tiek anulētas. Ja pēc derīguma termiņa beigām neviens cits nebija mēģinājis iegūt objekta rediģēšanas tiesības, tad sākotnējais lietotājs šīs tiesības saglabā.

Cits sistēmas arhitektūras līmeņa risinājums ir saistīts ar vienumu tipu un vienumu ierakstu uzturēšanu. Kā jau tika minēts konceptuālajā aprakstā, projektam var būt jebkāda veida vienumi, kur katrs ir aprakstāms ar visdažādākajiem atribūtiem. Ja sākotnējā iecere bija uzturēt tikai pašu minimālāko informāciju par vienumu (identifikatoru, nosaukumu un piezīmes), tad jau rīka sākotnējā izstrādes stadijā kļuva skaidrs, ka ir jāļauj lietotājiem uzturēt pēc iespējas pilnīgu informāciju par vienumu. Analīze liecināja, ka vienumu aprakstošā informācija varētu veidoties no laukiem, kuru tips ir: garš teksts, īss teksts (līdz 80 simboliem), datums, lietotājs, skaitliskais lauks, kā arī jebkāda veida konkrētajā projektā definētie pārskaitāmie lauki izvēles saraksta formā. Ņemot vērā to, ka sistēmas koncepcija paredzēja brīvu vienumu tipu konfigurēšanu, saskaņā ar klasisko pieeju tas nozīmētu, ka ir nepieciešams definēt un pārdefinēt atbilstoša tipa tabulas datubāzes līmenī. Lai arī tehniski tas būtu iespējams, šāda pieeja ieviestu nenoteiktības un sistēmas drošības apdraudējuma faktoru no arhitektūras viedokļa. Tā vietā tika nolemts visu organizēt iepriekš definētu tabulu ietvaros, dinamiski nemainot ER modeli, bet gan uzturēt trasējamības modeli kā augstāka līmeņa modeli bez tieša atspoguļojuma tabulu veidā. Izvēlētais risinājums pieprasa šādas entītes, kas nodrošina trasējamības modeļa un vienumu informācijas uzturēšanu:

- lauku tipu definīcijas,
- vienumu tipu saraksts, sasaistot to ar lauku tipiem,
- vienumu saraksts.

Konkrēti, tika izveidotas atsevišķas tabulas, kurās glabājas attiecīgā lauku tipa (datumu, garais teksts, izvēles lauki utt.) vērtības. Tas nozīmē, ka katra ieraksta attēlošana nav vienkāršs (tradicionāls) datubāzes pieprasījums, bet gan ietver tādas darbības, kā tipa definīcijas nolasīšanu un atbilstošo lauku vērtību nolasīšanu no saistītajām tabulām un individuāli sameklēto lauku vērtību apvienošanu vienā ierakstā.

6.10. Nodaļas secinājumi

Nodaļā tika aprakstītas pamatnostādnes, saskaņā ar kurām funkcionē rīks TraceIt. Tas ir veidots ar mērķi, lai programmatūras projekti ar minimālu

sagatavošanās darbu varētu sākt lietot rīku, un tā ikdienas izmantošana būtu ērta un saturiski noderīga. Rīks uzkrāj un apstrādā informāciju saskaņā ar tajā definēto trasējamības modeli un atkarību apstrādes principiem. Vienumi un saites ir konfigurējami, bet tajā pašā laikā tiem ir standartizēta uzturēšanas un apstrādes forma. Rīka tehniskais risinājums tika balstīts, ņemot vērā iepriekš novērotās problēmas ar vienkāršām pieraksta metodēm, izmantojot biroja programmatūras līdzekļus, kā arī ņemot vērā mūsdienu programmatūras risinājumus – interneta pārlūkprogrammā darbināma programma, kas izmanto centralizētu datubāzi. Izmaiņu ietekmes analīzes kontekstā ir ieviests vienuma statusa jēdziens, kas raksturo vienuma iespējamās mainīšanas nepieciešamību.

Nodaļa parāda iespējamos risinājumus šādām testēšanas problēmām: Te1 – eksistē iespēja ar TraceIt nodrošināt informāciju par izpildītajiem testiem un to atbilstību prasībām. Te2 – ar TraceIt kā uz datubāzes balstītu risinājumu pastāv iespēja atlasīt nepieciešamo informāciju. Te3 – ja problēmas tiek uzkrātas TraceIt rīkā, tad tā lietotāji (testētāji un izstrādātāji) var operatīvi veikt informācijas apmaiņu par problēmām.

Nodaļa parāda iespējamos risinājumus šādām trasējamības problēmām: Tr1 – tiek piedāvāts jauns risinājums trasējamības uzturēšanai, kas ir alternatīva biroja programmatūrai. Tr3 – ja vienuma informācija ir tāda, ka pašu vienumu var uzturēt rīka ietvaros, tad pastāv iespēja uzturēt trasējamības informāciju aktuālā stāvoklī pēc katras izmaiņas veikšanas. Tr4 – rīks TraceIt neuzspiež konkrētu izstrādes vai testēšanas metodiku, tādēļ pastāv iespēja to adaptēt dažāda tipa projektu apstākļiem.

7. Rīka TraceIt ieviešanas pieredze

7.1. Nodaļas mērķi

Šajā nodaļā ir aprakstīts, kā notika rīka TraceIt ieviešana industriālos projektos IT uzņēmumā. Mērķis ir novērtēt, kā šis rīks atbalstīja projekta darbus atkarībā no projekta izmēra, projekta uzdevuma un projektu darbinieku lomām. Tiek novērtētas projektu aktivitātes, kurās rīks tika izmantots visvairāk. Atziņas par rīka pielietošanu iegūtas gan no autora individuālajiem novērojumiem projektos, gan arī no saskaņā ar Delfi metodi veiktas aptaujas lietotāju-ekspertu vidū. Tādējādi tiek atspoguļots projektu viedoklis par rīka TraceIt sniegto atbalstu un iespējami izraisītajām problēmām reāla darba apstākļos.

Nodaļas ietvaros tiek norādīti rīka sniegtie risinājumi iepriekš identificētajām testēšanas un trasējamības problēmām, kā arī sniegts pamatojums darbā izvirzītajam apgalvojumam.

7.2. Ieviešanas pieeja

Ņemot vērā to, ka rīks TraceIt tika izstrādāts zinātniski-pētnieciskā iestādē, kas strādā komerciāla uzņēmuma paspārnē, teorētiski bija relatīvi liela projektu izvēle (vienlaicīgi ir aktīvi aptuveni 20 projekti), kuros būtu iespējams ieviest rīku. Galvenā problēma bija risks traucēt vai pārtraukt projekta darbu rīka neveiksmīgas funkcionēšanas gadījumā. Risks ir jo lielāks, jo vairāk projekta informācijas tiks uzturēts rīkā. Tādēļ tika izvēlēta pakāpeniskas ieviešanas stratēģija ar iteratīvu atgriešanos pie uzlabojumu realizēšanas un ekspluatācijas laikā konstatēto problēmu novēršanas. Viena projekta ietvaros pakāpeniska pieeja netika uzskatīta par labu, jo tādā gadījumā noteiktos laika posmos projekta informācija dublēsies vairākās vidēs, un būs problēmas ar informācijas sinhronizāciju.

Apkopojot to, kā notika TraceIt ieviešana, var iezīmēt šādus etapus:

1. testēšana un eksperimentāla ekspluatācija izstrādes grupas ietvaros,
2. ieviešana pirmajos projektos ar sistemātisku izstrādātāju atbalstu - pilotprojekts,
3. atsauksmju savākšana par rīka darbību,
4. rīka uzlabojumu realizēšana un problēmu novēršana,
5. ieviešana jaunos projektos (un pāreja uz 4.etapu).

Gan TraceIt izstrādes laikā, gan tā eksperimentālās ekspluatācijas laikā izstrādes grupa saviem nolūkiem uzturēja vairākas rīka darbības vides un versijas:

- izstrādes vide – rīka attīstīšanai, labojumu un papildinājumu implementēšanai, atkļūdošanai,

- testa vide – rīka testēšanai ar speciāli sagatavotiem testa datiem,
- demonstrācijas jeb eksperimentālās ekspluatācijas vide – ieinteresētiem projektu pārstāvjiem izmēģināšanai.

Pirmie projekti, kuros TraceIt tika ieviests, bija nelieli, un eksistēja rezerves variants (elektronisko tabulu formā), ko pēc papildus darba ieguldīšanas varētu izmantot kā aizstājošu pagaidu risinājumu. Atsauksmes tika ievāktas kā projekta apskašu rezultāti.

7.3. Organizatoriskie apsvērumi

Apkopojot iepriekš minētos trasējamības rīka lietošanas apsvērumus, varēja secināt, ka tā izmantošanai ir virkne veicinošu faktoru:

- Nav jāpielāgojas rīka piedāvātajai metodikai, jo to var izvēlēties lietotājs pats.
- Vieglāka kļūst dokumentu uzturēšana un to skaitu vai apjomu ir iespējams samazināt.
- Visi projekta darbinieki var operatīvi iegūt projekta darba informāciju.
- Nekas nav īpaši jāinstalē uz lietotāju datoriem.
- Nav jāpērk lietošanas licences.

7.4. Pilotprojekts

Pirmā ieviešana balstījās uz autora veikto situācijas novērtējumu uzņēmumā. Pilotprojekta ietvaros rīks tika ieviests trijos projektos viena gada garumā [8], par diviem no kuriem ir sniegta informācija tālākajā tekstā. Paralēli tika veikts papildus pētījums par to, kādi varētu būt uzņēmumam tipiski nepieciešamie trasējamības modeļi. Mērķis būtu izveidot rīkam TraceIt bibliotēku ar trasējamības modeļiem, kurus jaunie projekti varētu ņemt kā sagataves sava trasējamības modeļa nodefinēšanai.

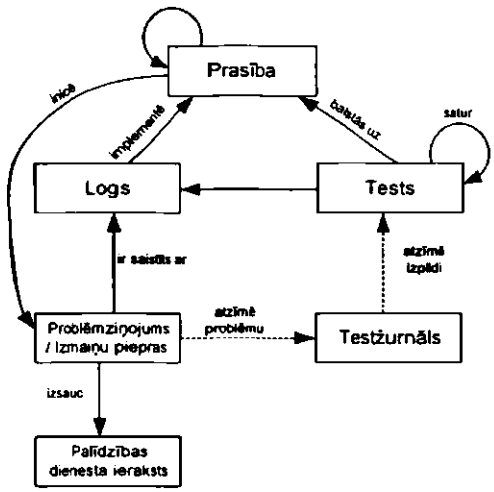
Projekts A

Projekts A bija komerciāls pilnas izstrādes projekts. Maksimālajā brīdī tajā bija iesaistītas vairāk kā 25 personas, un tā ilgums bija 5 mēneši. Izstrāde notika saskaņā ar klasisko ūdenskrituma modeli ar atsevišķiem izņēmumiem. Pirms aplūkojamā projekta jau bija veikta sistēmanalīze, tādēļ šī projekta sākuma aktivitātes bija saistītas ar projektējuma izstrādi. Projekta laikā tika identificēti šādi projekta vienumu tipi (trasējamības modelis - 14.attēlā):

- prasība,
- logs,
- tests,
- testžurnāls,
- problēmziņojums / izmaiņu pieprasījums un

- palīdzības dienesta ieraksts.

Implementācijas sākuma etapā projektā dominēja jautājumi, kas saistījās ar projektējuma elementu saistību ar prasībām. Prasības bija strukturētas hierarhiski, tādēļ starp prasībām arī bija iespējamas saites (prasība P1 „ietver" ietver prasību P2). Implementācija un testu definēšana notika paralēli.



14.attēls. Pilotprojekta A trasējamības modelis.

Projektējums tika gatavots kā nododams dokuments MS Word formā. Dokumenta struktūra bija veidota atbilstoši veidojamās sistēmas ekrāna formām. Ekrāna formas identificēšana norādīja uz atbilstošo realizējamo funkcionalitāti. Vadoties no šādiem apsvērumiem, Tracelt rīkā tika uzturēts ekrānu saraksts, kuru projektējuma pilns apraksts bija atrodams sagatavotajā dokumentā. Ekrānus ar prasībām saistīja stingra saite („implementē"). Vispārējā gadījumā starp prasībām un ekrāniem bija novērojama „daudz pret daudz" tipa attiecības, tomēr vairumā gadījumu vairākas prasības attiecās uz vienu ekrānu. Tika veidoti divu veidu testi: saskarnes orientētie testi, kas balstījās uz ekrānu projektējumu, un funkcionalitātes testi, kas balstījās uz prasībām. Tā rezultātā testam ir iespējamas stingras saites ar prasībām un logiem. Testi, līdzīgi kā prasības, bija ar hierarhisku struktūru. Tests var būt vai nu konkrēts izpildāms tests, vai arī kalpot kā ietvars vairākiem citiem testiem ar kopīgu mērķi. Visa testa informācija tika glabāta Tracelt-ā.

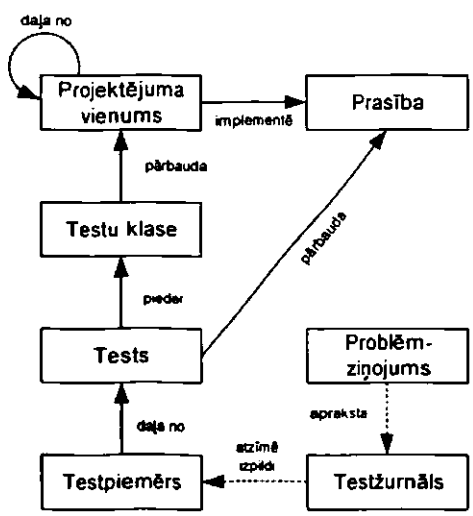
Līdz ar testu izpildes uzsākšanu tika nodefinēti projekta vienumu Testžurnāls un Problēmziņojums / Izmaiņu pieprasījums (PZ/IP). Testžurnāla funkcija bija reģistrēt testu grupas izpildes rezultātus. Testētāji par katru veikto testu veidoja vienu testžurnāla ierakstu, un ar saiti atzīmēja norādi uz konkrētajiem testiem. PZ/IP tika veidots atbilstoši citos uzņēmuma projektos ieviestajai praksei, ka problēmziņojums un izmaiņu pieprasījums var tikt aprakstīts ar vienāda tipa veidlapu. Citu projektu prakse bija liecinājusi, ka ne vienmēr uzreiz ir iespējams noskaidrot, vai pieteiktā problēma ir tiešām ir problēma, vai arī tas ir izmaiņu pieprasījums. Testžurnālam bija notikuma izraisītā saite ar testu („iezīmē izpildi"). PZ/IP bija ar divām nestingrajām (notikuma izraisītajām) saitēm – norāde uz logu un norāde uz testžurnāla ierakstu.

Ņemot vērā to, ka izmaiņu pieprasījumi nosaka izmaiņas prasībās, tika nodefinēta stingra saite starp PZ/IP un prasībām (prasība „iniciēta no" PZ/IP). Kad programmas pirmās versijas tika nodotas pasūtītājiem eksperimentālai ekspluatācijai, parādījās nepieciešamība reģistrēt lietotāju jautājumus un sūdzības – klienta atbalsta ierakstus. Daļa no reģistrētā parādīja jaunas sistēmas problēmas, kas izstrādes projektā bija jāizpēta un pēc iespējas savlaicīgi jānovērš. Atsevišķos gadījumos lietotāji bija izteikuši ierosinājumus, ko viņi vēlētos sistēmā redzēt citādāk. Šādos gadījumos runa ir par izmaiņām prasībās. Ja pieteicējs bija no pasūtītāja puses pilnvarots ierosināt izmaiņas, tad uz šāda pieteikuma bāzes projektā varēja nodefinēt formālu izmaiņu pieprasījumu. Klientu atbalsta ierakstus varēja pilnībā uzglabāt TraceIt-ā, un tika izveidota iespēja ar notikuma izraisīto saiti tam piesaistīt PR/IP (PZ/IP „iniciēts ar" Klienta atbalsta ierakstu).

Iespēja uzturēt PZ/IP ierakstus projekta gaitā izrādījās ļoti būtiska, jo projektam vienīgā pieejamā alternatīva bija MS Excel tabulas, kas pēc projekta pārvaldnieka iepriekšējās pieredzes nebija piemērots risinājums projektam ar samērā lielu skaitu iesaistīto darbinieku.

Projekts B

Atšķirībā no projekta A, trasējamības rīks tika ieviests, uzsākoties testēšanas darbiem. Ņemot vērā to, ka testēšanu veica neatkarīga testētāju grupa, bija iespēja nodalīt testēšanas procesa informāciju no pārējās projekta informācijas. Rīka galvenie lietotāji bija testētāju grupa. Trasējamības modelis ir parādīts 15.attēlā.



15.attēls. Pilotprojekta B trasējamības modelis.

Prasību un projektējuma informācija tika glabāta daudzos dažādos veidos. Projektējums bija veidots konspektīvā formā. Bija problēmas ar informācijas integritāti starp projektējumu un prasībām. Tajā pašā laikā projektā bija iespējams identificēt prasību un projektējuma vienumu sarakstu, kuri tika ieimportēti TraceIt rīkā kā atsevišķi projekta vienumi kopā ar informāciju par savstarpējām saitēm.

Projekta vienumiem bija pieļauta hierarhiska struktūra. Ievērības cienīgs ir fakts, ka informācijas importēšanas nolūkos notika prasību un projektējuma analīze, kas ļāva veikt aktualizējošus labojumus, un uzlaboja informācijas kvalitāti projektā.

Testēšanas informācija bija strukturēta vairākos līmeņos. Ņemot vērā to, ka projektējuma vienumi nebija ar augstu detalizācijas pakāpi, uz to bāzes tika definētas testu grupas. Testu grupas vienumi bija saistīti ar stingro saiti („pārbauda") ar projektējuma vienumiem – ekrāniem u.c. elementiem. Uz definēto testa klašu pamata tika definēti testi, kas vienlaikus bija atkarīgi no prasībām (tests „pieder pie" testu klases; tests „pārbauda" prasību). Testpiemērs bija stingri saistīts ar testu („atkarīgs no"). Testžurnāls ar nestingro saiti iezīmēja atsauci uz testpiemēru, bet problēmziņojums ar nestingro saiti iezīmēja atsauci uz testēšanas notikumu – testžurnāla ierakstu.

Abos projektos A un B projekta darbiniekus interesējošie jautājumi, kuru atbildes bija atkarīgas no trasējamības, pamatā bija:

- Kā savstarpēji ir saistīti divi vienumi?
- Vai visas prasības ir pārklātas ar testiem?
- Vai visas prasības ir notestētas (konkrētā programmas versijā)?
- Kāds ir problēmziņojumu statuss?
- Kas ir problēmu cēlonis?

7.5. Atziņas no ieviešanas

Viens no secinājumiem bija arī tas, ka rīka lietošanas panākumi ir atkarīgi no rīka funkcionālajām iespējām, lietojamības un veiktspējas (atbildes laika).

Ļoti būtisks apstāklis ir tas, ka rīks ļauj strādāt katram projekta darbiniekam. Katrs darbinieks ir atbildīgs par savas informācijas sadaļas uzturēšanu, un rīkā esošā pamatinformācija parasti ir oriģinālā vieta, kur tā tiek ievietota. Ja ir jātur informācijas kopija, parādās risks, ka tā nebūs aktuāla. Ja trasējamības rīku projektā lieto tikai neliela daļa no visiem, tad arī parādās risks, ka ne viss tiks ievietots vai saņemts no rīka - tā izmantošana nebūs efektīva. Viens no rīka efektivitātes kritērijiem ir tā popularitāte - cik daudz cilvēku to izmanto.

Informācijas apmaiņa ar programmatūras pasūtītājiem / lietotājiem - būtisks arguments par labu interneta bāzētam risinājumam. Katrā projektā, kur tika izmantots TraceIt, autors apzināja lietotāju viedokli un vēlmes par konkrētā rīka darbību. Ekspluatācijas gaitā ir uzkrāti daudz dažādi ieteikumi, kas pamatā attiecas uz darbības ērtuma uzlabošanu, vai arī tehniskās realizācijas niansēm. Parasti šādas atsauksmes tika uzzinātas neformālu sarunu formā, un daļu no tām ir izdevies arī realizēt. Ir tādas, kas prasītu pārāk daudz darba to realizēšanai.

Pēc TraceIt izmēģināšanas paša rīka izstrādes projektā, kā arī tā eksperimentālas ekspluatācijas divu cilvēku projektā, tika pieņemts lēmums rīku ieviest 5 mēnešus ilgā

projektā, kur maksimālais dalībnieku skaits pārsniedza 25 darbiniekus. Projekta vadība piekrita izmantot TraceIt, jo viens no rīka autoriem pats tika iesaistīts par projekta dalībnieku – vadīt projekta testēšanas grupu. Šāds variants garantēja potenciālu problēmu gadījumā operatīvu palīdzību. Tiesa, būtisku problēmu nebija – tika konstatētas atsevišķas problēmas rīka darbībā lietotāja saskarnes līmenī, kā arī saņemti ierosinājumi funkcionalitātes papildināšanai no lietotājiem. Liela daļa no ierosinājumiem vēlāk tika arī realizēta.

Rīkā tika uzturēti šādi vienumi:

- prasību saraksts,
- funkciju saraksts,
- problēmziņojumi,
- lietotāju ierosinājumi.

Rīks TraceIt tika izmantots tieši vienā lielā IT uzņēmumā vairākos projektos. Kompānijas pamatdarbības virziens ir specializētu informācijas sistēmu izstrāde. Viena no šīs IT kompānijas iezīmēm ir tā, ka projektiem var būt stipri atšķirīgs izmērs, ilgums un izmantotā metodika. Lieli projekti ietver līdz pat 100 cilvēkus un ilgst vismaz vienu gadu. Mazākajos projektos ir viens vai divi cilvēki, un tie parasti ilgst līdz pusgadam. Kompānija ievēro TickIT [40] kvalitātes principus, kā arī programmatūras izstrādē balstās uz standarta J-STD-016 vai tā pielāgojuma. Tikai viens no projektiem izmantoja specializētu prasību pārvaldības rīku – UML modeļu veidošanai. Prasību pārvaldības rīki citos projektos netika izmantoti, jo netika saskatīti ieguvumi, ko dotu šāda rīka izmantošana salīdzinājumā ar teksta dokumentiem.

Ne visi projekti piekrita lietot TraceIt rīku. Arī autors nebija drošs, ka rīks spēs pilnībā apmierināt lielu projektu prasības, kā arī būt piemērots tādiem projektiem, kuri jau ir sākušies un izmanto kādu risinājumu, pat ja šis risinājums prasa papildus darbu.

Tālāk ir atspoguļota ieviešanas pieredze 12 projektos trijos aspektos – projekta izmērs, uzdevumi un darbinieku iesaistīšanās. Papildus ir bijuši vēl 5 projekti, kas nav sistemātiski novērtēti, bet pieejamā informācija liecina, ka tajos gūtie secinājumi ir analoģiski pamata pētījumā veiktajiem. Galvenā rīka lietotāju auditorija bija mazi un vidēji projekti.

Kopīgais ieviešanas procesā

IT kompānijā lielākā daļa projekti ir saistīti ar programmatūras izstrādi, un neliela daļa ir orientēta uz produktu uzturēšanu vai neatkarīgo testēšanu. Pamata programmatūra balstās uz klienta-servera arhitektūras risinājumiem. Tipisks dzīves cikls atbilst inkrementālam vai cikliskam.

TraceIt ieviešana visos projektos bija līdzīga, un tā iekļauj šādus soļus:

1. Vispārēja informācijas apmaiņa starp rīka ekspertu un projekta pārstāvjiem par projektā notiekošajiem procesiem.
2. Rīka eksperts sagatavo sākotnējo trasējamības modeli.

3. Projekts veic modeļa apskati.
4. Koriģēts modelis tiek iekonfigurēts rīkā.
5. Rīks strādā atbilstoši modelim.
6. Nepieciešamības gadījumā modelis tiek mainīts.

Rīka lietošana bija projektu brīvprātīga izvēle. Par vienu no nodrošinājumiem kalpoja garantija, ka nepieciešamības gadījumā visu informāciju ir iespējams izeksportēt MS Excel saprotamā formātā. Tiesa, šajā gadījumā saitēm jāseko līdzi manuāli, un pazūd citas priekšrocības. Tomēr informācija nav pazaudēta, un stāvoklis vienkārši būtu analoģisks tam, kā bija citos projektos, kur trasējamībai un problēmu uzskaitei izmanto MS Excel. Ieviešanas rezultāts bija tāds, ka nevienam projektam neradās nepieciešamība pēc TraceIt ieviešanas no tā kādā brīdī atteikties vai kādā laika momentā izmantot drošības kopijas Excel formātā. Dažiem projektiem bija arī citi iespējamie alternatīvie varianti. TraceIt pārstāja lietot tie projekti, kuriem beidzās darbi. Bija projekti, kuri bija iecerējuši, bet tomēr vadības lēmumu vai organizatorisku pārmaiņu rezultātā tomēr neuzsāka rīka lietošanu. Nevienā rīka lietošanas gadījumā netika izmantotas piespiedu administratīvās metodes – katrs projekts pieņēma lēmumu pats.

Projekta izmērs

Atkarībā no iesaistīto personu skaita var iezīmēt trīs grupas: 5 nelieli (2-4 cilvēki), 5 vidēji (8-15 cilvēki) un 2 daļēji lieli (25-35 cilvēki). Nelielo projektu ilgums bija līdz 6 mēnešiem, vidējiem tipiski ilgums bija no 3 mēnešiem līdz 1 gadam, un abi novērotie daļēji lielie projekti darbojās attiecīgi 1 un 2 gadus.

Visu projektu gadījumā rīks tika piedāvāts projekta sākuma etapos, tādēļ tie bija samērā atvērti jaunu pieeju izmēģināšanā. Lomu spēlēja veiksmīgā eksperimentālā sākotnējā ekspluatācija vienā no projektiem. Visos gadījumos autors veica rīka eksperta lomu, bet tikai atsevišķos gadījumos piedalījās arī paša projekta norisē. Pēc trasējamības modeļa iekonfigurēšanas rīkā autors iemācīja kādam no projekta pārstāvjiem TraceIt administrēšanu, lai tālāko konfigurēšanu projekts varētu veikt pats.

Mazie projekti diezgan ātri apguva rīku un sev atklāja iespēju uzglabāt dažādas detalizācijas pakāpes datus – arī tādus, kas nepieder pie tipiskajiem projekta vienumiem – piezīmes, notikumu reģistru, atsauces uz dokumentiem, ierosinājumus utt. Galvenā izmantojamā priekšrocība bija iespēja sakārtot dažādos laika brīžos tapušo informāciju un to padarīt pieejamu kolēģu vidū. Alternatīvs risinājums būtu izmantot kādu no tiešsaistes diskusiju forumiem, bet tajā gadījumā būtu ierobežotas trasējamības iespējas. Mazajos projektos problēmu reģistrēšana netika saskatīts kā būtisks ieguvums, jo nelielās darba grupās tiek izmantotas arī alternatīvās komunikāciju metodes – klātienes saruna vai e-pasts. Divos gadījumos projekti pārslēdzās no neformālas uz detalizētu dokumentēšanu – lai varētu izsekot no līguma

caur prasībām līdz konkrētām funkcijām. Šāda pieeja prasa sistemātisku dokumentēšanu un izmaiņu gadījumā arī papildus darbu.

Vidējie un daļēji lielie projekti bija līdzīgi tādā veidā, ka tie nebija ieinteresēti ar šī rīka palīdzību uzturēt visu trasējamību līdz pat koda līmenim. Galvenā uzmanība bija veltīta prasībām un projektējumam. Projektus interesēja iespēja saistīt vienumus ar projekta personālu kā atbildīgajām personām, kā arī aplūkot izmaiņu vēsturi.

Kopumā izmērs neiezīmēja lielas atšķirības, izņemot šādus novērojumus:

1) Mazāki projekti bija vairāk ieinteresēti mēģināt kaut ko jaunu, atvērtāki riskam.

2) Lielāki projekti vairāk orientējās uz liela apjoma informācijas apstrādes problēmu, un tie nevēlas riskēt ar jaunu metodiku izmēģināšanu. Šādi projekti cenšas arī jebkura veida informāciju uzturēt formālākā veidā.

Projekta uzdevumi

No aplūkojamās grupas projekti, kuros tika izmantots TraceIt, var tikt iedalīti trijās kategorijās: pilnas izstrādes (7 projekti), uzturēšanas (3 projekti) un neatkarīgās testēšanas (2 projekti). Ar pilnu izstrādi ir domāts dzīves cikls, kas sākas ar prasību analīzi un noslēdzas ar funkcionējoša koda un dokumentācijas piegādi. Tipiskie vienumu tipi, kas tika iekļauti trasējamības modelī TraceIt rīkā, bija: dažāda līmeņa prasības, projektējuma vienumi vai programmatūras objekti (funkcijas, ekrāni, DB elementi), testi un problēmziņojumi. Izmaiņu pieprasījumi tipiski tika aplūkoti kā problēmziņojumu speciāls gadījums. Dažos projektos projektējuma vienumi tādi kā ekrāni tika identificēti atsevišķi, lai padarītu vienkāršāku izmaiņu ietekmes analīzi no lietotāja skata punkta, kā arī lai uzlabotu problēmu izsekošanu.

Projekti izskatīja arī iespēju modelī iekļaut atsauces uz kodu dažādos līmeņos (fails, klase, metode). Pirmie šī papildus darba veikšanas eksperimenti izrādījās maz efektīvi, jo izstrādes laikā kods ļoti bieži mainās. Daudz praktiskāka pieeja koda kopas uzturēšanai bija veikt koda versiju kontroli ar speciāli šiem mērķiem veidotiem rīkiem. Tajā pašā laikā tika secināts, ka tie piedāvā ļoti ierobežotas trasējamības iespējas – faktiski vienīgā iespēja bija nodrošināt trasējamību ar atbilstoši dotiem nosaukumiem, kas projektu vajadzībām bija pietiekami.

Uzturēšanas projektiem parasti ir stabilas prasības, un to galvenais uzdevums parasti ir novērtēt veicamo izmaiņu ietekmi, jo vienas prasības maiņa var nozīmēt plašas izmaiņas projektējumā, kodā, testos un dokumentācijā. Aplūkojamo uzturēšanas projektu trasējamības modeļi ietvēra atsauci uz kodu. Izmaiņas notiek regulāros laika intervālos, un tika konstatēts, ka ir vērts dokumentēt saites TraceIt datubāzē. Svarīgas ir saites ar izmaiņu pieprasījumiem – tie kalpo kā pamats veselai izmaiņu grupai. Motivējošs faktors detalizētam modelim bija fakts, ka uzturēšanas uzdevumi laiku pa laikam tiek nodoti no vienas grupas citai. Šajā gadījumā ir svarīgi uzturēt detalizētu projekta dokumentāciju.

Neatkarīgās testēšanas projektos programmatūra tika aplūkota kā melnā kaste. Papildus tam, novērotajos projektos tika veikti tikai funkcionālie testi. Tas izraisīja šādu vienumu tipu definēšanu: prasības/funkcijas, ekrāni, konfigurācijas, testi, testu rezultāti un problēmziņojumi. Testētāji izmantoja iespēju glabāt visu informāciju TraceIt rīka datubāzē. Ja projektos būtu bijusi testu automatizācija, noteiktas problēmas varētu parādīties saistībā ar automātisko testu koda glabāšanu – to glabāt TraceIt-ā, vai arī kā kodu versiju pārvaldības rīkā vai testēšanas rīka repozitorijā.

Ņemot vērā to, ka netika novērotas problēmas ar TraceIt lietošanu, var secināt, ka tas bija piemērots visiem iepriekšminētajiem projektu tipiem.

Projekta darbinieku iesaistīšanās

Programmatūras projektos darbiniekiem parasti eksistē dažādas lomas. Loma var būt vai pastāvīga visa projekta gadījumā, vai arī tā var mainīties atkarībā no etapa. Piemēram, sistēmanalītiķis laikā, kad kods ir implementēts, varētu veikt testētāja pienākumus. TraceIt lietotājus var iedalīt šādās grupās:

- projekta pārvaldnieks,
- sistēmanalītiķis / projektētājs,
- programmētājs,
- testētājs.

Projekta pārvaldnieki parasti neanalizē individuālas vienumu saites, bet gan ir ieinteresēti aplūkot kopējo stāvokli un novērot projekta attīstības tendences. Ja nelielā projektā projekta pārvaldnieks veic arī kādu citu lomu, piemēram, sistēmanlītiķa, tad lielākos projektos viņš neveic citus darbus. Pārvaldniekus TraceIt kontekstā interesēja šādas jautājumu sfēras: pēc specifiskiem kritērijiem atlasītu vienumu filtrētu vienumu saraksti ar apkopojošu informāciju, statistikas analīze, vienumu piederība personām, ikdienas vai iknedēļas darbu progresa novērtējums.

Sistēmanalītiķi un/vai projektētāji ir tiešā veidā iesaistīti ar trasējamību saistītās aktivitātēs. Viņi analizē un definē prasības, kā arī pārvērš tās projektējuma vienumos – ekrāna formās, klasēs, funkcijās. Nelielos vai vidēja izmēra projektos viena vai divas personas pārzina visas sistēmas īpašības, un viņiem nevarētu būt pārāk liela interese formalizēt visu ar rīka palīdzību. Prakse liecina, ka informācijas kvalitāte pārsvarā ir atkarīga no atbildīgā projekta darbinieka paša vēlmes, jo priekšraksti reti kad reglamentē to, cik smalki ir jāveic pieraksti.

TraceIt ekspluatācijas pieredze liecina, ka programmētājiem nav liela interese par trasējamības uzturēšanu. Iespējams, tas ir atkarīgs no katra profesionālā darba kultūras, kā arī no tā, ka kods pats netika glabāts rīkā, kā arī koda informācija bija pārstāvēta tikai augsta līmeņa jēdzienos. Neatkarīgi no projekta izmēra visvairāk rīku izmantoja testētāji. Viņi bija ieinteresēti iegūt labus prasību pārklājumu ar testiem, kā arī sekot līdzi testu izpildes statusam. Gan paši testi, gan testu izpildes rezultāti glabājās rīka datubāzē.

Testētāji trasējamības rīka izmantošanā bija vairāk ieinteresēti tādēļ, ka tas atbalstīja izmaiņu pārvaldības procesu, problēmu apstrādi un izpildīto testu uzskaiti. Projektā tā ir strauji mainīga informācija, tādēļ bez rīka atbalsta informācija kļūst vai nu nepilnīga, vai arī neaktuāla. Testētāji rīkā TraceIt var apkopot visu sev nepieciešamo [16].

Apsvērumi un novērotais atbalsts

Kopumā var secināt, ka galvenie iemesli, kādēļ rīks netika ieviests kādos projektos, ir:

- Eksistējošie un strādājošie projekti nevēlējās mainīt sava darba metodes, rīkus un procesu;
- Gan autors, gan programmētāji, kas veica šī rīka implementāciju nevarēja dot drošas garantijas par rīka stabilitāti un veiktspēju;
- Liela izmēra projekti dod priekšroku pārbaudītām metodēm, un vēlas nogaidīt ar eksperimentālu rīku tādu kā TraceIt lietošanu;
- Ilgtermiņa projekti nevarēja saņemt drošu garantiju, ka rīks tiks atbalstīts arī nākotnē;
- Nav tiešas sadarbības ar citiem rīkiem – informācija notiek datu eksporta un importa līmenī.
- Projekti pārāk maz velta uzmanību trasējamības jautājumiem.

Kā ir redzams, lielākā daļa apsvērumu ir vadības lēmumi, kas faktiski nebalstās uz iespējamām negatīvām gūtajām atziņām no TraceIt lietošanas.

Raksturojot galveno atšķirību starp projektiem, kur tika izmantots un kur nebija trasējamību atbalstošs rīks, var konstatēt virkni atšķirību. Kā pozitīvu rezultātu no rīka lietošanas var minēt:

- vieglāk pārvaldīt projekta vienumu atkarības,

- detalizēta izmaiņu ietekmes analīze un darbu apjoma novērtējums,

- projekta statusa informācija,

- visa ar testēšanu saistītā informācija tiek glabāta vienkopus ērti pieejamā formā,

- ģeogrāfiski sadalītām darba grupām izdevīgs ir tīmekļa risinājums.

Lietotāji intensīvi lietoja vienumu filtrētus sarakstus, saišu informāciju un statistiku. Tā kā programmas veiktspēja bija pieņemama, lietotājiem nebija problēmu veikt ātru meklēšanu un analizēt izvēlētu vienumu apakškopu. Noderīga ir rīka iespēja konfigurēt vienumus tā, lai visa aprakstošā informācija glabātos tieši rīkā. Īpaši tas attiecas uz testēšanas procesu. Ja atsevišķiem projektiem radās nepieciešamība veidot formālus dokumentus, atbilstošie dati tika eksportēti.

Galvenie lietošanas šķēršļi

Līdzās pozitīvajai pieredzei tika novēroti arī daži lietošanas apgrūtinājumi. Kā jau tika iepriekš minēts, TraceIt nerisina visas projekta problēmas. Īpaši tas bija spēkā

pirmajos projektos, kur tika ieviests šis rīks. Pirmkārt, pašā sākumā TraceIt bija ar mazāku funkcionalitāti un, otrkārt, autors nebija gatavs projektiem ieteikt labākos risinājumus. Autoram bija idejas par šī rīka lietošanu, bet eksperimentālā ekspluatācija norādīja uz lietām, ko var veikt citādi.

Tika novērotas trīs galvenās problēmas:

1. Lai efektīvi izmantotu TraceIt iespējas, projekta darbiniekiem bija jābūt labām zināšanām par rīku un trasējamību kopumā.
2. Dažos projekta darba gadījumos bija nepieciešams papildus darbs, lai informāciju uzturētu aktuālā formā rīka datubāzē.
3. Pirms trasējamības modeļa definēšanas konkrētajā projektā ir jāveic detalizēta piedāvātās metodikas un modeļa analīze.

Iepriekšminētās problēmas ir savstarpēji saistītas. Vienkārši projekta darbinieki nevēlas domāt trasējamības jēdzienos. No visa trasējamības modeļa konkrēta uzdevuma veikšanai viņi redz vienu vai divus vienumu tipus ar iespējamu saiti starp tiem. Tas, ka rīku var pielāgot dažādām metodikām un ka to var konfigurēt, nozīmē kompetentas personas nepieciešamību projektā – ne katrs projekta dalībnieks ar to var nodarboties. Ņemot vērā to, ka projektiem bieži vien sākumā ir tikai aptuvens priekšstats par projekta procesiem, dokumentiem un veicamajiem darbiem, tie labprātāk sāktu izmantot kādus pārbaudītus variantus, t.i. adaptētu arī pārbaudītus trasējamības modeļus. Autora mērķis ir rīkā TraceIt izveidot trasējamības modeļu bibliotēku, no kuras projekti varētu izvēlēties piemērotākos modeļus savai projekta situācijai. Tajā pašā laikā ir tāds apgrūtinājums, ka projektiem nav kritēriju konkrētas metodikas izvēlei.

Ir maza iespēja, ka komerciāliem programmatūras projektiem būs liela interese par atsevišķu trasējamības rīku. Reālāka iespēja ir šāda rīka iespējas iekļaut citos projektā izmantojamajos rīkos.

7.6. TraceIt lietotāju aptauja

Aptaujas nostādne

Neskatoties uz to, ka pēc TraceIt ekspluatācijas pirmajos projektos eksperimentālā režīmā tika konstatēts, ka šo rīku ir lietderīgi attīstīt tālāk un izmantot arī citos projektos, nebija formāli noskaidrota viedokļa par konkrētā rīka īpašībām un projekta darbinieku ieguvumiem. Atsevišķi projekti bija veikuši projekta noslēguma apskates, kur tika aplūkota arī TraceIt izmantošanas lietderība. Lai noskaidrotu, kāds ir lietotāju kopējais skatījums uz šī rīka lietošanu, tika noorganizēta aptauja. Tajā piedalījās lietotāji, kuri bija izmantojuši rīku vienā vai vairākos projektos. Daļa dalībnieku šobrīd vairs nav TraceIt lietotāji, jo strādā citā projektā.

Mērķi:

- Salīdzināt situāciju projektā ar un bez TraceIt rīka;

- Noskaidrot biežāk izmantotās funkcijas;
- Noskaidrot vēlmes attiecībā pret trasējamības rīkiem vispār;
- Uzzināt par lietotāju pozitīvajiem un negatīvajiem iespaidiem;
- Uzzināt viedokli par tā salīdzinājumu ar citu risinājumu izmantošanu.

Norises gaita

TraceIt aptauja tika veikta saskaņā ar Delfi (*Delphi*) metodi [50]. Delfi metode tika izvēlēta tādēļ, ka aptauja ļauj noskaidrot dažādu ekspertu vērtējumus. Vairāku aptaujas iterāciju laikā aptaujas dalībnieki iepazīstas ar visu dalībnieku sniegto atbilžu apkopojumu, kā arī šajā kontekstā var koriģēt un precizēt savu viedokli. Katrs vērtējums balstās uz aptaujas dalībnieka kā eksperta viedokli. Metode tiek plaši izmantota pētījumos ekspertu viedokļu apkopošanai. Būtiski ir arī tas, ka ne aptaujas noslēgumā, ne norises starpposmos neparādās katra individuālā eksperta viedoklis, bet gan tikai apkopojums.

Norises soļi tika veidoti saskaņā ar klasisko Delfi pieeju un tika ievērots tipiskais veicamo iterāciju skaits – trīs anketēšanas reizes. Anketu sagatavošanu un apstrādi veica autors, bet anketu izplatīšanu un savākšanu organizēja IT uzņēmuma Kvalitātes pārvaldības dienests. Funkcijas bija sadalītas, lai autors saņemtu rezultātus anonīmā formā – bez informācijas par to, kurš attiecīgo anketu ir aizpildījis. Soļi bija šādi:

1. Tika izvēlēta ekspertu kopa. Eksperti bija TraceIt lietotāji, kas vienā vai vairākos projektos bija ilgstoši izmantojuši šo rīku. No visiem potenciāli iespējamiem lietotājiem tika atlasīti tie, kuru pārziņā bija kādi projekta procesi vai kvalitātes jautājumi. Delfi metode atšķirībā no sabiedriskās domas vai citām aptaujām neprasa gadījumiski izvēlēties statistiski reprezentatīvu respondentu kopu. Šeit mērķis ir uzzināt tieši ekspertu viedokli. Tādēļ nav stingri noteikta viedokļa par to, cik ekspertus ir nepieciešams aptaujāt. Tā kā piedalīšanās aptaujā ir uz brīvprātīgiem principiem, autors ņēma vērā iespēju, ka daļa no izvēlētajiem ekspertiem būs pārāk aizņemti un nevarēs vai nevēlēsies piedalīties šajā pētījumā. Šajā nolūkā tika izvēlēti 26 potenciālie dalībnieki, pieļaujot, ka aptuveni 50% nepiedalīsies.
2. Eksperti saņem un aizpilda pirmo anketu. Pirmā anketa saturēja 15 jautājumus, uz kuriem dalībnieki varēja atbildēt brīvā formā. Plānots, ka aizpildīšana varētu prasīt 30-60 minūtes dalībnieka laika. Atbildes sniegšanai tika piedāvātas 10 dienas laika.
3. Autors analizē rezultātus. Tiek apkopoti izteiktie viedokļi, kas tiek sagrupēti pa jautājumu grupām. Ņemot vērā to, ka daļa jautājumu bija nevis par viedokli, bet gan par to, kuras rīka funkcijas respondents bija izmantojis un kuras nē, tad šāda tipa jautājumi netika iekļauti otrās kārtas anketā.

4. Eksperti saņem un aizpilda otro anketu. Iepretim apgalvojumiem eksperti var norādīt to, cik stipri viņi piekrīt izvirzītajam apgalvojumam. Iespējamie varianti ir 0 – nemaz nepiekrīt, 1 – daļēji piekrīt, 2 – pilnībā piekrīt. Papildus pie attiecīgā jautājuma ir iespēja norādīt vēl kādu viedokli, kas nav parādīts piedāvāto variantu vidū – gadījumā, ja iepriekšējās kārtās šāds variants bija piemirsies vai nešķitis pietiekami aktuāls.
5. Autors sagatavo statistisku apkopojumu. Katram viedoklim visi vērtējumi tiek summēti (0, 1 vai 2 punkti). Apkopojuma mērķis ir sagrupēt viedokļus pēc to svarīguma – zems, vidējs vai augsts. Tas tiek iegūts no katra viedokļa iegūtā indeksa vērtības, kas parāda, cik procentus no maksimāli iespējamā punktu skaita katrs viedoklis saņēma. Zems vērtējums atbilst indeksam no 0% līdz 34%, vidējs - no 35% līdz 66% un augsts – no 67% līdz 100%.
6. Eksperti saņem 3.kārtas anketas un iepazīstas ar dalījumu grupās pēc svarīguma līmeņa. Ekspertiem ir iespēja koriģēt pozīciju. Jebkuram viedoklim(-iem) var norādīt, vai tas būtu jāpaaugstina, vai jāpazemina reitinga ziņā. Papildus ir jānorāda pamatojums, kādēļ. Anketas aizpildīšanai tiek dots laiks 10 dienas.
7. Autors sagatavo noslēguma rezultātus. Katra eksperta vērtējums par pozīcijas maiņu pieskaita vai atskaita punktu skaitu proporcionāli 3.kārtā iesaistīto skaitam. Papildus viedokļi arī tiek vērtēti saskaņā ar principu, ka nesniegts vērtējums un zems būtiskums tiek vērtēts ar 0 punktiem, vidējs ar 1 punktu un augsts ar 2 punktiem. Tā kā ne visiem obligāti bija jāsniedz atbildes (t.i. tikai gadījumos, kad bija vēlme kādu no vērtējumiem koriģēt), svarīguma līmeņa aprēķinam tiek izmantots tas skaits, cik tika iesaistīti 3.kārtā.

Anketas un pavadvēstules ir sniegtas šī darba pielikumos.

Ņemot vērā to, ka TraceIt lietotājiem nav sveši trasējamības pamatprincipi, aptaujā netika vaicāts par to, kā lietotāji vērtē trasējamību kopumā.

Lai arī katrā kārtā anketas aizpildīšanai tika dotas 10 dienas laika, daļa respondentu anketas neatsūtīja laikā, un bija nepieciešams atgādināt par sagaidāmo termiņu. Lielās aizņemtības vai citu personisku iemeslu dēļ daļa aptaujā iesaistīto dalībnieku atbildes neatsūtīja. Dalības līmenis aptaujā bija šāds:

1. Pirmās kārtas anketa tika nosūtīta 26 dalībniekiem. Atbildes tika saņemtas no 13 dalībniekiem, kur 12 bija pieturējušies pie anketas jautājumiem, bet viena atbilde bija neliela stāstījuma formā par rīka izmantošanas pieredzi.
2. Otrās kārtas jautājumi tika nosūtīti visiem tiem pašiem 26 pirmās kārtas adresātiem. Uz otrās kārtas jautājumiem atbildēja 12 dalībnieki.

3. Ņemot vērā to, ka trešā kārta ir domāta otrajā kārtā sniegtā vērtējuma precizēšanai, trešajā kārtā respondentu grupa tika ierobežota ar tiem 12 dalībniekiem, kuri atbildēja uz 2.kārtas jautājumiem. Trešajā kārtā savus precizējumus sniedza 8 dalībnieki. Ņemot vērā to, ka jautājumu raksturs bija tāds, ka precizēšana nebija obligāta, tad rezultātu precizējumu aprēķinā rezultāta izmaiņa tika rēķināta nevis proporcionāli 8 dalībnieku balsīm, bet 12 balsīm.

Aptaujas rezultāti

Veidojot anketu, bija jāņem vērā aptaujas dalībnieku aizņemtība. Tās aizpildīšana nedrīkst prasīt pārāk daudz laika, tādēļ jautājumu skaits bija jāpieskaņo tam, lai respondenti varētu orientējoši iekļauties plānotajā 30-60 minūšu laikā.

Anketā iekļautie jautājumi ir nosacīti iedalāmi šādās piecās grupās:

- Par sagaidāmajām īpašībām no trasējamības rīkiem;
- Par TraceIt izmantotajām funkcijām;
- Atziņas no savas pieredzes par TraceIt izmantošanu;
- TraceIt un citu trasējamības pieeju salīdzinājums;
- Ieteikumi un viedokļi par TraceIt potenciālo lietderību citos projektos.

No aptaujas pirmās kārtas jautājumiem tika iegūtas atbildes brīvā formā, kas raksturoja katra dalībnieka viedokli. Starp paustajiem viedokļiem bija gan līdzīgi (bet formulēti atšķirīgi), gan atšķirīgi pēc satura. Jāņem vērā, ka brīvas formas atbildes sniegšanas gadījumā respondents ne vienmēr uzreiz atcerējās vai iedomājās uzrakstīt visu būtiskāko, ko viņš būtu uzrakstījis, veltot tam ilgstošu laiku. Tieši tādēļ svarīga ir otrā kārta, kas ļauj izmantot citu sniegto viedokli savam kvantitatīvajam vērtējumam. Ja pirmā kārta ir domāta viedokļu uzkrāšanai, tad otrā kārta ļauj tos sagrupēt pēc svarīguma.

Trešās kārtas uzdevums ir caurskatīt iegūtos rezultātus un koriģēt viedokli. Šajā aptaujā trešās kārtas rezultātā svarīguma līmenis mainījās 41 viedoklim. Svarīgums tika samazināts vairāk viedokļiem nekā palielināts (attiecīgi 24 un 17). No augsta uz vidēju tika samazināts 16 un no vidēja uz zemu – 8. Savukārt, no zema uz vidēju tika pārcelti 3 viedokļi, no vidēja uz augstu – 14.

Pēc trešās kārtas klāt nāca 10 jauni viedokļi. Neviens no tiem neieguva vērtējumu kā ar augstu svarīgumu. 7 jaunie viedokļi ar zemu svarīgumu, bet 3 - ar vidēju.

Bija jautājumi, par kuriem pirmajā kārtā atsevišķi respondenti nepiedāvāja savus viedokļus, vai arī iesniegtie viedokļi bija stipri līdzīgi. Aplūkojot noslēguma rezultātu, var secināt, ka vismazāk dažādo viedokļu bija jautājumiem par to, kādu darbu veikšanai, kuriem projekta darbiniekiem, kura tipa projektiem TraceIt vissliktāk ir piemērots, kā arī par nepieciešamajiem TraceIt uzlabojumiem un kādas ir MS Excel priekšrocības salīdzinājumā ar TraceIt. Savukārt, citiem jautājumiem viedokļu bija vairāk, un starp tiem bija proporcionāli vairāk augstu vērtējumu. Var secināt, ka

lietotājiem nebija negatīva viedokļa par rīku TraceIt, bet gan bija konstruktīvas vēlmes par papildus iespējām, un tika saskatītas rīka priekšrocības salīdzinājumā ar MS Excel vai parastā teksta risinājumiem.

Atbildot uz jautājumu par to, kādi darbi projektā ir visvairāk atkarīgi no trasējamības, vissvarīgāko vidū tika ierindoti:

- problēmziņojumu apstrāde,
- audits,
- testu sagatavošana un
- problēmu novēršana.

Ar vidēju svarīgumu tika novērtēta testu izpilde, projekta darbu pārņemšana no kolēģiem un atkļūdošana. Savukārt, trasējamības atkarīgi darbi, bet ar mazāku atkarību tika ierindotas prasību specificēšana, implementēšana (kodēšana) un projektēšana.

Respondenti uzskata, ka trasējamība vislielāko labumus sniedz, uzlabojot kvalitāti projektā un ļaujot pārvaldīt projektu. Kā vidēji svarīgi labumi tiek minēts tas, ka darbu pārņēmējam būs vieglāk saprast, uz ko attiecas koda gabali, var labāk veikt kļūdu labojumus, labāk kontrolēt kļūdu labojumus, palīdzēt realizēt prasības programmas koda formā, kā arī tad, ja kāda funkcija ir mainīta vairākkārt. Mazāk svarīgs labums tiek sniegts darbu pārņēmējam, kad kļūst vieglāk saprast, uz ko attiecas testi un prasības.

Pētījuma dalībnieki trasējamības rīkam izvirza virkni būtiskāko prasību:

- Ērtu lietotāja saskarni;
- Jāuztur informāciju par ierakstu veidotājiem vai izpildītājiem;
- Saišu atbalsts/nodrošinājums;
- Izmaiņu pārvaldības atbalstu;
- Brīvu meklēšanu pēc jebkuriem atribūtiem;
- Aptuveni to, ko pašlaik piedāvā TraceIt;
- Automātiskās vēstules darbiniekiem par izmaiņām.

Kā vidēji svarīgas prasības tiek minētas: iespēja saņemt atgādinājumus par termiņiem un iespēju redzēt, kas, kad un ko ir darījis ar konkrētu vienumu. Kā mazāk būtiskas prasības ir ierindota iespēja, lai trasējamība būtu integrēta izstrādes rīkā, kā arī salīdzinājumā ar TraceIt nepieciešams vienkāršāks un caurspīdīgāks sasaistes veids.

Balstoties uz TraceIt lietošanas pieredzi projektos, kuros bija piedalījušies eksperti, viņi vērtēja, kādiem darbiem TraceIt būtu izmantojams darbiem projektā. Visaugstāko novērtējumu ieguvuši šādi projektu darbi:

- Problēmziņojumu (vai izmaiņu pieprasījumu) reģistrēšanai;
- Problēmziņojumu (vai izmaiņu pieprasījumu) apstrādei;
- Projekta statistikai;
- Testu izpildes reģistrēšanai;
- Testu izpildes plānošanai;

- Darba uzdevumu un atgādinājumu reģistrēšanai;
- Projekta vadības informēšanai par konstatēto kļūdu novēršanas gaitu;
- Testu un prasību pārklājuma kontrolei;
- Lai sagatavotos auditam.

Savukārt, pārraudzība un testu pierakstīšana ir darbi, kuros būtu daļēji svarīgi izmantot TraceIt, bet vismazāk būtiska rīka izmantošana tiek vērtēta atkļūdošanas aktivitātēs.

Atbildes uz jautājumiem par rīkā visbiežāk izmantotajām funkcijām liecina, ka tika izmantotas praktiski visas ikdienas funkcijas. Atsevišķu projektu gadījumā mazāk tika izmantota specifiskā ietekmes analīzes funkcija, jo visbiežāk tas tika realizēts ar tiešo saišu aplūkošanu starp diviem vai vairāk vienumiem. Ne visi lietoja statistiku, datu importēšanas un eksportēšanas iespēju. Daļēji tas ir atkarīgs no lietotājam piešķirtajām tiesībām, kas arī raksturo atbilstošā lietotāja darbu loku projektā. Daļa lietotāju nebija droši, ka pārzina visas rīka funkcijas, lai varētu nosaukt tās, kuras izmantoja vismazāk.

Aptaujas dalībnieki vērtē, ka vislabāk TraceIt atbalsta darbu testētāju grupas vadītājam, sistēmanalītiķim, testētājam un projekta kvalitātes pārvaldniekam. Ar vidēji labu piemērotību - projekta pārvaldniekam un programmētājam.

Atbildot uz jautājumu par novērotajām TraceIt rīka nepilnībām, kā visbūtiskākās tiek atzīmētas tādas, kā administratora rokasgrāmatas trūkums, izpalikusī iespēja automātiski izsūtīt e-pastu atbildīgajiem cilvēkiem, ja mainās atbilstošs ieraksts, kā arī problēmas ar visu pieejamo saišu pārskatīšanu vienā ekrānā. Kā vidēji būtiski trūkumi tiek atzīmēts, ka nav lietotāja rokasgrāmatas, nav iespēju meklēt vienumu vēsturiskajā informācijā, varēja būt vairāk iespēju uzstādīt noklusējumus, nevar veikt automātisku e-pastu pārsūtīšanu adresātam, nevar veikt brīvu meklēšanu pēc vārdiem un to kombinācijām, kā arī prasība par filtra nosaukuma unikalitāti. Kā mazāk būtiski trūkumi ir minēti tādi, ka kopā ar rīku netiek doti veiksmīgu projektu modeļu piemēri un ka lietotāji saraksta laukos ir ar reģistrācijas vārdiem, bet nevis ar vārdiem un uzvārdiem.

Eksperti nespēja sniegt daudzveidīgus variantus tam, kuriem projekta darbiem TraceIt būtu vismazāk piemērots. Kā vienīgais variants tika minēta koda rakstīšana. Savukārt, runājot par projekta darbiniekiem, kuriem TraceIt būtu vismazāk piemērots, ar vispārliecinošāk tika minēti tehniskā atbalsta dienesta darbiniekam, vidēji nepiemērots rīks ir dokumentētājam, bet ne visi uzskatīja, ka rīks nebūtu piemērots arī sistēmanalītiķim un programmētājam.

Atbildot uz jautājumu par to, kāda tipa projektiem TraceIt vislabāk būtu piemērots, bija novērojami atšķirīgi viedokļi, sniedzot dažādas projektu raksturojošās īpašības. Vislielākā vienprātība bija par to, ka TraceIt ir izmantojams projektos, kur ir iesaistīti nopietni resursi, tas varētu derēt jebkura izmēra projektiem, bet īpaši vidējiem projektiem, tas ir izmantojams testēšanas projektos un izstrādes projektos, kas ietver produkta pilnu dzīves ciklu, kā arī projektos, kur ir atgriezeniskā saite.

Vidēji piemērots rīks ir lieliem projektiem, projektos ar ūdenskrituma modelis un uzturēšanas projektos. Vismazāk būtiska piemērotība ir prototipēšanas projektu gadījumā. Norādot projektu tipu, kur vismazāk būtu piemērots TraceIt, nebija īpaši dominējoša tipa. Tika piedāvāti varianti par projektiem, kur nenotiek programmēšanas darbi un maziem projektiem, kur ir vieglāk organizēt komunikāciju bez rīku starpniecības. Pie nepiemērota tipa projektiem bija arī tie, kas bija minēti kā mazāk piemērotie – projekti bez atgriezeniskās saites un uzturēšanas vai palīdzības tipa projekti.

Visas priekšrocības, kas trasējamības uzturēšanā ir rīkam TraceIt salīdzinājumā ar teksta vai MS Excel formātu, bija ar augstu vai vidēju svarīgumu. Starp svarīgākajām TraceIt priekšrocībām ir ierindojamas:

- Specializēta programma ar lielākām iespējām;
- Nodrošina daudzlietotāju režīmu;
- Saišu mehānisms;
- Iespējams augsts reaģēšanas ātrums;
- Paaugstina lietotāju disciplīnu;
- Grupas vadītājs var kontrolēt lietotājus;
- Vienkāršo darbu;
- Labāka filtrācija.

Kā vidēji būtiskas priekšrocības ir minēts tas, ka pieejamā informācija ļauj vadīt projekta darbu, rīks automatizē darbu, ir iespēja vieglāk veidot „kvalitātes" atskaites, var uzkrāt failu informāciju, kā arī informācija ir klasificējama un ar filtriem ir aplūkojama dažādos veidos.

Respondenti nenorādīja nevienu augsta būtiskuma pretējo īpašību (MS Excel priekšrocības salīdzinājumā ar TraceIt). Vidēji būtiska ir iespēja mazā projektā visu nepieciešamo uzkrāt vienā tabulā vai ekrānā, kā arī tas, kas brīvi regulējama tabulas veida informācija ir labāk pārskatāma.

Viens no summārajiem lietderības atzinumiem ir eksperta vēlme ieteikt vai neieteikt rīku TraceIt izmantošanai citos projektos. Visstabilākie pozitīvie argumenti ir tādi, ka rīku kā interneta programmu var lietot no jebkuras vietas, darbs ir iespējams vairākiem cilvēkiem reālā laika režīmā, nav IT uzņēmuma ietvaros jāiegādājas licences, ir iespēja pielāgot projekta vajadzībām, interneta programmu var lietot bez sistēmas konfigurēšanas, kā arī ir lielākas iespējas par MS Excel risinājumu. Mazāka nozīme ir tam, ka rīkam ir laba atbalsta komanda, tas ir pārbaudīts darbā lielā projektā, kā arī to var izmantot ne tikai trasējamībai, bet arī datubāzes prototipa sagatavošanai. Nebija augsta svarīguma argumentu, lai neieteiktu TraceIt izmantot citos projektos. Ar vidēju svarīgumu bija raizes par to, ka nav skaidra uzņēmuma nostāja šī rīka tālākajai attīstībai (jo tas ir iegūts kā eksperimentālas pētniecības rezultāts). Mazāk būtiski argumenti rīka neizmantošanai tika norādīti tādi, ka mazu projektu gadījumā nav obligāti uzturēt trasējamību, kā arī tīmekļa programma lēnāka ātrdarbība par lokāli uzstādītām programmām.

Kā vēlamos uzlabojumus TraceIt lietotāji norāda nepieciešamību uzlabot saskarni, lai būtu vienkāršāk sasaistīt vienumus, kā arī paplašinātas grafiskās iespējas aplūkot saites starp vienumiem un mehānisms PZ (vai citu vienumu) lietotāja izvēlētai kārtošanai un grupēšanai.

Lietotāji atzīmē iespēju ātri izveidot datubāzi ar vajadzīgajiem vienumiem, un projektam jau to tieši to vajag – netērēt laiku palīgrīka vai darba vides sagatavošanai. Cits risinājums ir izmantot jau gatavu rīku, kurš nav jākonfigurē, bet tad projektam ir jāpielāgojas tieši šim rīkam.

Tas, ka lietotāji izvirzīja trasējamības rīkam prasības, kas būtībā neattiektos uz trasējamību, bet gan vairāk uz projekta darbu, liecina, ka projektiem ir nepieciešams tehniskais atbalsts ne tikai trasējamībai, bet arī informācijas pārvaldībai kopumā.

7.7. TraceIt ieviešanas ekonomiskie apsvērumi

TraceIt aptaujas ietvaros aptaujātie eksperti bija vienisprātis, ka nav jēgas izmantot vienkāršu tekstu kā trasējamības pieraksta formu, jo pierakstītā informācija pēc tam ir grūti izmantojama – jāpatērē proporcionāli daudz papildus laika. Populārākais risinājums ir biroja programmatūras, bet it īpaši elektronisko tabulu, piemēram, MS Excel programmatūras, izmantošana. Kā zināms, šī ir universāla tipa programmatūra, kas nodrošina daudzveidīgu darbu ar tabulas tipa informāciju. Tabulas šūnās var pierakstīt tekstuāla tipa informāciju, un citas šūnas izmantot klasifikācijas un papildinformācijas uzkrāšanai. Būtisks ierobežojums Excel izmantošanai ir tas, ka ar elektronisko tabulu vienlaikus var strādāt tikai viens darbinieks. Tas nozīmē, ka piemēram, problēmziņojumu reģistrēšanas gadījumā katram testētājam ir jāaizpilda savs fails, bet dienas noslēgumā tos kādam ir jāapvieno centrālajā failā vai arī tie jāsinhronizē pēc noteikta algoritma. Šādu pieeju autors ir novērojis vairākos projektos, un konkrētajās situācijās šāds manuālais darbs tika novērtēts kā labākais pieejamais risinājums. Priekšrocība elektroniskajām tabulām varētu būt īpaši ātra risinājuma iegūšana (t.i. bez jebkādas specializētas programmas izmantošanas), ko bez īpašām problēmām var izmantot viens cilvēks.

Lai arī TraceIt ieviešanai kā galvenie nebija izvirzīti konkrēti ekonomiskie apsvērumi, bija skaidrs, ka rīka lietošanai ne tikai jāuzlabo projekta trasējamības tehniskās iespējas, bet tai jābūt arī ekonomiski izdevīgai projekta saimniecībā. No finansiālā viedokļa elektronisko tabulu programmatūra nerada papildus izmaksas – tā ietilpst vairākos populāros biroju programmu komplektos. Ja specializētā rīka cena pārsniegs projekta vai organizācijas finansiālās iespējas, tad komerciālos darbības apstākļos tiks izvēlēts lētāks variants, pat ja tas neapmierinātu visas tehniskās un kvalitātes prasības (protams, saglabājot darbošanos atbilstoši uzņēmumā definētajām minimālajām kvalitātes prasībām). Darbā aplūkotā TraceIt ieviešana projektiem bija bez papildus izdevumiem, jo tas tika izstrādāts IT uzņēmuma ietvaros kā eksperimentāls produkts. Ja tiktu apsvērta iespēja šo rīku tirgot citiem uzņēmumiem

vai rīka uzturēšanai piesaistīt līdzekļus no pašu projektiem, tad būtu jāņem vērā divi galvenie apsvērumi: (1) izstrādē un uzturēšanā ieguldītās izmaksas un (2) projekta iespējas apmaksāt rīka lietošanu. Rīka lietošanas licenču cenai jābūt tādai, lai iegādāto licenču summa pārsniegtu izstrādes un uzturēšanas izmaksas. Tā kā projektu piesaiste un cenu noteikšana ir tirgvedības nozares jautājums, tad sīkāk šie jautājumi šajā darbā netiks analizēti. Ja viens no rīka lietošanas mērķiem ir samazināt projekta izmaksas, tad licences cenai ir jābūt zemākai par to summu, ko ir iespējams ietaupīt, ja MS Excel vietā darbam izmanto TraceIt. Ekonomiskais izdevīgums ir viens no kritērijiem šī rīka ieviešanai. Būtiskākais rādītājs ir laiks, kas parasti ir arī noteicošais rādītājs projekta izmaksām. Ja ir iespējams iegūt laika novērtējumu, tad ir iespējams spriest par projekta ietaupītajām vai zaudētajām izmaksām.

Lai veiktu novērtējumu, tiks aplūkoti divu projektu piemēri – ar 4 un 25 cilvēkiem. Šie ir teorētiski projekti, kas pēc savas struktūras un darba stila veidoti atbilstoši reāliem izstrādes projektiem. Vispirms aplūkosim aktivitātes, kuras pēc darbietilpības apjoma ir ar vienādu darba apjomu MS Excel vidē un TraceIt rīkā. Vienumu tipu tabulu formāta definēšana TraceIt rīkā aizņem vairāk laika nekā Ar MS Excel palīdzību. Prakse rāda, ka minētās lietas projekta vajadzībām var sagatavot viena persona 4 stundu laikā. Lietotāju tiesību definēšana prasa papildus 4 stundas. Prasību, testu vai funkciju saraksta izveide abās vidēs ir līdzvērtīga. Arī tad, ja TraceIt vajadzībām sākotnējo sarakstu sagatavo MS Excel formātā, to ir iespējams dažu minūšu laikā ieimportēt TraceIt rīka datubāzē. Salīdzināšanai būtiskākais ir ikdienas darbos patērētais laiks. Individuālo aktivitāšu apjoma novērtējums ir balstīts uz autora veiktajiem novērojumiem reālos projektos. Abi projekti tiek aplūkoti laikā, kad ir veikta liela daļa implementācijas, ir sākusies testēšana, tiek gatavotas daļēji gatavas versijas, ar kurām tiek iepazīstināts pasūtītājs, lai apspriestu to atbilstību sākotnējām iecerēm. Novērtējumam izmantotais laika periods – 5 darba dienas jeb 1 darba nedēļa.

Projekta darbinieku slodzes sadalījums pa lomām ir parādīts 23.tabulā.

23.tabula

Loma	4 cilv. projekts	25 cilv. projekts
Projekta pārvaldnieks	0,5	1
Sistēmanalītiķis	1	4
Programmētājs	2	8
Testētājs	0,25	5
Dokumentētājs u.c.	0,25	2

Novērtējumam ir izvēlētas aktivitātes, kas tipiski tiek veiktas projekta laikā, kad notiek implementācija un testēšana. Saskaņā ar 24.tabulā iegūtajiem rezultātiem, būtiskākie ieguvumi parādās pie operācijām, kas prasa informācijas apvienošanu un

kopīgās informācijas analīzi. Jo lielāks ir projekts, jo vairāk cilvēku informācija ir jāapkopo un jāanalizē. Tieši šādās situācijās TraceIt uzrāda labu ieguvumu.

24.tabula.

	Operācija	Patērētais laiks (minūtes)		4 cilvēku projekts		25 cilvēku projekts	
		Excel	TraceIt	reizes/ 5dienās	cilvēku skaits	reizes/ 5dienās	cilvēku skaits
1	Aplūkot prasību sarakstu	10	8	5	1	5	1
2	Aplūkot testu sarakstu	10	8	5	1	5	1
3	Pievienot jaunu testu	25	25	1	1	1	2
4	Aplūko funkciju sarakstu	10	8	5	1	5	6
5	Reģistrēt problēmziņojumu	15	15	8	1	8	5
6	Aplūkot problēmziņojumu sarakstu	15	12	5	1	5	8
7	Reģistrēt testēšanas rezultātu	3	3	10	1	10	5
8	Apkopot testētāju pieteiktos problēmziņojumus vienā sarakstā	30	0	2	1	2	5
9	Apkopot testu izpildes rezultātus vienā sarakstā	30	0	2	1	2	5
10	Novērtēt izpildīto testu pārklājumu	20	5	4	1	4	2
11	Statistikas sagatavošana	30	15	3	1	3	2
12	Noteikt, kādas funkcijas ietekmēs konkrētas prasības maiņa	15	3	2	1	2	2

Būtiskākā laika starpība konkrētām operācijām ir tajās vietās, kur ir jāveic informācijas apkopošana, meklēšana un analīze. Saskaņā ar tabulā veiktajiem novērtējumiem, 4 cilvēku projektā nedēļas laikā darbam ar Excel tiek veltītas 720 minūtes, bet to pašu uzdevumu veikšanai ar TraceIt – 426. Starpība ir 294 minūtes jeb 4,9 stundas. Savukārt, 25 cilvēku projekta gadījumā Excel darbi patērē 2800 minūtes, bet TraceIt – 1742. Starpība vienas darba nedēļas gadījumā ir 1058 minūtes jeb 17,6 stundas. Varam secināt, ka TraceIt ļauj ietaupīt projekta patērēto darbietilpību.

Jāņem vērā, ka ekonomiskā novērtējums balstās uz teorētiska modeļa, un praksē katrā konkrētā projektā var parādīties savas nianses. Novērtējumā nav ņemti vērā faktori, kas tiešā veidā neparādās kā patērētā laika ieguvums. Ja trasējamības, testu izpildes, problēmziņojumu un cita projekta informācija ir labi pieejama, projekta darbinieki labprātāk ar to strādās – pētīs, precizēs, reģistrēs utt. Tieši tas ļaus projektam strādāt labāk, jo darbinieku rīcībā būs korekta informācija. Piemēram, ja ir zināms, ka testu izpildes rezultāti tiek apkopoti tikai katras otrās darba dienas beigās, un to pārvalda viens cilvēks, citiem potenciālajiem interesentiem par testu izpildes statusu vienmēr būs pieejami dati tikai ar nokavēšanos (tajā skaitā tam, kurš šo apkopojumu veic). Savukārt, operatīvi pieejamu pārskatu iegūšanas gadījumā visiem interesentiem ir iespēja novērtēt darbu progresu un veltīt uzmanību problēmapgabaliem.

7.8. Trasējamības kvalitāte ar un bez speciālā rīka

Trasējamības rīka TraceIt nostādņu apraksts un rīka lietotāju aptaujas rezultāti ļauj veikt trasējamības procesa kvalitātes salīdzinājumu divos gadījumos – projektā bez specializēta rīka lietošanas (bet tiek izmantoti standarta līdzekļi tādi, kā elektroniskās tabulas) un ar specializēta rīka pielietošanu. Salīdzinājums ir sniegts 25.tabulā. Aplūkotas ir situācijas, ka projektā tiek mērķtiecīgi domāts par trasējamību, bet atšķirošā pazīme ir specializēta rīka esamība vai iztrūkums.

25.tabula.

Kritērijs	Bez trasējamības rīka	Ar trasējamības rīku	Pamatojums
Vienumu aptvērums	Niecīgs (1) vai Vidējs (2)	Plašs (3)	Bez trasējamības rīka ir problemātiski nodrošināt plašu trasējamības uzskaiti, jo tas prasa lielu informācijas apstrādi.
Iesaistītie darbinieki	Viens (1) vai Daži (2)	Daži (2) vai Visi (3)	Rīka lietošana ļauj to izmantot vairākiem cilvēkiem līdz pat tam, ka to izmanto visi projekta dalībnieki.
Uzglabāšanas vieta un metode	Teksta dokumenti (1)	Dokumenti un rīks (2) vai Rīks (3)	Uzglabāšana rīkā nodrošina informācijas integritāti. Ja tā ir saskaldīta pa dokumentiem, zūd iespēja to sinhronizēt. Vidējs risinājums ir uzturēt rīkā saišu informāciju, bet dokumentos uzglabāt vienumu saturisko daļu.
Trasējamības mērķis	Izpildīt formālās prasības (1) vai Atvieglot darbu (2)	Izpildīt formālās prasības (1) vai Atvieglot darbu (2)	Rīka lietošana vai nelietošana neiespaido projekta attieksmi pret trasējamības mērķi
Informācijas reģistrēšanas regularitāte	Ar laika intervālu (1)	Vienlaicīgi ar vienuma izmaiņām (2)	Rīka izmantošana ļauj reģistrēt informāciju vienlaikus ar notikumu.

No salīdzinājuma var secināt, ka bez rīka lietošanas trasējamības kvalitāti var vērtēt ar 5-8 punktiem, bet ar trasējamības rīka lietošanu – no 10 līdz 13 punktiem. Minētais salīdzinājums pamato darbā izvirzītā apgalvojuma (b) daļu - trasējamības kvalitāti var uzlabot ar trasējamību atbalstoša rīka palīdzību.

7.9. Divu testēšanas procesu kvalitātes salīdzinājums

Ņemot vērā iepriekš aprakstītās īpašības, kuras programmizstrādes procesiem piešķir rīks TraceIt, ir iespēja novērtēt testēšanas procesa kvalitāti diviem gadījumiem: (1) elektronisko tabulu izmantošana un (2) TraceIt izmantošana. Vērtējumā tiks izmantoti 2.7. nodaļā aprakstītie kritēriji. Abos gadījumos tiek aplūkota situācija, ka procesā ir iesaistītas vairākas projekta personas. Jāņem vērā, ka viena vai otra tehniskā risinājuma lietošana pati par sevi nenodrošina kvalitatīvus rezultātus. Projektam ir jābūt ar atbilstošajām zināšanām un jāapzinās mērķi, kurus nepieciešams sasniegt. 26. un 27. tabulās ir sniegts augstākais sasniedzamais rezultāts, izmantojot tehniskā risinājuma iespējas.

26.tabula. Testēšanas procesa kvalitāte elektronisko tabulu lietošanas gadījumā.

Kritērijs	Saistīto aktivitāšu raksturojums	Līmenis
Informācijas pieejamība	Informācija ir pieejama atsevišķos failos vai tabulās. Ja ir iesaistīti vairāki faili, tad problēmas var parādīties ar saitēm starp atsevišķu failu elementiem.	vidējs (1)
Operativitāte	Vairāku projekta darbinieku gadījumā ir nepieciešams informāciju apvienot, sinhronizēt. Ar elektronisko tabulu palīdzību to nav iespējams veikt uzreiz pēc katras izmaiņas veikšanas. Starp notikuma vai fakta reģistrēšanu un atbilstošās informācijas pieejamību veidojas nenoteikts laika intervāls. Īpaši kritiski šī problēma izpaužas gadījumos, ja ir jāveic problēmu pieteikšana, apstrāde un pārbaude – var veidoties nevēlami ilga labojumu kavēšanās, kuru paātrināt ir problemātiski tieši informācijas apvienošanas procesa dēļ.	zems (0)
Pilnīgums	Elektroniskās tabulas neuzliek ierobežojumus uzkrāt tik pilnīgu testēšanas informāciju, cik tas nepieciešams testēšanai.	augsts (2)

27.tabula. Testēšanas procesa kvalitāte TraceIt lietošanas gadījumā.

Kritērijs	Saistīto aktivitāšu raksturojums	Līmenis
Informācijas pieejamība	Ja lietotājam ir atbilstošas tiesības, tad ir pieejama visa nepieciešamā informācija	augsts (2)

	dažādos skatījumos.	
Operativitāte	TraceIt attēlojamā informācija atbilst aktuālākajam stāvoklim.	augsts (2)
Pilnīgums	TraceIt var uzkrāt visu tipveida testēšanai nepieciešamo informāciju.	augsts (2)

Jāņem vērā, ka var vērtēt tikai tajās robežās, cik tālu aplūko procesu. Var eksistēt vēl citi kritēriji, kas nav tiešā veidā saistīti ar izvēlētā rīka lietošanu. No 26. un 27.tabulās var iegūt salīdzinājumu, ka elektronisko tabulu lietošanas gadījumā testēšanas procesu var vērtēt ar 3 punktiem, bet trasējamības rīka TraceIt gadījumā – ar 6 punktiem no 6 iespējamiem. Var secināt, ka trasējamības rīka lietošana var nodrošināt testēšanas procesa uzlabojumu – procesa kvalitātes kritēriji ir ar augstāku līmeni nekā ikdienas prakses gadījumā (konkrēti, izmantojot elektroniskās tabulas). Iepriekšminētais kalpo par pamatojumu darbā izvirzītā apgalvojuma (c) daļai - ar trasējamību atbalstošu rīku testēšanas process ir kvalitatīvāks nekā bez tā.

7.10. Nodaļas secinājumi

Nodaļā tika aplūkota TraceIt ieviešanas pieredze dažāda tipa projektos viena IT uzņēmuma ietvaros. Ieviešanai tika izmantota pakāpeniska pieeja, sākotnēji to izmēģinot paša rīka izstrādes projektā, vēlāk 2 atsevišķos projektos, un tikai pēc tam – pārējos interesi izrādījušos projektos. Paralēli ekspluatācijai tika veidotas jaunas TraceIt versijas, kurās bija konstatēto kļūdu labojumi un uzlabojumi funkcionalitātē vai lietotāja saskarnē. Pakāpeniska ieviešana bija veiksmīga, jo tā samazināja projektu darba pārtraukuma risku rīka problēmu gadījumā.

No līdzšinējās ekspluatācijas ir konstatēts, ka rīks ir labi piemērots vidēja izmēra projektiem, kā arī testēšanas procesa atbalstam. Lietotāju-ekspertu aptauja liecina, ka lietotāji kopumā ir bijuši apmierināti ar rīka funkcionalitāti. Ir izteiktas piezīmes par atsevišķiem implementācijas risinājumiem, bet kopumā rīks ir veicis nozīmīgu lomu projekta darba nodrošināšanā.

Nodaļas ietvaros aplūkotie jautājumi saistās ar darbā izvirzītā apgalvojuma (b) un (c) daļām – ar TraceIt kā trasējamības rīku var uzlabot trasējamības kvalitāti un testēšanas procesu, jo ir iespējas paaugstināt informācijas apmaiņas operativitāti, nodrošināt tās pilnīgumu visiem testēšanas procesa dalībniekiem, kā arī sniegt pieeju visai ar procesu saistītajai informācijai – testu sagatavošanai, izpildei, problēmu pieteikšanai un apstrādei.

Lai arī rīks TraceIt bija veidots tieši kā trasējamības rīks, tas ļauj projektam apstrādāt dažāda tipa darba informāciju tā, ka ar tā palīdzību var aizvietot tabulveida informācijas reģistrus, bet īpaši labi tas ļauj aizvietot problēmziņojumu pārvaldības rīkus. Šādu iespēju ir izmantojuši vidēji lieli projekti.

8. Prasības pret trasējamības atbalstu projektos

8.1. Nodaļas mērķi

Balstoties uz TraceIt izstrādē un ieviešanā gūtajām atziņām, autors ir definējis prasības trasējamības atbalstam programmatūras projektos. Viens no būtiskākajiem secinājumiem ir tas, ka rīks TraceIt ir uzskatāms par izmantojamu piemēru problēmas risinājumam, kur projektiem nav bijis pietiekama tehniskā atbalsta trasējamībai. Tajā pašā laikā autors ir konstatējis, ka labos programmizstrādes apstākļos projektiem būtu jāspēj iztikt bez specializēta trasējamības rīka – attiecīgajām iespējām jābūt integrētām citos programminženierijas rīkos.

Iepriekšminētais izriet kā daļa no notikumu attīstības paradigmas, kas novērojama daudzās dzīves situācijās kopā ar jaunu lietu ieviešanu:

1. Ir nobriedusi problēma, un parādās tehniskas iespējas to risināt.
2. Pirmie risinājumi veic savu uzdevumu, bet tie ir apjomīgi, no lietotāja prasa papildus darbu.
3. Pateicoties atziņām no pirmajiem paraugiem un jaunu tehnisko iespēju attīstībai, jaunais risinājums tiek izveidots tāds, ka tas kļūst ikdienišķs un nemanāms.

Tieši saskaņā ar šo paradigmu autors vēlās redzēt trasējamības rīku attīstību. Šajā nodaļā ir aprakstītas prasības nosacīti trešajam attīstības posmam.

8.2. Trasējamība kā integrēta īpašība

Šajā darbā aplūkotā TraceIt lietošana projektos ir uzskatāma analoģiska situācijai, kad projektam, kuram ir problēmas, tiek iedoti vitamīni tablešu formā, lai viss noritētu kvalitatīvi. Normālā situācijā projektam vitamīni būtu jāuzņem ikdienā no citiem lietojamiem produktiem (projekta rīkiem). Ja uzturs (izmantojamais rīku komplekts un konfigurācija) ir sabalansēts, nerodas nepieciešamība pēc papildus stimulējošām vai atbalstošām lietām. Mākslīgi sagatavotus vitamīnus var izmantot līdz tam brīdim, kamēr netiek atrasta iespēja tos iegūt dabiskā ceļā. Bieži šāda risinājuma meklēšana var prasīt ilgāku laiku. Ja projekts savā apkārtnē neredz labākus trasējamības risinājumus, tad tam ir minimālas iespējas uzlabot savu trasējamību. Projektiem parasti nav laika nodarboties ar novatoriskiem risinājumiem, bet, balstoties uz pieejamajiem līdzekļiem, ir jāstrādā pie savu pamatuzdevumu izpildes.

Kvalitātes nodrošinātājiem nav jāsoda projekti par to, ka tie var piedzīvot vienu vai otru neveiksmi. Tiem ir jāatbalsta problēmas izjūtošais projekts. Un nākošā projekta etapā vai cita projekta gadījumā jau tiks ņemtas vērā

Kā jau bija minēts iepriekš, projektā tiek izmantoti rīki tādiem uzdevumiem, kā plānošanai, prasību specificēšanai, modelēšanai, kodēšanai, versiju kontrolei, konfigurācijas pārvaldībai, problēmu apstrādei, dokumentu vadībai, komunikācijai un

citiem programmatūras projekta ikdienas uzdevumiem. Lai arī saskaņā ar atsevišķām taksonomijām eksistē arī trasējamības rīku kategorija, autors uzskata, ka šāda tipa rīki būtu nepieciešami tikai atsevišķu projektu gadījumos – tādos, kur netiek izmantoti prasību pārvaldības, modelēšanas rīki un problēmziņojumu pārvaldības rīki. Lai gan daudzos gadījumos viena rīka lietošana ietekmē cita rīka nepieciešamību un izmantošanas formu, ne vienmēr tas tā ir. Piemēram, trasējamības rīka nepieciešamību neietekmē, testēšanas rīka lietošana.

Lai arī trasējamības īpašību var integrēt rīkā tā, lai tā kļūst nemanāma, jāņem vērā, ka ne visi projekti vēlas strādāt sava veida slēptas trasējamības apstākļos. Rīkiem vai rīku kopai jāspēj precīzi sniegt informāciju par uzturamo trasējamības modeli, vienumu tipiem, saišu tipiem. Jāņem vērā, ka ne vienmēr konkrētais rīks satur sevī visu tos projekta informācijas tipus, kas pakļaujas trasējamībai. Šajā nolūkā ir nepieciešama saskarne starp dažādiem rīkiem, kas ļauj no viena rīka nodrošināt saiti ar citā rīkā uzturamu vienumu. Universāls risinājums šādai prasībai neeksistē, jo katrs rīks ir unikāls, un problēmas var rasties ne tikai tehniskā, bet arī metodiskā līmenī. Ja pat saites starp dažādās vidēs esošiem vienumiem ir iespējams nodrošināt informatīvu atsauču līmenī, tad tas nerisina informācijas sinhronizācijas problēmu – kā nodrošināt to, ka izmaiņas vienā rīkā izsauc izmaiņu nepieciešamību citā rīkā esošajos vienumos? Šīs un citas datu saderības un saskaņotības problēmas būtu risināmas projekta globālās trasējamības nodrošināšanas gadījumā.

8.3. Konfigurējamība un dabiskums

Lai arī projekta mēroga trasējamība var prasīt dažādu rīku izmantošanu, ir jāpastāv iespējai vienotā veidā nodefinēt projekta trasējamības modeli. Projekts ir dinamisks pasākumu komplekss, kur var mainīties uzdevuma nostādne, pielietotās metodes, resursi un termiņi. Rīkam vai rīku kopai jāspēj nodrošināt tas, lai trasējamības modelis būtu konfigurējams. Konfigurēšana nozīmē to, ka var mainīt tam piederošos vienumu tipus un pieļaujamās saites starp tiem. Ja konfigurēšana nav iespējama, trasējamība nedzīvo līdzi projekta vajadzībām.

Īpašs skatījums ir lietotāju konfigurācijas. Katram lietotājam vai lietotāju grupai var būt nodefinēts savs skats uz trasējamības modeli. Lietotājam nav jāizjūt tas, ka viņš kaut kā īpaši uztur trasējamību. Katrs projekta dalībnieks veic kādu noteiktu darbu mērķa sasniegšanai. Mērķis ir izveidot programmu, sagatavot projektējumu, notestēt sistēmu, bet ne īpaši uzturēt trasējamības informāciju.

Hipotētiskajai jaunajai trasējamības sistēmai ir jāpiedāvā lietotājam pieņemamu un viegli uztveramu informāciju. Lietotājam jāveic izvēle, un tas automātiski nozīmē saites nodefinēšanu starp vienumiem. Ja lietotājs maina kādu no vienumiem, automātiski tiek apzināti maināmie vienumi.

Cita iespējamā pieeja ir minimāli likt lietotājam operēt ar saišu jēdzieniem, bet gan novērot un analizēt dažādās vietās esošo informāciju, kurai izmanto, apstrādā vai

rada lietotājs. Lietotājs veic darbus ar projektējuma vienumiem, bet ir novērojošais mehānisms analizē kopīgo pēc atslēgvārdiem vai citām pazīmēm. Ja darbs organizēts iterācijās, tad pēc katras iterācijas sistēma lietotāju informē par identificētajām saitēm. Varētu būt droši zināmās saites, kas izriet no apstiprināta rakstura informācijas, kā arī tādas saites, kas ir ar potenciālo saišu statusu – t.i. sistēma tās ir identificējusi, bet lietotājam tas vēl jāapstiprina.

Lai arī kā matemātiski un tehniski domājoši cilvēki vēlētos strādāt tikai ar formalizētām un labi strukturētām prasībām un citiem projekta dokumentiem, daļa dokumentu vienmēr paliks formulēti cilvēku valodā gadījumos, ja programmatūra ir orientēta nevis uz saskarnes realizāciju ar citu sistēmu, bet gan cilvēka darba atbalstam.

Sistēmas sākotnējās prasības parasti formulē nākošie programmatūras lietotāji, pēc tam sistēmanalītiķi izvērš un precizē šīs prasības, noskaidrojot no potenciālajiem lietotājiem papildus vēlmes. Tā rezultātā top dokuments, uz kura pamata tiek izstrādāta sistēma. Parasti šis dokuments satur daudz vairāk prasību nekā bija aprakstīts sākotnējā dokumentā. Šis dokuments tiek saskaņots ar pasūtītāju. Tālākie izstrādājamie dokumenti tādi kā projektējums, sistēmas darbības modeļi u.tml. lietotājiem ir grūti uztverami, tādēļ bieži vien to saskaņošana ir formāla. Lietotājs bieži jau implementācijas laikā (īpaši tad, kad ir aplūkojams kaut kas darbināms), konstatē, ka viņam ir nepieciešami vēl vairāki papildinājumi.

Mēs labi zinām, ka klasiskajā inženierijā, piemēram, būvniecībā, izmaiņas pēc projekta apstiprināšanas parasti netiek veiktas, vai arī tas notiek ar nosacījumu, ka tiek apstādināti visi darbi, un viss mainītais iziet caur nepieciešamajiem attīstīšanas etapiem. Kopumā būs aktuāli tas, ka informācija būs cilvēku lasāmā valodā. Nav iespējams izvairīties no tā, ka lasāmā valoda var būt diezgan bagātīga, ar daudziem sinonīmiem, kā arī var saturēt pretrunas. Trasējamības dabiskums būtu tad, ja dabiskās valodas tekstu varētu efektīvi un bez liekas piepūles no lietotāja puses sasaistīt ar programmas jēdzieniem.

Kopumā var definēt, ka dabiskums ir tad, ja:

- lietotājs par trasējamību nedomā, bet tā tiek uzturēta,
- lietotājs vajadzības gadījumā var painteresēties par trasējamību, bet citādi sistēmas informācijas atlase pati par sevi jau balstās uz to, ka pieejama ir tikai kontekstam atbilstošā informācija.

Ja iepriekšminēto var nodrošināt jaunās paaudzes rīks, tad var uzskatīt, ka tas labi integrējas lietotāju darbā.

8.4. Saistība ar pamatprocesu

Būtiska nozīme ir ne tikai tam, lai rīks būtu konfigurējams un tā lietošana būtu dabiska, bet arī tam, lai tā izmantošana atbilsto projektā noteiktajai kārtībai.

Trasējamības informācija ir jāizmanto pamatprocesa vadīšanā, kā arī informācijai par trasējamību jānāk no pamatprocesa.

Vislielākā integrācija būtu nepieciešama ar konfigurācijas pārvaldības rīkiem. Tas nozīmētu, ka tiek pārvaldīta vertikālā un horizontālā trasējamība. Trasējamībai jābūt individuālo vienumu līmenī un trasējamība apvienoto nodevumu līmenī (vienumu kopa ar fiksētām versijām).

Būtiski ir, lai ar rīku strādā visi projekta darbinieki. Svarīgi ir ar produkta saistītās informācijas izmaiņu notikumus dokumentēt vienā vienīgā vietā, un tam ir jāizsauc izmaiņu ietekmes pārbaude atbilstoši trasējamības modelim. Ja ar rīku nestrādās personas, kas atbild par kāda vienuma tipa uzturēšanu, tad var faktiski šo vienumu tipu var neaplūkot kā trasējamības modeļa sastāvdaļu, jo tas nenodrošinās iespēju trasēt no/uz attiecīgajiem vienumiem – to nebūs vai arī tie nebūs aktuāli.

Trasējamība kļūst sarežģītāka tajās situācijās, kad ir jāsasaista dažādos terminos, stilos un vidēs veidotu informāciju. Konkrēti, saskarne pasūtītājs-izstrādātājs ir samērā problemātiska, jo lietotājam ir nepieciešami formulējumi, kas atbilst viņa darbības sfēras terminoloģijai, bet izstrādātājiem analoģiski tam, kā precīzi tiek veidotas datorprogrammas, būtiski ir precīzi tehniski aprakstīt stāvokli.

8.5. Nodaļas secinājumi

Lai arī viena no autora izvirzītajām tēzēm ir par to, ka TraceIt būtu uzskatāms kā viens no iespējamiem trasējamības problēmu risinājumiem, un labos projekta apstākļos tas nebūtu jāizmanto, bet gan jānodrošina trasējamība ar citiem projektā izmantojamajiem līdzekļiem, ir jāsecina, ka alternatīvu risinājumu ieviešana saskartos ar dažādām problēmām. Galvenais apgrūtinājums ir tādu saskarņu nodrošināšana starp rīkiem, kas sinhronizē informāciju par vienumu stāvokli.

Tādēļ projektos var eksistēt divu veida noteicošie risinājumi – (1) specializēta trasējamības rīka izmantošana, kas ietver sevī plašas dažāda tipa informācijas uzkrāšanas un apstrādes iespējas; (2) projektā izmantojamā produkta izstrādes rīku kopa ir integrējama un nodrošina vienota veida trasējamību.

9. Nobeigums

Darba mērķis bija raksturot testēšanas procesa specifiku un izpētīt, kā testēšanas procesu ietekmē trasējamība. Praktisks izvirzīts mērķis bija parādīt, kādas ir iespējas uzlabot trasējamību jebkura tipa programmatūras projektā ar specializētas programmatūras jeb rīka palīdzību.

Testēšanas problēmu ietvaros tika aplūkoti testēšanas procesa kvalitātes jautājumi reālos programmatūras izstrādes projektos. Papildus testēšanas saistībai ar trasējamību autors pievērsa uzmanību arī testēšanas organizatoriskajiem jautājumiem, īpaši raksturojot situāciju Latvijas IT uzņēmumos. Secinājumos ir izmantota autora pieredze un pētījumu rezultāti, kā arī citu trasējamību pētījušo autoru atziņas. Autora pieredzes pamats veidojies, darbojoties IT uzņēmuma kvalitātes sistēmas elementu izstrādē un uzturēšanā, kā arī reālu projektu atbalstīšanā ar testēšanas metodiskajiem jautājumiem. Praksē gūtā pieredze sniedza atziņas par reālajām projektu problēmām un to iespējamajiem risinājumiem.

Darba ietvaros ir formulētas tipiskākās testēšanas procesa norises un trasējamības problēmas, kā arī izvirzīts no trijām daļām sastāvošs apgalvojums par testēšanas un trasējamības savstarpējo saistību. Darba ietvaros ir sniegti identificēto problēmu risinājumi un ir pamatots apgalvojums ar autora veiktajiem pētījumiem un trasējamības rīka TraceIt lietošanu.

Apgalvojuma (a) daļa par to, ka testēšanas process ir atkarīgs no trasējamības kvalitātes tiek pamatota ar testēšanas definīciju analīzi un testēšanas procesa pamatuzdevumu novērtējumu. Tas, ka trasējamības kvalitāti var uzlabot ar trasējamību atbalstoša rīka palīdzību (apgalvojuma (b) daļa), tiek pamatots pēc trasējamības rīka izstrādes, tā ieviešanas un lietotāju aptaujas. Savukārt, apgalvojuma (c) daļa par to, ka ar trasējamību atbalstošu rīku testēšanas process ir kvalitatīvāks nekā bez tā, tiek pamatota ar trasējamības rīka sniegtajām īpašībām testēšanas procesam.

Kā būtisks praktisks rezultāts ir raksturojams rīks, kas atbalsta ne tikai testēšanas procesu, bet arī ir veiksmīgi izmantojams arī citos programmatūras procesos, jo saistība ar testēšanu tam nav stingra, bet gan izrietoša no konkrētās konfigurācijas. Autors ir pārliecināts, ka rīka izmantošana programmatūras izstrādes un testēšanas projektos sniedz atbalstu un rada lietotāju gandarījumu par iegūto atbalstu. Rīks ir konfigurējams ar metamodeļa palīdzību, un katrā projektā tas ir pielāgojams konkrētā projekta prasībām.

Rīks sākotnēji tika pielietots TraceIt izstrādes grupā, pēc tam 2 pilotprojektos, un vēlāk vēl 12 projektos, kuru pieredze ir analizēta detalizētākā formā. Pēc minēto projektu pieredzes apzināšanas rīku uzsāka izmantot 5 projektos. TraceIt lietotāji ir snieguši savu vērtējumu par to īpaši organizētā aptaujā pēc Delfi metodes.

Darbā ir iegūts teorētisks un eksperimentāls skatījums, un galvenais rezultāts ir reālos projektos strādājošs programmatūras rīks trasējamības atbalstam, kas

eksperimentāli demonstrē trasējamības automatizācijas/atbalsta pieeju, kas uzlabo testēšanas procesu un projekta darbu kopumā. Trasējamības uzturēšanas pieeju var aplūkot ne tikai kā tieši konkrēti izstrādāto programmu, bet tajā ietvertos principus integrēt citos programmatūras izstrādes rīkos.

Pētījumu rezultāti par programmatūras kvalitāti, testēšanu un trasējamību ir atspoguļoti rakstos recenzējamos starptautiskos rakstu krājumos vai konferenču materiālos, kā arī referēti starptautiskās konferencēs.

Autors pateicas RITI par atbalstu eksperimentālo pētījumu veikšanai, Latvijas Zinātnes padomei par piešķirto grantu darba izstrādes periodā. Autors pateicas arī kolēģim Mihailam Bogdanovam, kurš veica lielāko daļu no rīka TraceIt implementācijas darbiem.

10. Bibliogrāfiskais saraksts

10.1. Autora raksti recenzējamos starptautiskos rakstu krājumos vai konferenču materiālos

1. B.Apine, M.Gills, J.Plume. Software Development Process Improvement through Measurements and Requirements Traceability // DB&IS 2002 conference proceedings. Tallinn, June 3-6, 2002. Vol 2. pp 65-76.
2. J.Borzovs, M.Gills, A.Adamsone, S.Linde, J.Plume. Software Testing in Latvia: Lessons Learned. 1st International Conference on Software Testing ICSTEST - Conference Proceedings. Bonn, April 5-7, 2000. Track A, Paper 2. p 15.
3. J.Borzovs, M.Gills. Building up a team for manual testing. Proc. 2nd Intern. Conf. on Software Testing ICSTEST, Bonn, 2001, 12 p.
4. J.Borzovs, M.Gills. Software Testing in Latvia: Lessons Learned. Proceedings of Software Quality Week conference. San Francisco, 2001. Paper 6A2. p 15.
5. M.Gills, J.Iesalnieks. Remote-Entwicklung - was eignet sich, was nicht? Kongressunterlagen: 5.Kongress Software-Qualitätsmanagement SQM. Bonn, 5.-7. April 2000 Vortragsreihe 1.3, 11.S.
6. M.Gills. The Concept of a Universal Traceability Tool for Software Development Projects. Scientific Proceedings of Riga Technical University, Computer Science, Applied Computer Systems, 2st thematic issue, pp 33-41, 2001, Riga.
7. M.Gills, M.Bogdanovs. TraceIt: Enabling Extended Use of a Traceability Tool within Software Development Project. // DB&IS 2002 conference proceedings. Tallinn, June 3-6, 2002. Vol 1. pp 147-158.
8. M.Gills, M.Bogdanovs. Extended Use of a Traceability Tool within Software Development Project.// H-M. Haav, A. Kalja (Eds), Databases and Information Systems II, Selected Papers from the Fifth International Baltic Conference, BalticDB&IS'2002, Kluwer Academic Publishers, 2002, pp 175-186.
9. M.Gills. Review of Traceability Model Properties for Software Testing Process. // Scientific papers in Computer Science and Information Technologies. University of Latvia, 2004. Volume 669; pp. 80-88.
10. M.Gills. Review of Traceability Models for Software Testing Processes. SOFTTEST: UK Software Testing II Conference Proceedings. University of York, 4-5 September 2003. p.7.,
11. M.Gills. Experience of Introducing a Metamodel-based Traceability Tool into Software Development Projects // Scientific papers in Computer Science and Information Technologies. University of Latvia, 2004. DBIS'2004. Volume 673; pp. 208-219.
12. М.Богданов, В.Люмкис, М.Гилл Использование конструкций языка PHP для разработки средств тестирования проектов. // Transport and Telecommunication, Vol 4, No 1-2003, Riga-2003 pp 58.-65.

10.2. Citas autora publikācijas

13. M.Gills. Neatkarīgā programmatūras testēšana - mīts vai realitāte? Latvijas IT uzņēmumu 2.konference "Testēšanas teorija un prakse". Konferences materiāli. Rīga, 2001.gada 10.maijs.
14. M.Gills. Kāda neatkarīgā testēšana ir nepieciešama programmatūrai? A/S DATI 6.konferences "Informācijas tehnoloģija: zinības un prakse" materiāli. Rīga, 2001.gada .28.novembris.
15. M.Gills, S.Linde. Jūsu testēšanas ceļvedis 2002.gadā. Latvijas IT uzņēmumu 3.konference "Testēšanas teorija un prakse". Konferences materiāli. Rīga, 2002.gada 9.maijs. 2.-3.lpp.
16. M.Gills, M.Bogdanovs. Trasējamības rīka izmantošanas iespējas testēšanas procesā. Latvijas IT uzņēmumu 3.konference "Testēšanas teorija un prakse". Konferences materiāli. Rīga, 2002.gada 9.maijs. 33.-36.lpp.
17. M.Gills Programmatūras testēšanas vieta IT kvalitātes nodrošināšanā. "Kvalitāte"-"Quality". 2002. Nr. 4, 22.-23.lpp.
18. M.Gills. Traceability and Testing. Dagstuhl Seminar No. 02361 Report. Supporting Customer-Supplier Relationships: Requirements Engineering and Quality Assurance./Editor Barbara Paech. 2002. pp.6.
19. M.Gills. Par testētāja profesijas standartu. Latvijas IT uzņēmumu 4.konference "Testēšanas teorija un prakse". Konferences materiāli. Rīga, 2003.gada 7.maijs. 36.-41.lpp.
20. M.Gills. Kā trasējamībai likt strādāt testētāju labā? Latvijas IT uzņēmumu 5.konference "Testēšanas teorija un prakse". Konferences materiāli. Rīga, 2004.gada 12.maijs. 34.-37.lpp.

10.3. Citi darbā izmatotie avoti

21. Alexander, I. Towards Automatic Traceability in Industrial Practice. Proceedings of 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, UK. September 28th, 2002. pp 26-31
22. Alves-Foss, J.; Conte de Leon, D.; Oman, P. Experiments in the Use of XML to Enhance Traceability between Object-Oriented Design Specifications and Source Code. 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9, January 2002, pp. 276
23. Ambler, Scott. Tracing Your Design. Software Development Magazine. April 1999
24. ANSI/IEEE Std 830-1983 IEEE Guide for Software Test Documentation. 1983, 1991, p 48.
25. ANSI/IEEE Std 830-1984 IEEE Guide to Software Requirements Specifications. 1984, p 24.
26. Antoniol, G.; Canfora, G.; De Lucia, A. Maintaining Traceability During Object-Oriented Software Evolution: a Case Study. Proceedings of the IEEE International Conference on Software Maintenance, 1998, 9 p.
27. Antoniol, G.; Potrich, A.; Tonella, P.; Fiutem, R. Evolving Object Oriented Design to Improve Code Traceability. Seventh International Workshop on Program Comprehension. 5 - 7 May, 1999.Pittsburgh, Pennsylvania. pp. 151
28. Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, Ettore Merlo. Recovering Traceability Links between Code and Documentation. IEEE Transactions on Software Engineering, October 2002. pp. 970-983
29. Apache Software Foundation. Apache HTTP Server Project, 2004. http://httpd.apache.org/docs-project/
30. Paul Arkley, Paul Manson, Steve Riddle. Position Paper: Enabling Traceability. 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, UK. September 28th, 2002. p 5
31. Bach J. Risk and Requirements-Based Testing. Computer, June 1999, pp. 113-114.
32. K. S. Barber, T. J. Graser, Col. John Silva, M.D. RARE -- A Tool to Guide Development of an Object-Oriented Architecture while Maintaining Traceability to Requirements. Copyright The University of Texas at Austin, 1999. p 18.
33. Boris Beizer. Black-box testing: techniques for functional testing of software and systems. John Wiley & Sons, 1995. p 294.
34. B.Beizer. The Future of Software Quality. Software Quality Week Europe, Brussels, 1997, Keynote 7P p p 22.
35. Bogdan Bereza-Jarocinski. Is Software Testing Scientific? E-Proceedings of Software Quality Week Europe, Brussels, 9-13 November, 1998, p 1146.-1164.
36. Bogdan Bereza-Jarocinski, Anna Eriksson. Test Outside Software Industry: A Comparative Study. 1st ICSTEST Conference Proceedings 2000, paper A1, p 9.
37. Berry, D.M., Daudjee, K., Dong, J., Nelson, M.A., and Nelson, T. User's Manual as a Requirements Specification. Technical Report, CS 2001-17, University of Waterloo, Waterloo, ON, Canada, May, 2001.
38. Binder, Robert V. Testing Object-Oriented Systems: A Status Report. American Programmer, April 1994. http://www.rbsc.com/pages/ootstat.html
39. Black, Rex. Managing the testing process: practical tools and techniques for managing hardware and software testing. Wiley Publishing, Inc, 2002. p 500.
40. British Standards Institution. The TickIT Guide Issue 5.0, 2001.
41. BS 7925-1:1998 Software testing – Vocabulary.
42. BSI BS-7925-2:1998 Software testing - Software component testing, p 67.
43. E. J. Byrne. Software Reverse Engineering: A Case Study. Software Practice and Experience, Vol. 21(12), December 1991, pp 1349-1364.
44. Aniello Cimitile, Anna Rita Fasolino, Giuseppe Visaggio. A Software Model for Impact Analysis: A Validation Experiment. Sixth Working Conference on Reverse Engineering October 06 - 08, 1999 Atlanta, Georgia p. 212
45. CiteSeer Scientific Literature Digital Library. http://citeseer.ist.psu.edu
46. Corriveau J.-P. Traceability Process for Large OO Projects. Computer, September 1996, pp. 63-68.
47. Bill Councill, Carol Councill. Automating Requirements Traceability. STQE magazine July/August 2000, pp 49-54.
48. U.S. Consumer Product Safety Commission http://www.cpsc.gov
49. Rick D. Craig, Stefan P. Jaskiel, Systematic Software Testing. Artech House, 2002, p 536.

50. Crance, J.H. Guidelines for using the Delphi technique to develop habitat suitability index curves. Washington, DC : National Ecology Center Division of Wildlife and Contaminant Research Fish and Wildlife Service U.S. Dept. of the Interior, 1987, p 22.
51. Robert Culberson, Chris Brown, Gary Cobb. Rapid Testing. Prentice Hall PTR, 2002. p 394.
52. Daneva, Maya. Lessons Learnt from Five Years of Experience in ERP Requirements Engineering. 11th IEEE International Requirements Engineering Conference (RE'03), September 2003, pp. 45.
53. Jeremy Dick. Rich Traceability. 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, UK. September 28th, 2002. p 8.
54. DSK RR413320010. Allgemeiner Umdruck 250/1. BWB IT 1997. Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell. (Teil 1: Regelungsteil, Teil 3: Handbuchsammlung).
55. A. Egyed. A Scenario-Driven Approach to Trace Dependency Analysis. IEEE Transactions on Software Engineering (TSE), Volume 29, Number 2, February 2003, pp. 116-132.
56. Alexander Egyed, Paul Grünbacher. Automating Requirements Traceability: Beyond the Record & Replay Paradigm. Proceedings of the 17th International Conference on Automated Software Engineering (ASE), Edinburgh, Scotland, September 2002. pp 163-171.
57. Anna Rita Fasolino, Giuseppe Vissaggio. Improving Software Comprehension through an Automated Dependency Tracer. Seventh International Workshop on Program Comprehension, May 05 - 07, 1999, Pittsburgh, Pennsylvania p. 58
58. Golan, Elise; Krissoff, Barry; Kuchler, Fred; Nelson, Ken; Price, Greg; Calvin, Linda. Traceability in the US Food Supply: Dead End or Superhighway? Choices: The Magazine of Food, Farm & Resource Issues; Summer 2003, Vol. 18 Issue 2, p17, 4p.
59. Orlena C. Z. Gotel, Anthony C. W. Finkelstein. An Analysis of the Requirements Traceability Problem. 1994. Proceedings of the First International Conference on Requirements Engineering (ICRE '94), IEEE Computer Society Press, Colorado Springs, Colorado, U.S.A., April 18-22, pp. 94-101.
60. Grove Consultants Ltd. www.grove.co.uk
61. P.Gruenbacher and A.Egyed. Towards Understanding Implications of Trace Dependencies among Quality Requirements. Proceedings of 2nd Int. Workshop on Traceability in Emerging Forms of Software Engineering, ISSN-1364 4009, 2003.
62. Hambling, Brian. Realistic and cost-effective software testing // Management and Measurement of Software Quality. Edited by Mike Kelly. UNICOM Seminars Ltd 1993. pp. 95-111.
63. Jun Han. TRAM: A Tool for Requirements and Architecture Management. Australasian Computer Science Conference (ACSC '01) 29 Jan. - 2 Feb. 2001Gold Coast, Queensland, Australia. p.9.
64. Peter Haumer, Klaus Pohl, Klaus Weidenhaupt, Matthias Jarke. Improving Reviews by Extended Traceability. CREWS Report 98-17. Emerging Technologies Track of the Thirty-second Annual Hawaii International Conference on Systems Sciences (HICSS-32), Maui, HI, January 5-8, 1999, p 10.
65. J.C.Huang and D.Schmelzer. Dynamic Tracing Non-functional Requirements through Design Pattern Invariants. Proceedings of 2nd Int. Workshop on Traceability in Emerging Forms of Software Engineering, ISSN-1364 4009, 2003.
66. IBM Rational RequisitePro www.rational.com/products/reqpro/index.jsp
67. IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries 610. IEEE 1990, p 218.
68. IEEE 1008-1987 IEEE Standard for Software Unit Testing, p 28.
69. J-Std-016-1995 EIA/IEEE Interim Standard for Information Technology - Software Life Cycle Processes - Software Development Acquirer-Supplier Agreement, p 232.
70. 1465-1998 IEEE (12119:1998 ISO/IEC) Information Technology - Software Packages - Quality Requirements and Testing, p 22.
71. IEEE Computer Society Digital Library www.computer.org/publications/dlib/
72. IEEE Guide to the Software Engineering Body of Knowledge. www.swebok.org
73. International Council on Systems Engineering – INCOSE. www.incose.org
74. ISO/IEC Guide 2:1991, General terms and their definitions concerning standardization and related activities.
75. ISO 9000:2001 Quality management systems -- Fundamentals and vocabulary.
76. ISO/IEC 12207:1995 Information technology - Software life cycle processes, p 57.
77. ISO/IEC TR 13233:1995 Information technology - Interpretation of accreditation requirements in ISO/IEC Guide 25 - Accreditation of Information Technology and Telecommunications testing laboratories for software and protocol testing services.
78. Izglītības un zinātnes ministrija, Profesionālās izglītības centrs. Profesijas standarts Datorsistēmu testētājs. http://www.izmpic.lv/Standartu_reg/Datorsistemu_testetajs_4.pdf

79. James L. Automatic Requirements Specification Update Processing From A Requirements Management Tool Perspective. Proceedings of the 1997 Workshop on Engineering of Computer-Based Systems (ECBS '97), pp. 2-9.
80. Jeffries, Ronald E. XProgramming.com – an Extreme Programming Resource. www.xprogramming.com
81. Paul C. Jorgensen. Software testing: a craftsman's approach. CRC Press, 2002. p 359.
82. Juran, J.M., Gryna, Frank M. Quality planning and analysis: From producēt development through use. Second edition. McGraw-Hill, 1980, p 629.
83. Cem Kaner, Jack Falk, Hung Quoc Nguyen. Testing Computer Software. Wiley Computer Publishing, 1999. p 480.
84. Cem Kaner, James Bach, Bret Pettichord. Lessons learned in software testing: a context-driven approach. John Wiley & Sons, 2002. p 286.
85. Tim Koomen, Martin Pol. Test Process Improvement: A practical stet-by-step guide to structured testing. Addison-Wesley, 1999. p 218.
86. Dean Leffingwell, Don Widrig. The Role of Requirements Traceability in System Development. The Rational Edge. September 2002. http://www.therationaledge.com/content/sep_02/m_requirementsTraceability_dl.jsp
87. Cesar Leite J., Rossi G., Balaguer F., Maiorana V., Kaplan G., Hadad G., Oliveros A. Enhancing a Requirements Baseline with Scenarios. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97), pp. 43-53.
88. J.C.S.P.Leite and K.K.Breitman. Experiences Using Scenarios to Enhance Traceability. Proceedings of 2nd Int. Workshop on Traceability in Emerging Forms of Software Engineering, ISSN-1364 4009, 2003.
89. William E Lewis. Software testing and continuous quality improvement. CRCPress LLC, 2000. p 620.
90. Lindsay P., Liu Y., Traynor O. A Generic Model for Fine Grained Configuration Management Including Version Control and Traceability. Proceedings of the Australian Software Engineering Conference (ASWEC '97), 10 p.
91. LVS 65:1996 Informācijas tehnoloģija - Programminženierija – Programmatūras kvalitātes nodrošināšanas plāns. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 12 lpp.
92. LVS 66:1996 Informācijas tehnoloģija - Programminženierija – Programmatūras lietotāja dokumentācija. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 16 lpp.
93. LVS 67:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras projekta pārvaldības plāns. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 12 lpp.
94. LVS 68:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras prasību specifikācijas ceļvedis. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 22 lpp.
95. LVS 69:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras konfigurācijas pārvaldības plāns. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 17 lpp.
96. LVS 70:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras testēšanas dokumentācija. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 17 lpp.
97. LVS 71:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras verifikācijas un validācijas plāns. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 27 lpp.
98. LVS 72:1996 Informācijas tehnoloģija - Programminženierija – Ieteicamā prakse programmatūras projektējuma aprakstīšanai. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 13 lpp.
99. LVS 73:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras vienībtestēšana. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 20 lpp.
100. LVS 74:1996 Informācijas tehnoloģija - Programminženierija - Programmatūras apskates un auditēšanas. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 33 lpp.
101. LVS 75:1996 Informācijas tehnoloģija - Programminženierija – Sistēmas darbības koncepcijas apraksts. Latvijas Nacionālais standartizācijas un metroloģijas centrs, 1996, 11 lpp.
102. Marcus, Andrian; Maletic, Jonathan I. Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing. 25th International Conference on Software Engineering, May 2003, p. 11.
103. Brian Marick. The craft of software testing: subsystem testing including object-based and object-oriented testing. Prentice Hall PTR, 1995. p 553.
104. Mason, Paul; Saeed, Amer; Arkely, Paul; Riddle, Steve. Meta-Modelling Approach to Traceability for Avionics: A Framework for Managing the Engineering of Computer Based Aerospace Systems. 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03), April 2003, pp. 233.

105. McClure, Stuart; Scambray, Joel; Kurtz, George. Hacking Exposed: Network Security Secrets and Solutions, 3rd edition. McGraw-Hill, 2001, p 730.
106. Merant PVCS Tracker. http://www.merant.com/Products/ECM/tracker/
107. Mohan, K.; Ramesh, B.. Managing Variability with Traceability in Product and Service Families. Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02),Volume 3, January 2002. pp. 76 (p 9)
108. Musumeci, Gian-Paolo D.; Loukides, Mike. System Performance Tuning. 2nd edition. O'Reilly, 2002, p 334.
109. Myers, G. The Art of Software Testing. Wiley Interscience, 1979, p 179.
110. MySQL AB. MySQL Documentation, 2004. http://dev.mysql.com/doc/
111. Nguyen, Tien N.; Munson, Ethan V. A Model for Conformance Analysis of Software Documents. Sixth International Workshop on Principles of Software Evolution (IWPSE'03), September 2003, pp. 24.
112. OMG Unified Modelling Language Specification. UML v1.5 03-03-01 www.omg.org
113. Pas, Jens. Test Outsourcing. From Necessary Evil to Competitive Advantage SQWE 2000, e-proceedings, pp 85-94.
114. Justin K. Pearson. Requirements, Traceability and Formal Software or a Further Analysis of Requirements Traceability. October 1, 1996. p 19.
115. PHP Documentation Group. PHP Manual, 2004. http://www.php.net/manual/en/
116. Pinheiro F.A.C., Goguen J.A. An Object-Oriented Tool for Tracing Requirements. IEEE Software, March 1996, pp. 52-64.
117. Klaus Pohl, Stephan Jacobs. Concurrent Engineering: Enabling Traceability and Mutual Understanding. CERAs, 2(2), 1994, pp 279-290.
118. Pohl K. PRO-ART: Enabling Requirements Pre-Traceability. 1996 IEEE Proceedings of ICRE '96, pp. 76-84.
119. Pol, Martin; van Veenendaal, Erik. Structured testing of information systems – an introduction to TMap. Kluwer Bedrijfs Informatie, 1998. p 177.
120. Popper, Karl R. Science as Falsification // excerpt from Conjectures and Refutations, London: Routledge and Keagan Paul, 1963, pp. 33-39.
121. Roger S. Pressman. Software engineering: a practitioner's approach. McGraw-Hill, Inc. 1997 (4th ed.) p. 852
122. B. Ramesh, T. Powers, C. Stubbs, M. Edwards. Implementing requirements traceability: a case study. Second IEEE International Symposium on Requirements Engineering. March 1995, pp. 89.
123. Balasubramaniam Ramesh and Matthias Jarke. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, vol. 27, no. 1, January 2001. pp 58-93.
124. Rational Unified Process. www-306.ibm.com/software/awdtools/rup/
125. Reid, Stuart. Software Testing Requirements in Safety-Related Standards. Proceedings of 2nd ICSTEST conference, Bonn, April 4-6, 2001. Track F, p 22.
126. J.Richardson, J.Green. Traceability through Automatic Program Generation. Proceedings of 2nd Int. Workshop on Traceability in Emerging Forms of Software Engineering, ISSN-1364 4009, 2003.
127. Rubin, Jeffrey. Handbook of usability testing: how to plan, design, and conduct effective tests. John Wiley & Sons, 1994, p 330.
128. Saiema. Profesionālās izglītības likums. http://www.likumi.lv/doc.php?id=20244
129. Serena RTM. http://www.serena.com/Products/rtm/home.asp
130. A.Sherba, K.M.Anderson and M.Faisal. A Framework for Mapping Traceability Relationships. Proceedings of 2nd Int. Workshop on Traceability in Emerging Forms of Software Engineering, ISSN-1364 4009, 2003.
131. G.Spanoudakis. Supporting the Construction of Decision Models for Software Selection: A Probabilistic Traceability Approach. Proceedings of 2nd Int. Workshop on Traceability in Emerging Forms of Software Engineering, ISSN-1364 4009, 2003.
132. Spillner, Andreas. The W Model: Strengthening the Bond Between Development and Test. www.stickyminds.com. Jul 11, 2002, p 8.
133. Glenn A. Stout. Requirements Traceability and the Effect on the System Development Lifecycle (SDLC). Spring Cluster, 2001 http://www.reveregroup.com/exchange/articles/stout4.pdf p. 17.
134. Darijus Strasunskas. Traceability in Collaborative Systems Development from Lifecycle Perspective. 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, UK. September 28th, 2002. p 7
135. Mike Taylor. Tracking Action Items and Projects with Rational RequisitePro. The Rational Edge. October 2001.

136. Telelogic. DOORS. www.telelogic.com/products/doorsers/doors
137. Tsumaki, T.; Morisawa, Y. A framework of requirements tracing using UML. Seventh Asia-Pacific Software Engineering Conference (APSEC'00), 5-8 December, 2000 Singapore. pp. 206
138. Antje von Knethen. A trace model for system requirements changes on embedded systems. Proceedings of the 4th international workshop on Principles of software evolution.// International Conference on Software Engineering, Vienna, Austria, 2001. pp 17-26.
139. Antje von Knethen. Automatic Change Support based on a Trace Model. 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, UK. September 28th, 2002. p 8
140. von Knethen, Antje; Grund, Mathias. QuaTrace: A Tool Environment for (Semi-) Automatic Impact Analysis Based on Traces. International Conference on Software Maintenance (ICSM'03), September 2003. pp. 246
141. Robert Watkins and Mark Neal, Why and How of Requirements Tracing. IEEE Software July 1994. pp. 104-106.

11. Pielikumi

11.1. Izmantotā terminoloģija

Darba ietvaros jēdziens „programmatūras projekts", „programmizstrādes projekts" un „IT projekts" ir lietoti līdzīgās nozīmēs.

Jēdziens „izstrādātāji" ietver sevī programmētājus, sistēmanalītiķus, sistēmas arhitektu un citas projekta lomas, kuru primārais darbs ir darbināma koda izveide.

Jēdzieni „IT kompānija" un „IT uzņēmums" ir lietoti vienādās nozīmēs.

Jēdzieni „reālais projekts" un „industriālais projekts" ir lietoti līdzīgās nozīmēs.

Daži iespējamie latviskie tulkojumi atsevišķiem angļu valodas terminiem

test log – testžurnāls, testu žurnāls, testēšanas žurnāls
unit test – vienībtests, vienības tests
problem report – problēmziņojums
integration testing – integrācijas tests, integrācijtestēšana
system testing – sistēmas testēšana, sistēmtestēšana
system analyst – sistēmanalītiķis

11.2. Trasējamības aptauja Latvijas IT projektos

Anketa

Aptauja par trasējamības praksi programmatūras projektos

Akadēmisks pētījums, autors - M.Gills

1. Vispārējā informācija

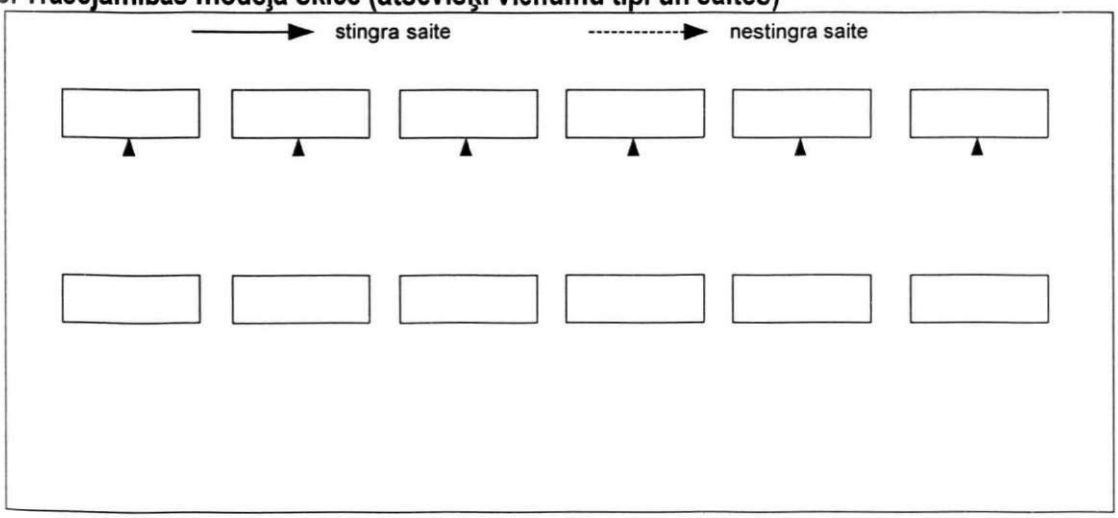
Aptaujas laiks:	Vieta:	Kontaktpersona:
Projekta nosaukums/kods (reāls vai anonimizēts):		
Projekta tips: ☐ Pilna programmatūras izstrāde ☐ Projektējuma sagatavošana ☐ Kodēšana	☐ Testēšana ☐ Uzturēšana ☐ Reinženierēšana ☐ cits:	
Projekta grupas izmērs: ☐ 1-4 cilvēki ☐ 5-15 cilvēki ☐ 16-40 cilvēki ☐ 41 un vairāk	Projekta ilgums: ☐ līdz 3 mēnešiem ☐ līdz 1 gadam ☐ vairāk kā gads	

2. Projekta norisei būtiskie vienuma tipi

☐ sākotnējās prasības *citi:* ☐ koncepcija ☐ līguma nosacījumi ☐ likums ☐ normatīvais dokuments	☐ prasības *citi:* ☐ lietojumgadījums (*use case*) ☐ scenārijs ☐ modelis

☐ standarts	
☐ projektējuma vienums *citi:* ☐ modulis ☐ funkcija ☐ entīte	☐ kods *citi:* ☐ koda funkcijas ☐ klases ☐ metodes
☐ testi *citi:* ☐ testpiemēri ☐ testu kopas/komplekti ☐ testžurnāli	☐ problēmziņojumi *citi:* ☐ izmaiņu pieprasījumi
(citi vienumu tipi)	*(citi vienumu tipi)*

3. Trasējamības modeļa skice (atsevišķi vienumu tipi un saites)



4. Projekta trasējamības modelis (precizēts variants)

5. Noslēguma jautājumi

5.1. Vai projekta laikā iznāca īpaši domāt par trasējamību ☐ Jā ☐ Nē ☐ Daļēji
5.2. Vai šajā anketā atspoguļotais variants atbilst pēdējam/aktuālākajam variantam? ☐ Jā ☐ Nē
5.3. Vai projekta metodika tika pārņemta no kāda veiksmīga iepriekšēja piemēra? ☐ Jā ☐ Nē ☐ Nezinu
5.4. Vai minētā projekta pieredze ir vai tiks izmantota arī citos projektos? ☐ Jā ☐ Nē ☐ Nezinu
5.5. Vai trasējamības tipa informācija tika uzturēta ar rīku palīdzību? ☐ Jā ☐ Nē ☐ Daļēji
5.6. Ja uz 5.5. jautājumu atbilde ir "Jā" vai "Daļēji", kādi rīki tika izmantoti?
5.7. Ja uz 5.5. jautājumu atbilde ir "Nē", vai tiek apsvērta doma izmantot kādus rīkus šajā vai analoģiskā projektā? ☐ Jā ☐ Nē ☐ Nezinu

Pavadvēstule

Par trasējamības aptauju

Godātais aptaujas dalībniek!

Vispirms pateicos par atsaucību, ka esat piekritis(-usi) piedalīties aptaujā.

Mazliet par to, kādēļ es rīkoju šādu aptauju:
Paralēli darbam Rīgas Informācijas tehnoloģijas institūtā veicu doktorantūras studijas Latvijas Universitātē, datorzinātņu nozarē. Mana zinātniskā darba centrālā tēma ir trasējamības (*traceability*) modeļi programmatūras projektos. Lai arī vairāku gadu laikā tiešā veidā esmu iepazinis dažādus projektus, kā arī esmu pētījis zinātnisko literatūru, vēlos plašāk uzzināt par projektu informācijas trasējamības praksi programmatūras projektos Latvijas IT uzņēmumos. Mans mērķis nav analizēt kādu konkrētu uzņēmumu, bet gan iegūt pēc iespējas plašāku kopskatu.

Aptaujas norisei ir divi galvenie soļi:
1. Projekta pārstāvis - aptaujas dalībnieks (t.i. Jūs vai Jūsu kolēģis) iepazīstas ar anketas jautājumiem. Pārdomā iespējamo atbildi viena vai vairāku konkrēti izvēlētu projektu gadījumā.
2. Es tiekos ar aptaujas dalībnieku, lai kopīgi aizpildītu anketu. Aizpildīšanas laikā izklāstu terminoloģiju un neskaidrās lietas, kopā uzzīmējam trasējamības modeli. Plānotais tikšanās laiks - ap 30 minūtēm.

Klātienes formu anketas aizpildīšanai esmu izvēlējies tādēļ, ka tas ļaus Jums ietaupīt laiku. Viena anketa domāta viena projekta datu reģistrēšanai.

Par pašu anketu:
1. Vispārējā informācija. Šī sadaļa ir domāta projekta vispārējai identifikācijai un galveno raksturlielumu fiksēšanai. Jā vēlaties, konkrētā projekta reālais nosaukums varat aizstāt ar citu nosaukumu. Katram projektam sava anketa.
2. Projekta norisei būtiskie vienumu tipi (*item types*). Šajā sadaļā būtu jāatzīmē tie projekta vienumu/informācijas tipi, kas bija būtiski konkrētā produkta iegūšanai. Šajā gadījumā ar projekta pārvaldību saistīti vienumi (*items*) tiek aplūkoti tikai tik tālu, lai tie attiektos uz iegūstamā produkta saturisko pusi. Pētījumā netiek aplūkoti resursu pārvaldības, finanšu u.tml. projekta informācijas vienumi.
3. Trasējamības modeļa skice. Šeit mēs kopā varēsim iezīmēt būtiskākos vienumu pārus un saites starp tiem. Piedāvātās sešas vienumu pāru sagataves nav ierobežojums - saišu var būt arī vairāk (tam var izmantot papildus lapu).
4. Projekta trasējamības modelis. Apkopojot identificētos vienumus un raksturīgās saites starp tiem, mēs kopīgi varēsim nodefinēt precizētu trasējamības modeli.
5. Noslēguma jautājumi. Noslēgumā ir daži jautājumi par trasējamību konkrētajā projektā.

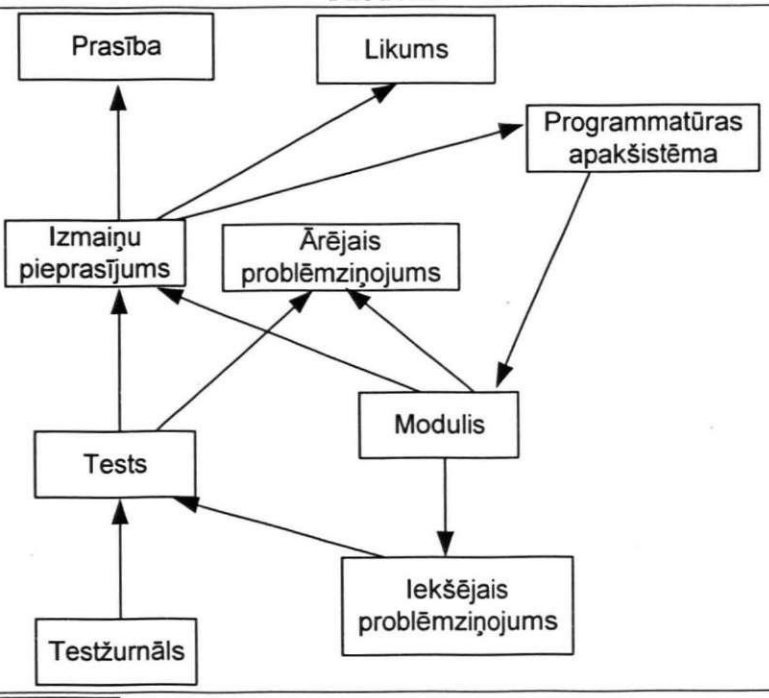
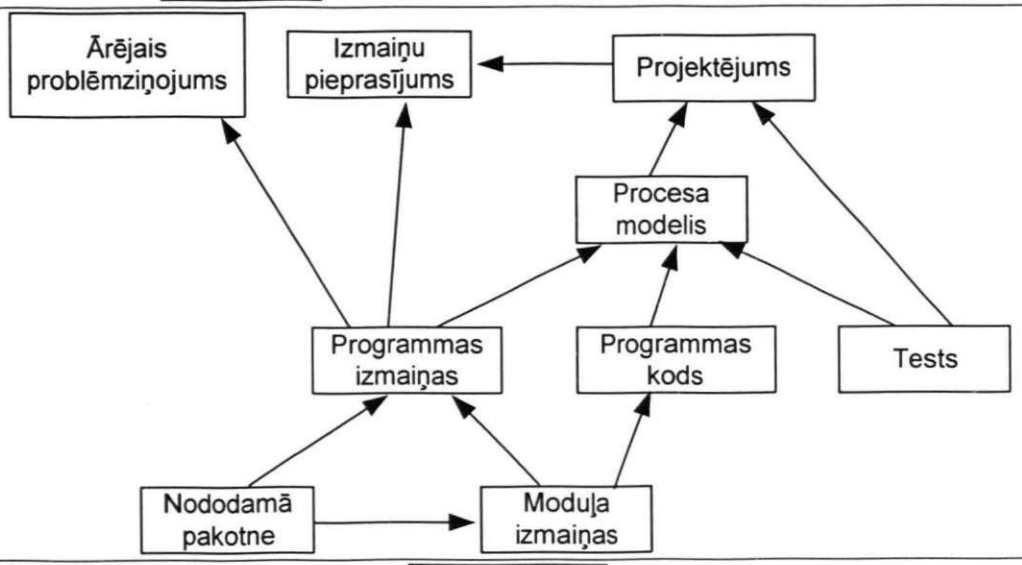
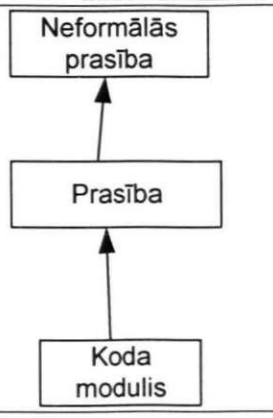
Nojaušu, ka pēc šī nelielā komentāra un anketas caurskatīšanas nebūt nerodas īsts priekšstats, kas ar to visu īsti ir domāts. Tādēļ piedāvāju klātienē atbildēt un aizpildīt anketu.

Lai sarunātu tikšanās laiku un vietu, vēlos no Jums saņemt kādu ziņu par iespējamajiem variantiem. Mans e-pasts ir Martins.Gills@riti.lv, telefons 9289205.

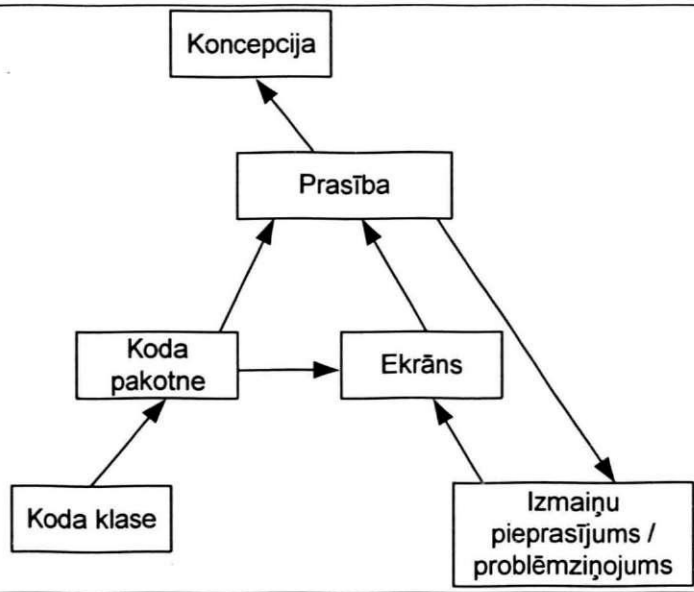
Ar cieņu,

Mārtiņš Gills

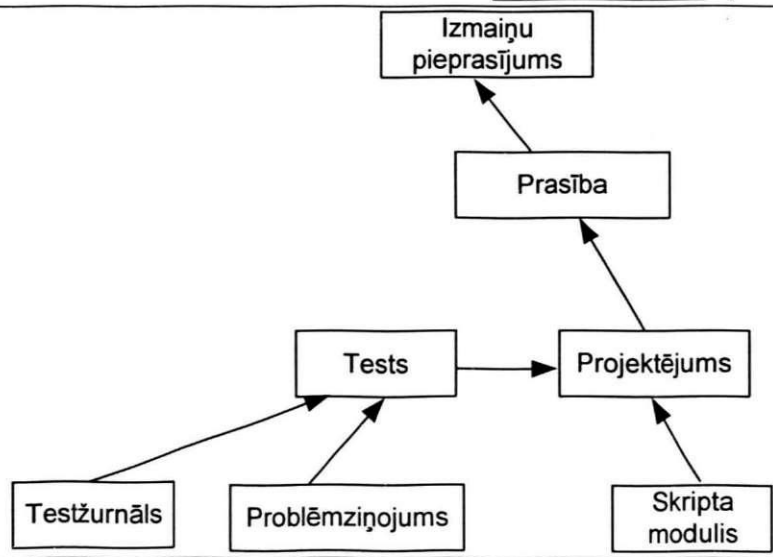
Identificētie trasējamības modeļi

Nr.	Modelis
1.	
2.	
3.	

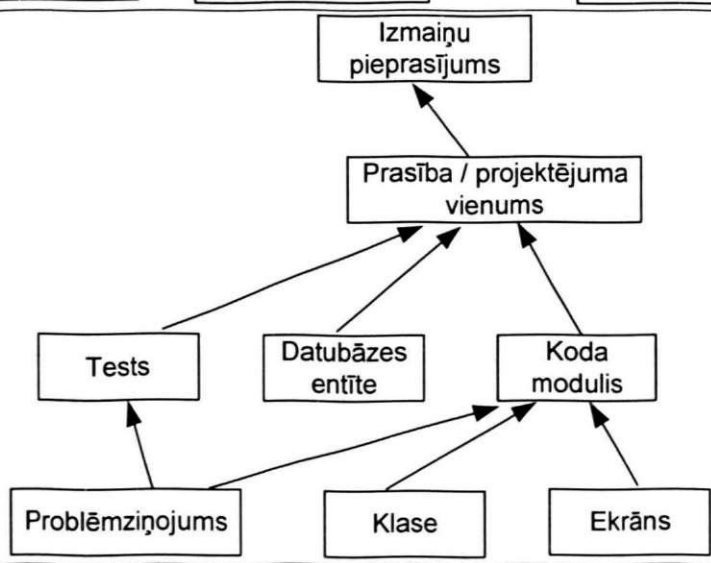
4.


Koncepcija
Prasība
Koda pakotne
Ekrāns
Koda klase
Izmaiņu pieprasījums / problēmziņojums

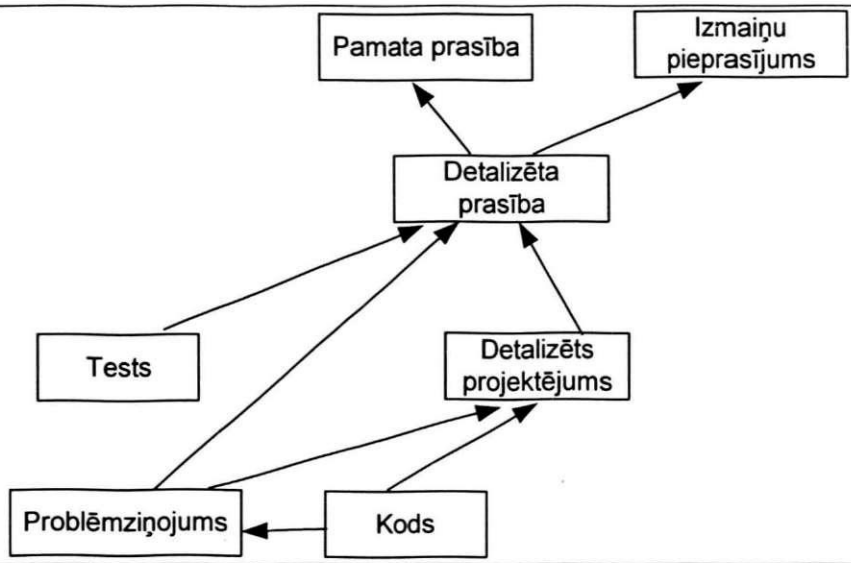
5.


Izmaiņu pieprasījums
Prasība
Tests
Projektējums
Testžurnāls
Problēmziņojums
Skripta modulis

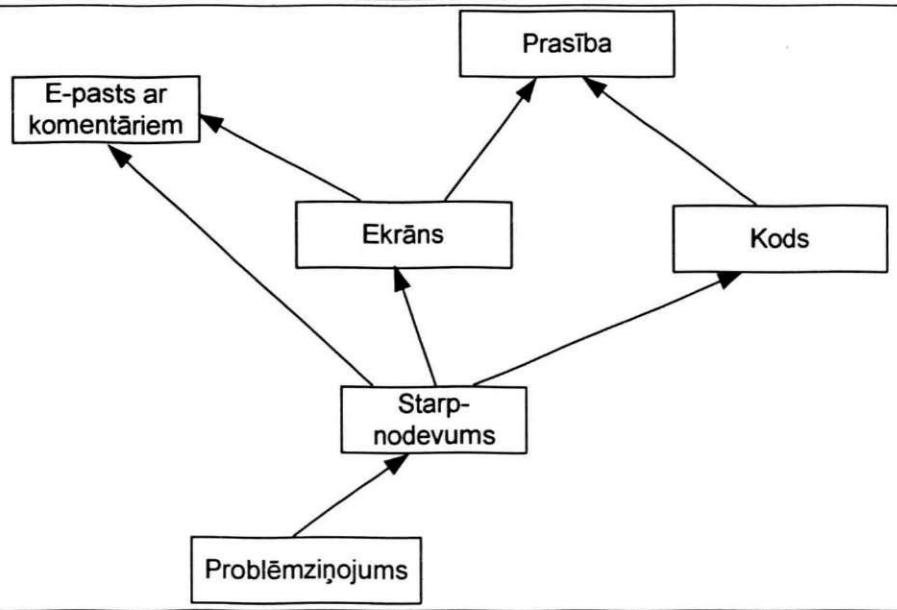
6.


Izmaiņu pieprasījums
Prasība / projektējuma vienums
Tests
Datubāzes entīte
Koda modulis
Problēmziņojums
Klase
Ekrāns

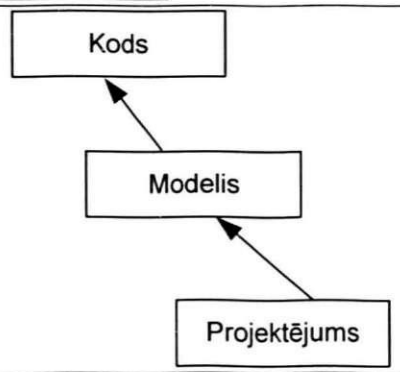
7.


Pamata prasība
Izmaiņu pieprasījums
Detalizēta prasība
Tests
Detalizēts projektējums
Problēmziņojums
Kods

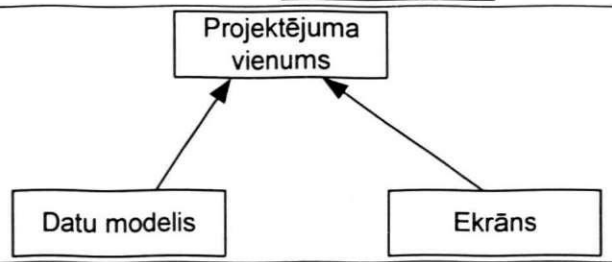
8.


Prasība
E-pasts ar komentāriem
Ekrāns
Kods
Starp-nodevums
Problēmziņojums

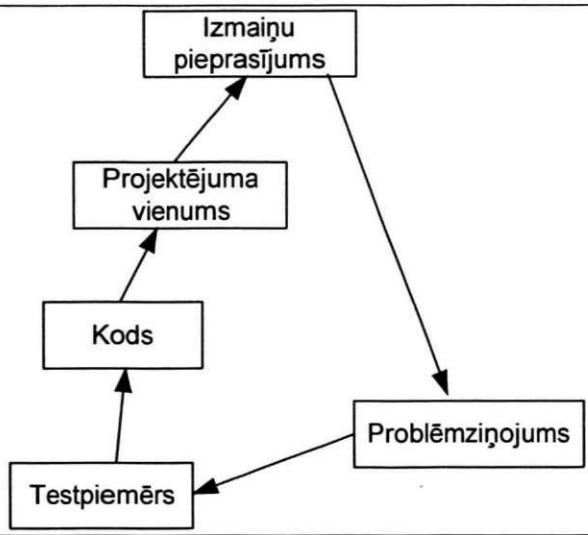
9.


Kods
Modelis
Projektējums

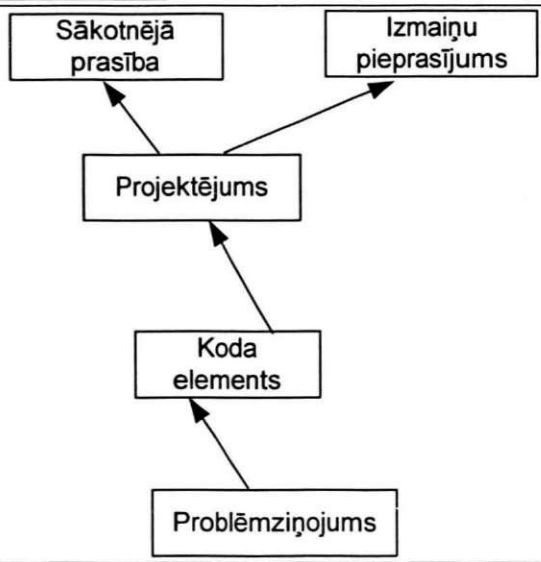
10.


Projektējuma vienums
Datu modelis
Ekrāns

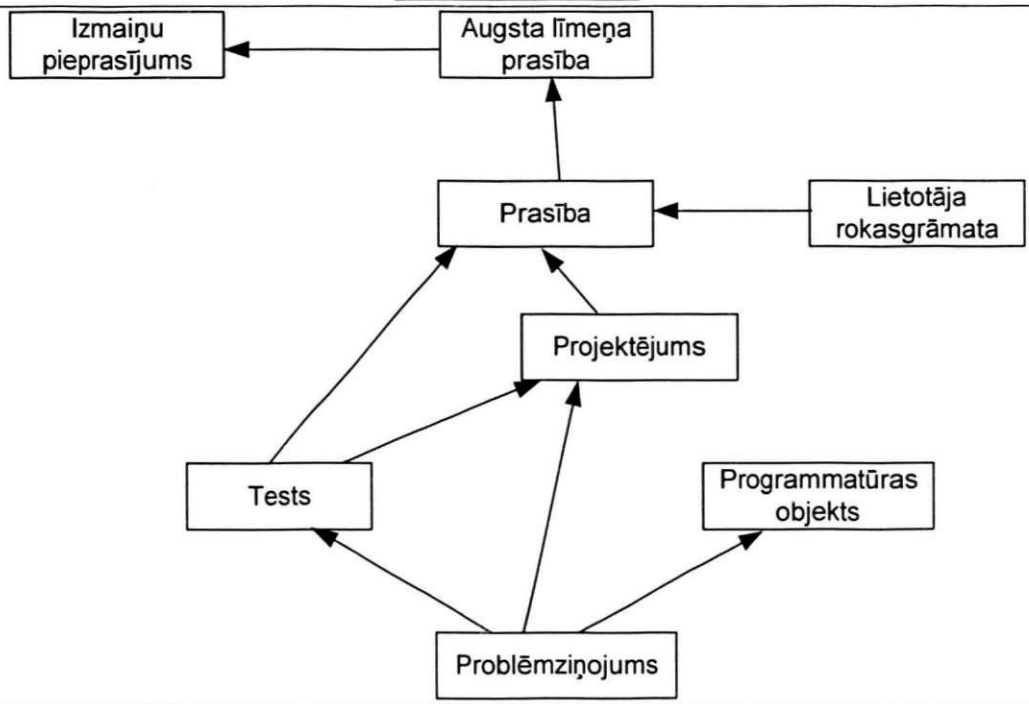
11.


Izmaiņu pieprasījums
Projektējuma vienums
Kods
Testpiemērs
Problēmziņojums

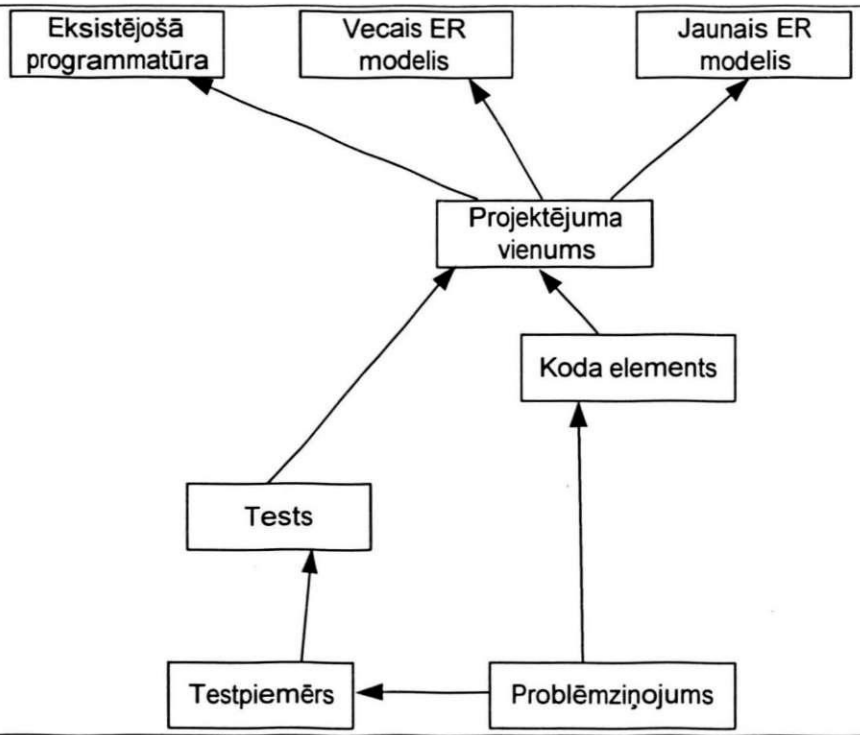
12.


Sākotnējā prasība
Izmaiņu pieprasījums
Projektējums
Koda elements
Problēmziņojums

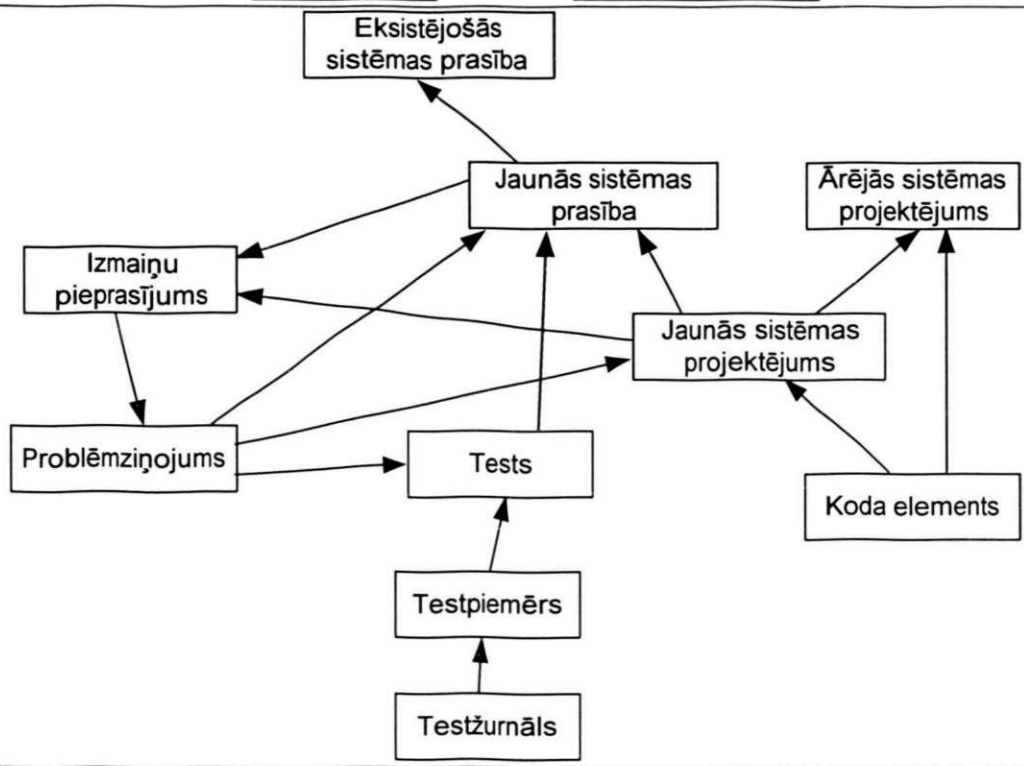
13.


Izmaiņu pieprasījums
Augsta līmeņa prasība
Prasība
Lietotāja rokasgrāmata
Projektējums
Tests
Programmatūras objekts
Problēmziņojums

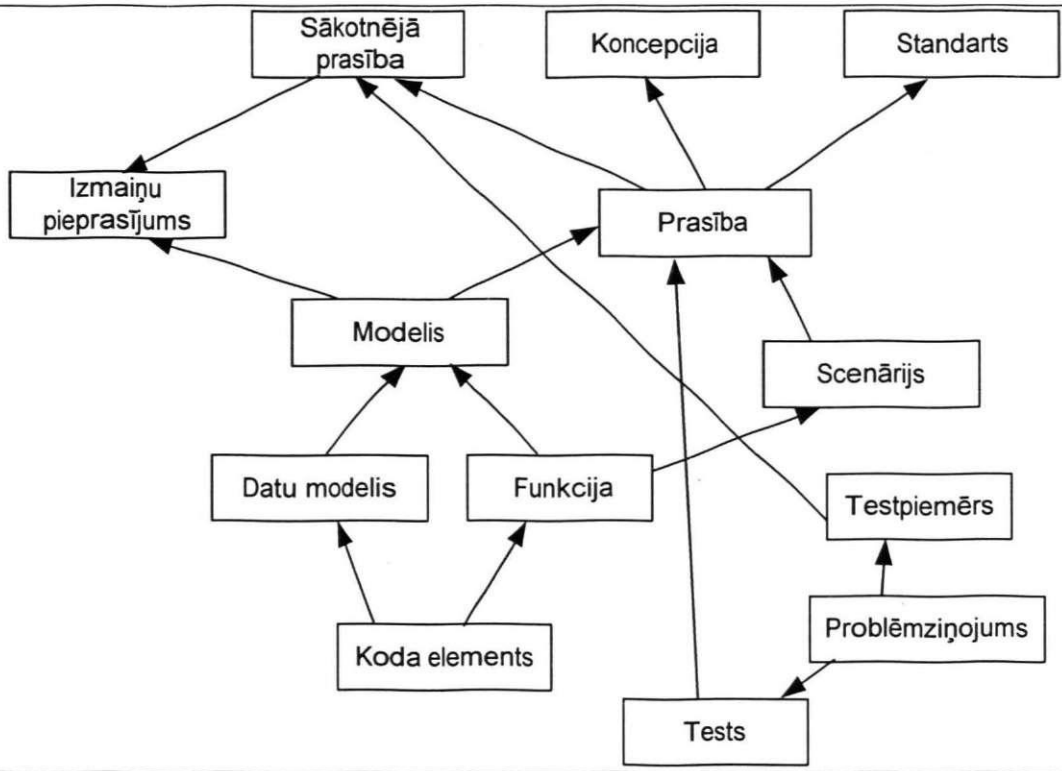
14.


Eksistējošā programmatūra
Vecais ER modelis
Jaunais ER modelis
Projektējuma vienums
Koda elements
Tests
Testpiemērs
Problēmziņojums

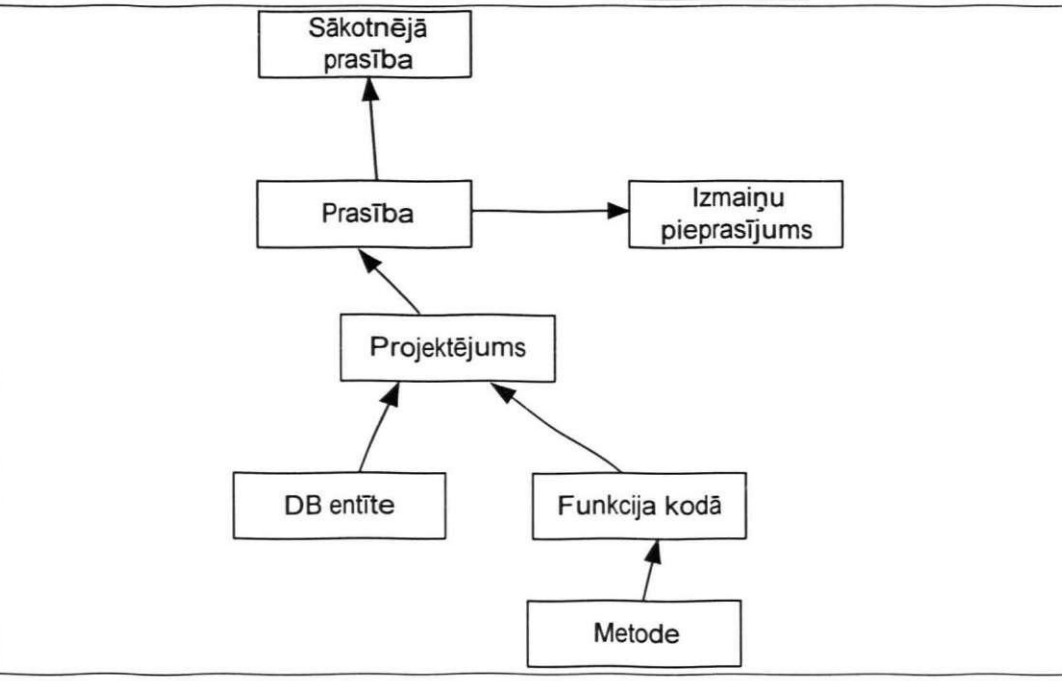
15.


Eksistējošās sistēmas prasība
Jaunās sistēmas prasība
Ārējās sistēmas projektējums
Izmaiņu pieprasījums
Jaunās sistēmas projektējums
Problēmziņojums
Tests
Koda elements
Testpiemērs
Testžurnāls

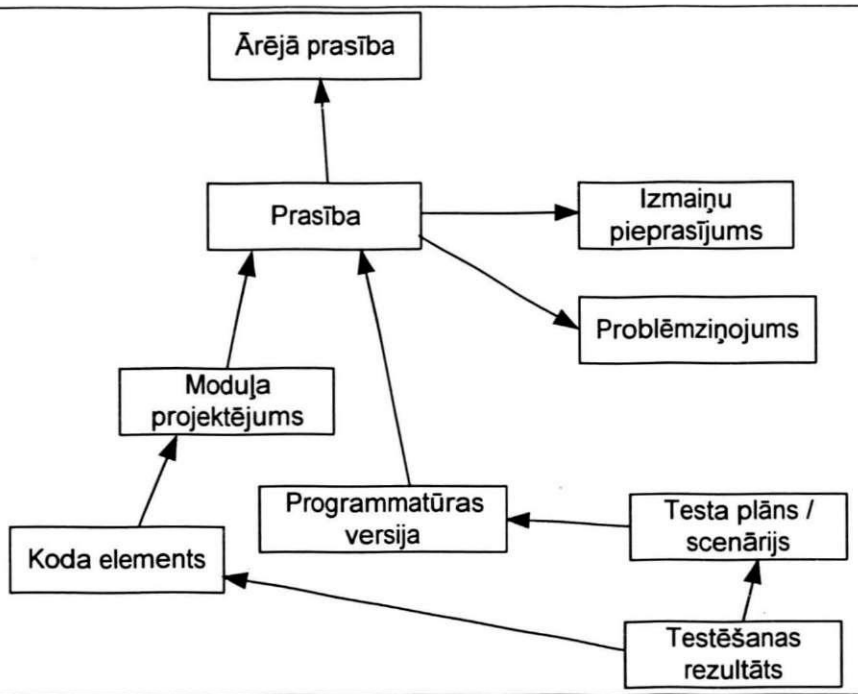
16.


Sākotnējā prasība
Koncepcija
Standarts
Izmaiņu pieprasījums
Prasība
Modelis
Scenārijs
Datu modelis
Funkcija
Testpiemērs
Problēmziņojums
Koda elements
Tests

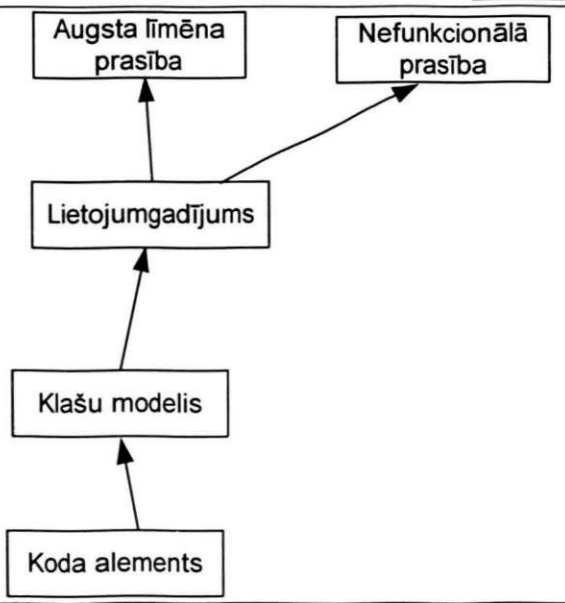
17.


Sākotnējā prasība
Prasība
Izmaiņu pieprasījums
Projektējums
DB entīte
Funkcija kodā
Metode

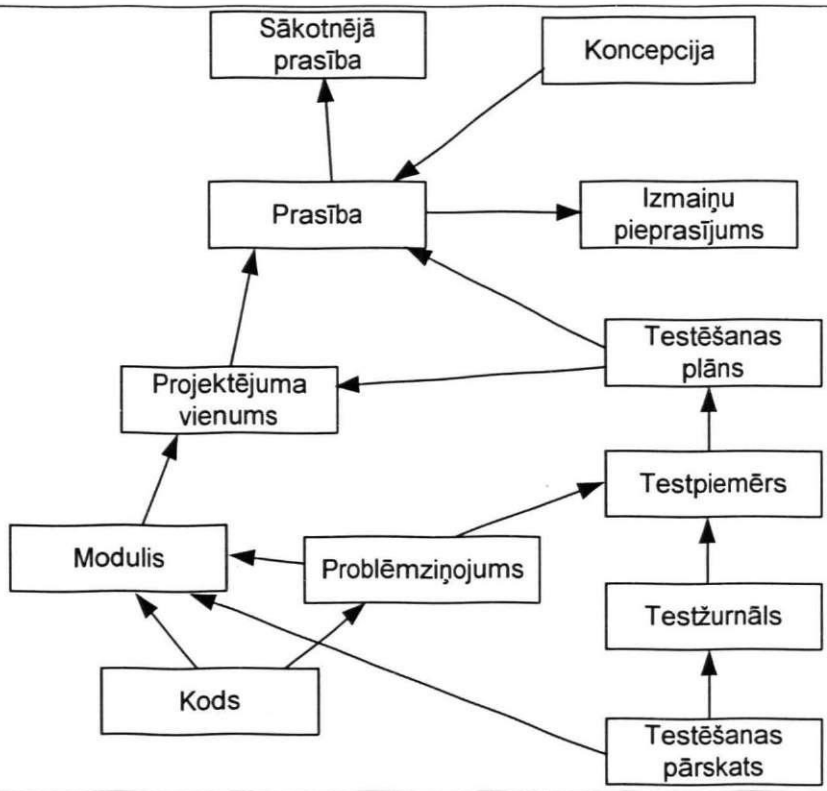
18.


Ārējā prasība
Prasība
Izmaiņu pieprasījums
Problēmziņojums
Moduļa projektējums
Programmatūras versija
Testa plāns / scenārijs
Koda elements
Testēšanas rezultāts

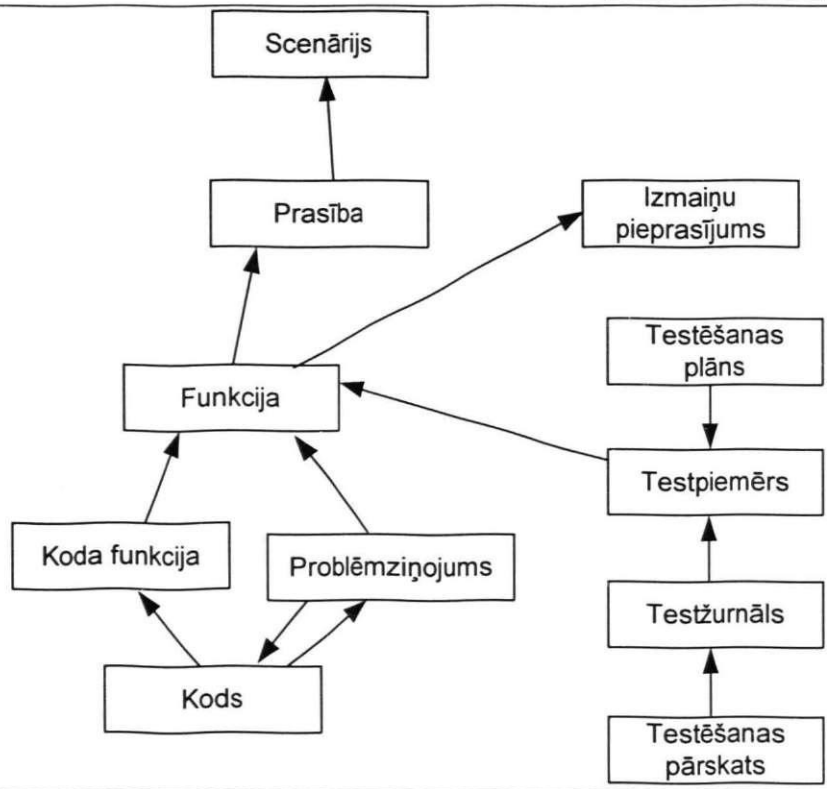
19.


Augsta līmena prasība
Nefunkcionālā prasība
Lietojumgadījums
Klašu modelis
Koda alements

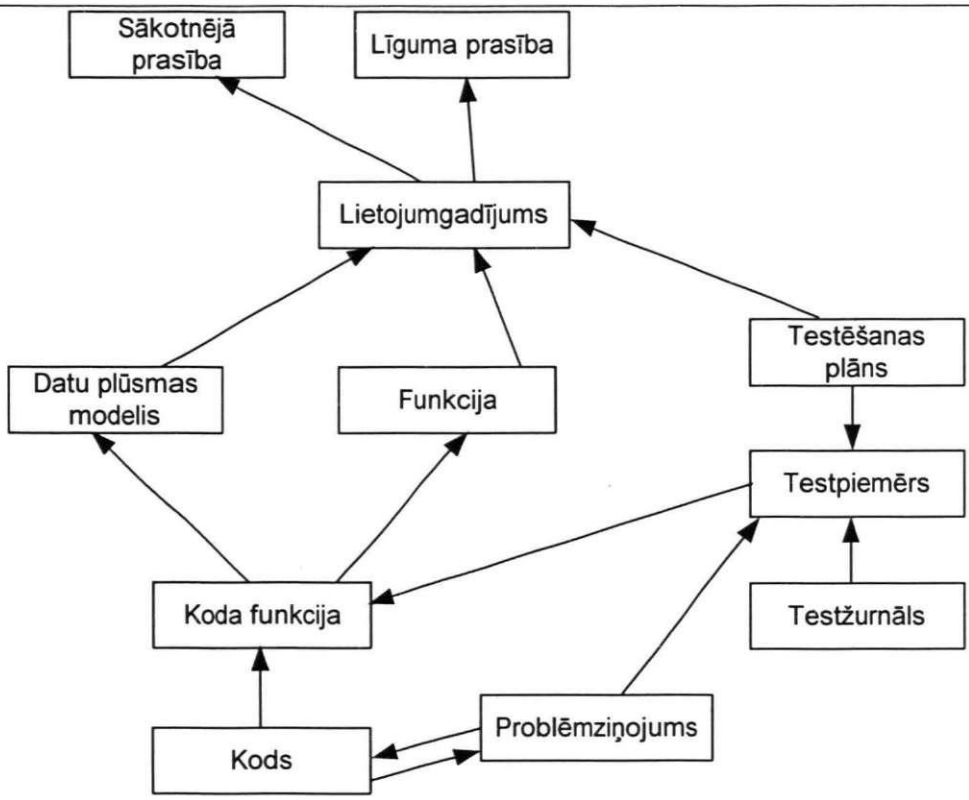
20.


Sākotnējā prasība
Koncepcija
Prasība
Izmaiņu pieprasījums
Testēšanas plāns
Projektējuma vienums
Testpiemērs
Modulis
Problēmziņojums
Testžurnāls
Kods
Testēšanas pārskats

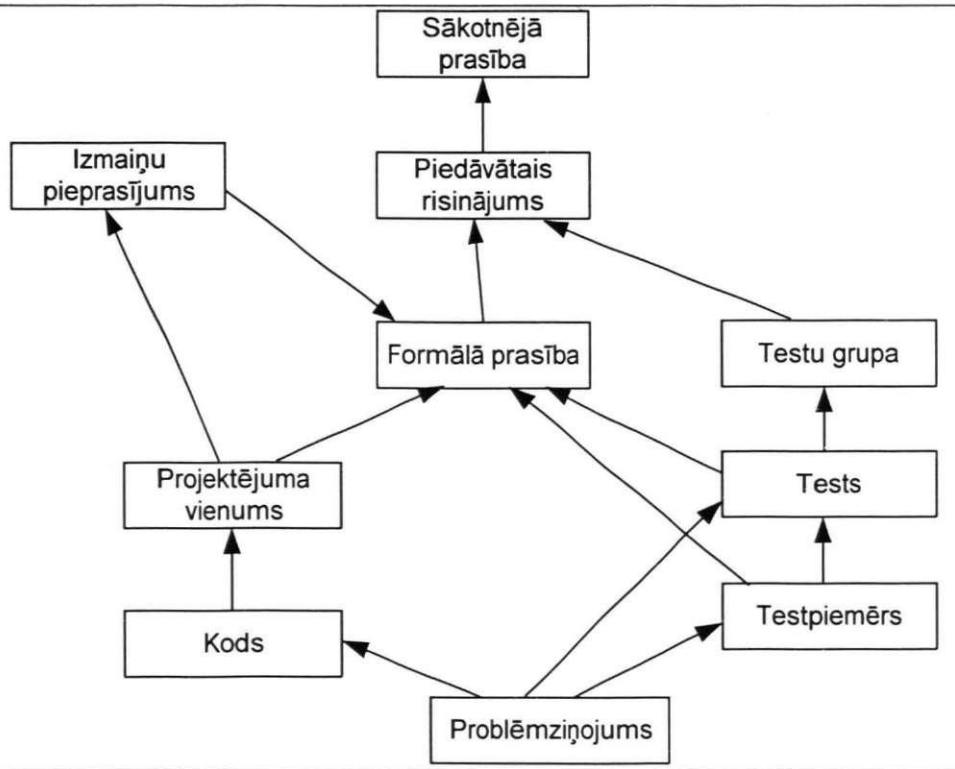
21.


Scenārijs
Prasība
Izmaiņu pieprasījums
Testēšanas plāns
Funkcija
Testpiemērs
Koda funkcija
Problēmziņojums
Testžurnāls
Kods
Testēšanas pārskats

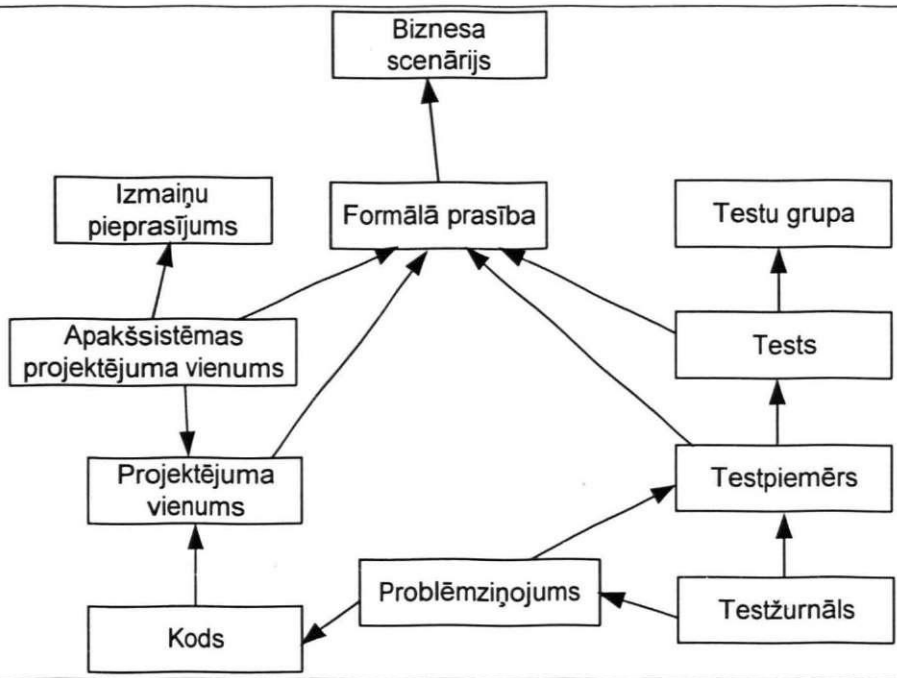
22.


Sākotnējā prasība
Līguma prasība
Lietojumgadījums
Testēšanas plāns
Datu plūsmas modelis
Funkcija
Testpiemērs
Koda funkcija
Testžurnāls
Problēmziņojums
Kods

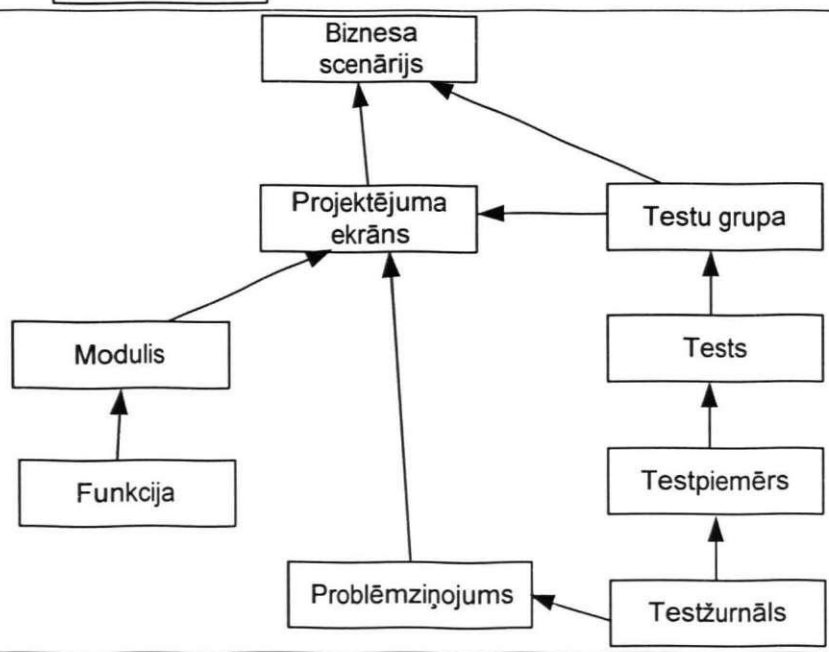
23.


Sākotnējā prasība
Izmaiņu pieprasījums
Piedāvātais risinājums
Formālā prasība
Testu grupa
Projektējuma vienums
Tests
Testpiemērs
Kods
Problēmziņojums

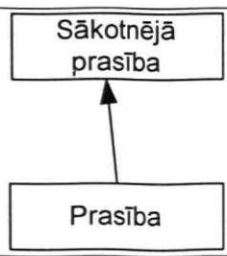
24.


Biznesa scenārijs
Izmaiņu pieprasījums
Formālā prasība
Testu grupa
Apakšsistēmas projektējuma vienums
Tests
Projektējuma vienums
Testpiemērs
Problēmziņojums
Testžurnāls
Kods

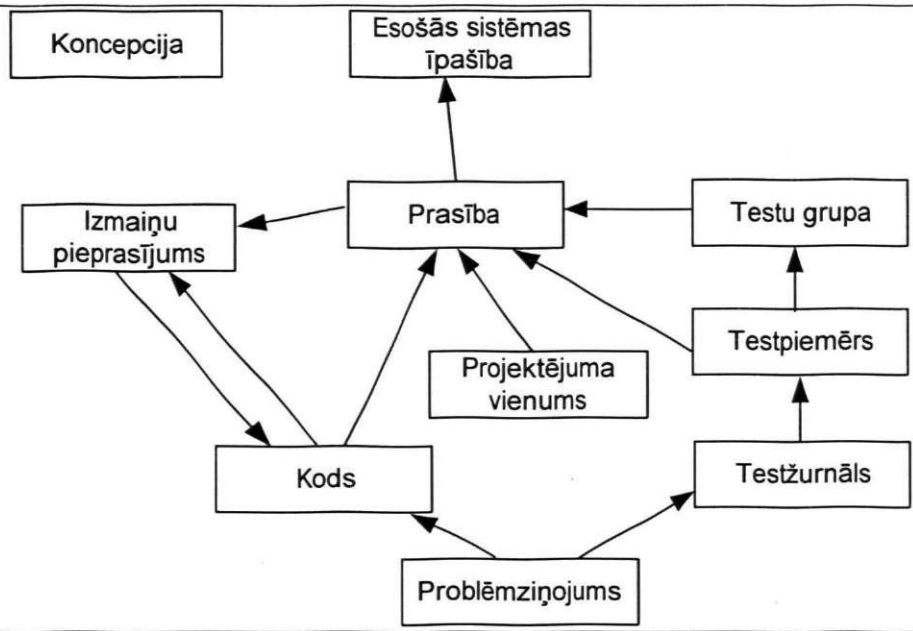
25.


Biznesa scenārijs
Projektējuma ekrāns
Testu grupa
Modulis
Tests
Funkcija
Testpiemērs
Problēmziņojums
Testžurnāls

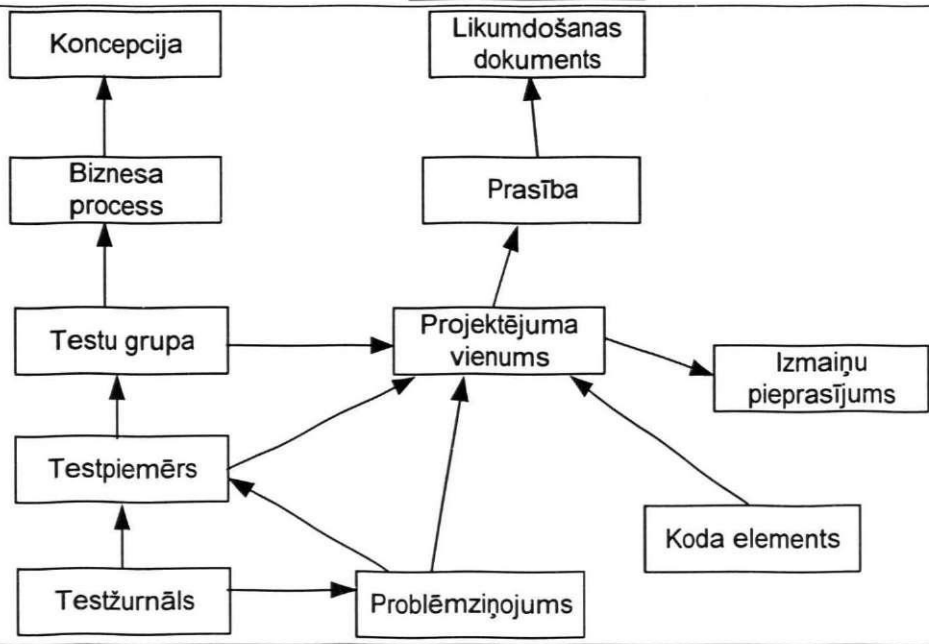
26.


Sākotnējā prasība
Prasība

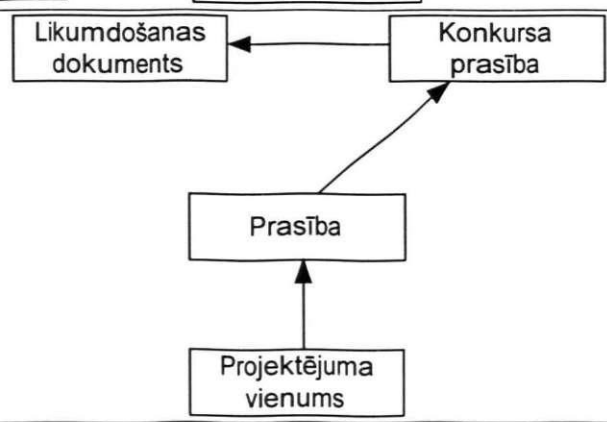
27.


Koncepcija
Esošās sistēmas īpašība
Prasība
Izmaiņu pieprasījums
Testu grupa
Testpiemērs
Projektējuma vienums
Kods
Testžurnāls
Problēmziņojums

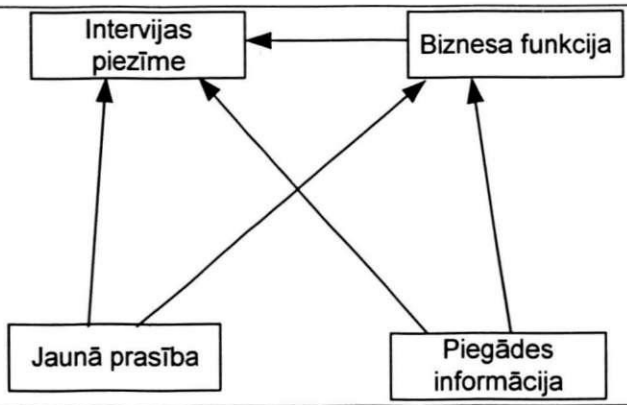
28.


Koncepcija
Likumdošanas dokuments
Biznesa process
Prasība
Testu grupa
Projektējuma vienums
Izmaiņu pieprasījums
Testpiemērs
Koda elements
Testžurnāls
Problēmziņojums

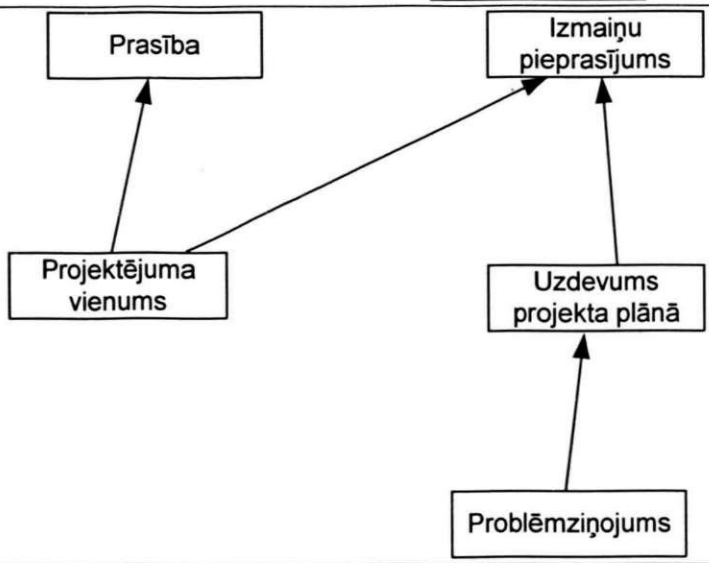
29.


Likumdošanas dokuments
Konkursa prasība
Prasība
Projektējuma vienums

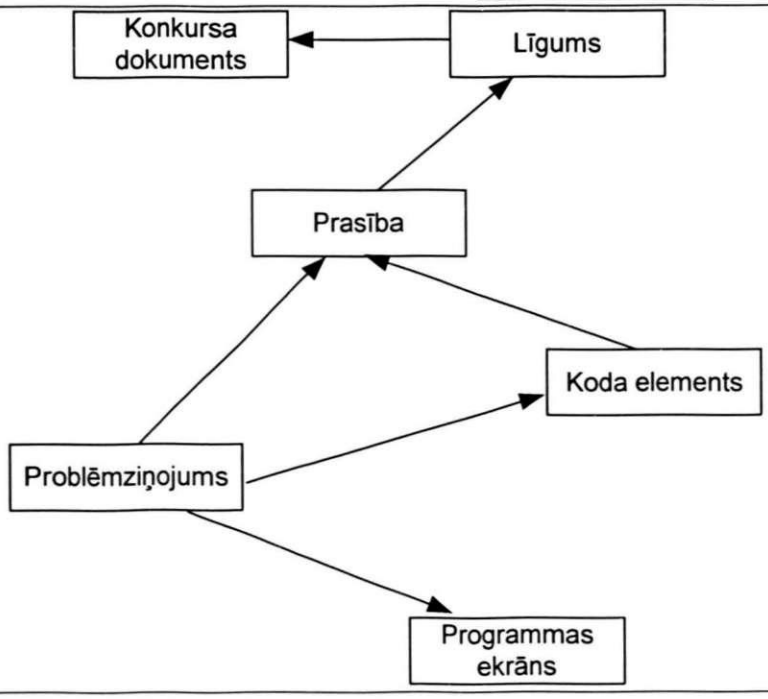
30.


Intervijas piezīme
Biznesa funkcija
Jaunā prasība
Piegādes informācija

31.


Prasība
Izmaiņu pieprasījums
Projektējuma vienums
Uzdevums projekta plānā
Problēmziņojums

32.


Konkursa dokuments
Līgums
Prasība
Koda elements
Problēmziņojums
Programmas ekrāns

11.3. TraceIt lietošanas pieredzes aptauja

1.kārta - Pavadvēstule

Godātais aptaujas dalībniek!

Jau vairākus gadus DATI ietvaros tiek izmantots rīks TraceIt, un katrā no projektiem tam ir rasts mazliet citāds pielietojums. Esam uzsākuši aptauju par šī un šim rīkam līdzīgu programmu izmantošanas pieredzi. Mērķis ir sistematizētā formā apzināt šī rīka izmantošanas veidus, iegūtos iespaidus, ierosinājumus un kritiku. Ņemot vērā to, ka kāda projekta ietvaros Jūs esat lietojis(-usi) (vai arī lietojat šobrīd) rīku TraceIt, Jūs esat izvēlēts(-a) kā viens no aptaujas dalībniekiem. Ceram, ka radīsiet iespēju tajā piedalīties.

Atšķirībā no tradicionālajām aptaujām šī tiek veidota saskaņā ar Delfi metodi. Tas nozīmē, ka papildus pirmajai anketai, ko lūgsim aizpildīt nedēļas laikā, sekos vēl divas, kur katra būs veidota, balstoties uz iepriekšējās reizes rezultātiem. Anketa ir veidota tā, lai nebūtu jāpatērē vairāk kā pusstunda tās aizpildīšanai.

Pirmās iterācijas jautājumi ir brīvā formā. Vēlams atbildēt uz visiem jautājumiem. Atbildes apjoms nav ierobežots. Lūgums atbildes rakstīt tā paša jautājumu faila (Word dokuments) ietvaros, un nosūtīt elektroniski atpakaļ sūtītājam.

Aptauju organizē divas ieinteresētās puses - jautājumus ir sagatavojis TraceIt līdzautors Mārtiņš Gills, bet izplatīšanu un apkopošanu veic DATI Kvalitātes pārvaldības dienests, kurš jautājumu autoram atbildes tālākajai apstrādei nodod tikai anonimizētā formā (ar dzēstu informāciju par konkrēto atbilžu autorību).

Lūgums anketu aizpildīt un nosūtīt uz e-pasta adresi [...].

Ceram uz atsaucību.

Ar cieņu,

[...]

1.kārta – anketa

Aptauja "TraceIt lietošanas pieredze"
1. anketa

Lūdzu rakstiet atbildes brīvā formā. Apjoms nav ierobežots (bet lūgums neatstāt tukšu).

1.	Lūdzu norādiet tipiska programmatūras projekta aktivitāti/procesu, kas, jūsuprāt, visvairāk ir atkarīgs no trasējamības

2.	Vai saskatāt iespēju, ka sistemātiskas trasējamības nodrošināšanas varētu dot kādu reālu labumu projektiem? Ja jā, tad ko tieši?

3.	Kādu atbalstu jūs sagaidāt no programmas, kas tiek saukta par trasējamības rīku?

4.	Kādām projekta ikdienas funkcijām izmantojāt TraceIt?

5.	Kuras TraceIt iespējas izmantojāt visbiežāk?

6.	Kuras speciālās TraceIt iespējas izmantojāt retāk?

7.	Ko vēlējāties iegūt ar TraceIt palīdzību, bet nevarējāt?

8.	Kurām projekta lomām šis rīks ir visvairāk noderīgs?

9.	Kurām projekta lomām šis rīks ir vismazāk noderīgs?

10.	Kāda tipa projektiem, jūsuprāt, šis rīks būtu vislabāk piemērots?

11.	Kāda tipa projektiem, jūsuprāt, šis rīks nav piemērots?

12.	Vai ir mēģināti citi rīki, kas labāk nekā TraceIt nodrošina trasējamības uzdevumus projektā? Ja jā, tad kādi?

13.	Vai bez TraceIt tika mēģināti citi rīki, bet tie nepietiekami atbalstīja projekta trasējamību? Ja jā, tad kādi?

14.	Kā Jūs raksturotu galveno atšķirību darba stilos, ja trasējamības informācijas uzturēšanai izmanto katru no minētajiem: parastā teksta pierakstu, MS Excel tabulas, TraceIt ?

15.	Vai ieteiktu izmantot TraceIt citos projektos? Kādēļ [jā/nē]?

2.kārta – pavadvēstule

Ļoti pateicamies par piedalīšanos TraceIt aptaujas 1.kārtā.

Ir apkopoti rezultāti, un uz to bāzes ir sagatavota 2.kārtas anketa. Kā jau bija minēts pirmajā aptaujas reizē, kopā būs trīs reizes, kur katra nākošā anketa tiks veidota, izmantojot iepriekšējās reizes rezultātus.

Vēlamies uzsvērt, ka otrajā kārtā aicinām piedalīties arī tos, kuriem pirmajā reizē vienu vai citu iemeslu nesanāca nosūtīt aizpildītu anketu. 2.kārtas aptaujas anketu ir vienkāršāk aizpildīt – iepretim atbildes variantam jeb sniegtajam viedoklim jānorāda tikai savs vērtējums, cik stipri tam piekrītat. Un tikai gadījumā, ja piedāvāto variantu vidū iztrūkst viedokļa, kuru vēlētos šeit redzēt, izmantojiet katra jautājuma pēdējo atbildes variantu „Cits", lai pierakstītu savu viedokli.

Jāpiezīmē, ka aptaujas atbilžu varianti nav savstarpēji izslēdzoši, bet gan apraksta dažādus jautājuma sfēras aspektus.

Lūgums anketu aizpildīt un nosūtīt uz e-pasta adresi [...].

Vēlam veiksmi!

Ar cieņu,

[...]

2.kārta – anketa

Aptauja „TraceIt lietošanas pieredze"
2. anketa

Iepretim katram jautājumam ir uzskatīti atbildes varianti jeb apgalvojumi, un Jūsu uzdevums ir norādīt, cik stipri tam piekrītat. Attiecīgajās ailēs ir jāieraksta skaitliskās vērtības 0, 1 vai 2, kas atbilst šādam vērtējumam:
0 - nepiekrītu, 1 - daļēji piekrītu, 2 - pilnībā piekrītu.

Ja konstatējat, ka būtu nepieciešams vēl kāds atbildes variants vai varianti, ierakstiet to ailē „Cits".

1. No trasējamības projektā ir atkarīgi šādi darbi	Mans viedoklis *(0-nepiekrītu 1- daļēji piekrītu 2 - pilnībā piekrītu)*
Implementēšana (kodēšana)	
Testu sagatavošana	
Testu izpilde	
Problēmziņojumu apstrāde	
Problēmu novēršana	
Atkļūdošana	
Projekta darbu pārņemšana no kolēģiem	
Audits	
Cits:	

2. Trasējamība sniedz šādus labumus	Mans viedoklis *(0-nepiekrītu 1- daļēji piekrītu 2 - pilnībā piekrītu)*
Ļauj pārvaldīt projektu	
Palīdz realizēt prasības programmas koda formā	
Labāk var veikt kļūdu labojumus	
Uzlabo kvalitāti projektā	
Ja kāda funkcija ir mainīta vairākkārt	
Darbu pārņēmējam būs vieglāk saprast uz attiecas koda gabals	
Cits:	

3. No trasējamības rīka sagaidu	Mans viedoklis *(0-nepiekrītu 1- daļēji piekrītu 2 - pilnībā piekrītu)*
Brīvu meklēšanu pēc jebkuriem atribūtiem	
Atgādinājumi par termiņu	
Automātiskās vēstules darbiniekiem par izmaiņām	
Aptuveni to, kas pašlaik ir TraceIt	

Izmaiņu pārvaldības atbalstu	
Iespēju redzēt, kas, kad un ko ir darījis ar konkrētu vienumu.	
Salīdzinājumā ar Tracelt - vienkāršāks un caurspīdīgāks sasaistes veids	
Ērtu lietotāja saskarni	
Saišu atbalsts/nodrošinājums	
Vēlams, lai tas būtu integrēts izstrādes rīkā(os)	
Jāuztur informāciju par ierakstu veidotājiem vai izpildītājiem	
Cits:	

4. Tracelt ir labi izmantojams šādiem darbiem projektā	Mans viedoklis *(0-nepiekrītu 1- daļēji piekrītu 2 - pilnībā piekrītu)*
Pārraudzībai	
Testu un prasību pārklājuma kontrolei	
Projekta statistikai	
Problēmziņojumu (vai ip) reģistrēšanai	
Problēmziņojumu (vai ip) apstrādei	
Darba uzdevumu un atgādinājumu reģistrēšanai	
Testu pierakstīšanai	
Testu izpildes plānošanai	
Testu izpildes reģistrēšanai	
Lai sagatavotos auditam	
Projekta vadības informēšanai par konstatēto kļūdu novēršanas gaitu	
Cits:	

5. Tracelt ir šādas nepilnības	Mans viedoklis *(0-nepiekrītu 1- daļēji piekrītu 2 - pilnībā piekrītu)*
Nevar veikt brīvu meklēšanu pēc vārdiem un to kombinācijām	
Nav iespēju meklēt vēsturē	
Pietrūkst administratora rokasgrāmatas	
Nav lietotāja rokasgrāmatas	
Nevar veikt automātisku e-pastu pārsūtīšanu adresātam	
Nevar veikt automātisku e-pasta izsūtīšanu cilvēkiem, uz kuriem attiecas ieraksts, ja tas mainās.	
Filtru nosaukumam jābūt unikālam	
Lietotāji saraksta laukos ir ar reģistrācijas vārdiem, bet nevis ar vārdiem un uzvārdiem	
Varēja būt vairāk iespēju uzstādīt noklusējumus	
Pietrūkst veiksmīgu projektu modeļu piemēru	
Nevar pārskatīt visas pieejamās saites vienā ekrānā	
Cits:	

6. Vislabāk Tracelt atbalsta šādu projekta personu darbu	Mans viedoklis *(0-nepiekrītu* *1- daļēji piekrītu* *2 - pilnībā piekrītu)*
Projekta pārvaldnieks	
Projekta kvalitātes pārvaldnieks	
Sistēmanalītiķis	
Programmētājs	
Testētāju grupas vadītājam	
Testētājs	
Cits:	

7. Tracelt vissliktāk piemērots šādu darbu veikšanai	Mans viedoklis *(0-nepiekrītu* *1- daļēji piekrītu* *2 - pilnībā piekrītu)*
Koda rakstīšana	
Cits:	

8. Tracelt vismazāk ir piemērots šādiem projekta darbiniekam	Mans viedoklis *(0-nepiekrītu* *1- daļēji piekrītu* *2 - pilnībā piekrītu)*
sistēmanalītiķim	
tehniskā atbalsta dienesta darbiniekam	
programmētājam	
dokumentētājam	
Cits:	

9. Tracelt vislabāk piemērots šāda tipa projektiem	Mans viedoklis *(0-nepiekrītu* *1- daļēji piekrītu* *2 - pilnībā piekrītu)*
Lieliem projektiem	
Vidējiem projektiem	
Jebkura izmēra projektiem	
Izstrādes projektiem, kas ietver produkta pilnu dzīves ciklu	
Uzturēšanas projektiem	
Testēšanas projektiem	
Kur tiek izmantots ūdenskrituma modelis	
Kur ir atgriezeniskā saite	
Kur ir iesaistīti nopietni resursi	
Cits:	

10. Tracelt vissliktāk ir piemērots šāda tipa projektiem	Mans viedoklis *(0-nepiekrītu* *1- daļēji piekrītu* *2 - pilnībā piekrītu)*
Uzturēšanas vai palīdzības tipa projektiem	
Kur nav atgriezeniskās saites	
Tāda, kur nenotiek programmēšanas darbi	
Maziem projektiem, kur ir vieglāk organizēt komunikāciju bez rīku starpniecības	
Cits:	

11. Trasējamības uzturēšanā Tracelt galvenā priekšrocība salīdzinājumā ar teksta vai MS Excel formātu ir:	Mans viedoklis *(0-nepiekrītu* *1- daļēji piekrītu* *2 - pilnībā piekrītu)*
Nodrošina daudzlietotāju režīmu	
Automatizē darbu	
Vienkāršo darbu	
Specializēta programma ar lielākām iespējām	
Paaugstina lietotāju disciplīnu	
Vieglāk veidot „kvalitātes" atskaites	
Grupas vadītājs var kontrolēt lietotājus	
Pieejamā informācija ļauj vadīt projekta darbu	
Iespējams augsts reaģēšanas ātrums	
Var veidot pielikumus	
Labāka filtrācija	
Saišu mehānisms	
Informācija ir klasificējama un ar filtriem ir aplūkojama dažādos veidos	
Cits:	

12. MS Excel priekšrocības salīdzinājumā ar Tracelt ir	Mans viedoklis *(0-nepiekrītu* *1- daļēji piekrītu* *2 - pilnībā piekrītu)*
Labāk pārraugāma tabulas veida informācija	
Mazā projektā visu nepieciešamo var salikt vienā tabulā/ekrānā	
Cits:	

13. Ieteiktu Tracelt izmantot citos projektos, jo:	Mans viedoklis *(0-nepiekrītu* *1- daļēji piekrītu* *2 - pilnībā piekrītu)*
Kā interneta programmu var lietot no jebkuras vietas	
Interneta programmu var lietot bez sistēmas konfigurēšanas	
Ir pārbaudīts darbā lielā projektā	
Ir laba rīka atbalsta komanda	
Ir lielākas iespējas par MS Excel risinājumu	

Nav jāiegādājas licences	
Darbs ir iespējams vairākiem cilvēkiem reālā laika režīmā	
Ir iespēja pielāgot projekta vajadzībām	
Var izmantot ne tikai trasējamībai, bet arī datubāzes prototipa sagatavošanai	
Cits:	

14. Neieteiktu TraceIt izmantot citos projektos, jo:	Mans viedoklis *(0-nepiekrītu 1- daļēji piekrītu 2 - pilnībā piekrītu)*
Nav skaidra uzņēmuma nostāja šī rīka tālākajai attīstībai	
Mazu projektu gadījumā nav obligāti uzturēt trasējamību	
Cits:	

15. TraceIt nepieciešami šādi uzlabojumi:	Mans viedoklis *(0-nepiekrītu 1- daļēji piekrītu 2 - pilnībā piekrītu)*
Uzlabota saskarne, lai būtu vienkāršāk sasaistīt vienumus	
Cits:	

3.kārta – pavadvēstule

Pateicamies par dalību TraceIt aptaujas 2.kārtā. Šobrīd piedāvājam piedalīties noslēdzošajā kārtā, kuras anketa ir veidota uz 2.kārtas rezultātu bāzes.
Atbilstoši tam, kā vērtējāt jautājuma atbildes variantu jeb viedokli, katrs ieguva noteiktu punktu skaitu, kas to ļāva ierindot vienā no trim svarīguma līmeņiem - augstā, vidējā vai zemā. Līmeņa ietvaros lielāku punktu skaitu ieguvušie varianti ir pozicionēti augstāk.

3.kārtā Jums ir iespēja aplūkot aptaujas starprezultātu un vēl veikt korekcijas iegūtajos rezultātos - viedokļa variantu ir iespējams pārcelt uz zemāku vai augstāku svarīguma līmeni. Papildus tam, atsevišķiem jautājumiem pēc aptaujas 2.kārtas ir nākuši klāt jauni viedokļi - tiem jānorāda vēlamā piederība pie svarīguma līmeņa.

Tas arī viss!

Lūgums anketu aizpildīt un nosūtīt uz e-pasta adresi [...].

Ar cieņu,

[...]

3.kārta - anketa

Aptauja "TraceIt lietošanas pieredze"
3. anketa

Šī ir aptaujas noslēguma anketa, kas iegūta, balstoties uz 2.kārtas rezultātiem. Viedokļi atbilstoši sniegtajiem vērtējumiem ir sagrupēti trijos svarīguma līmeņos - zems, vidējs un augsts. Šajā reizē Jums ir iespēja:

- aplūkot, kāds ir aptaujas dalībnieku kopējais viedoklis pēc 2.kārtas,
- koriģēt 2.kārtā iegūto starprezultātu, ja Jums ir atšķirīgs viedoklis par norādītā viedokļa svarīguma līmeni.

1) Ja uzskatāt, ka norādītais viedokļa variants neatbilst attiecīgajam svarīguma līmenim, labajā pusē esošajās rūtiņās ar "x" atzīmējiet, vai pazemināt, vai paaugstināt šo svarīgumu.
2) Papildus tam, atsevišķiem jautājumiem pēc aptaujas 2.kārtas ir nākuši klāt jauni viedokļi. Jaunajiem viedokļiem, lūdzu, norādiet, jūsuprāt, atbilstošāko svarīguma līmeni.

1. No trasējamības projektā ir atkarīgi šādi darbi		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	Problēmziņojumu apstrāde		
	Audits		
	Testu sagatavošana		
	Problēmu novēršana		
	Projekta darbu pārņemšana no kolēģiem		
	Testu izpilde		
vidējs	Atkļūdošana		
	Implementēšana (kodēšana)		

Papildus viedokļi 1.jautājuma kontekstā	Svarīguma līmenis		
	zems	vidējs	augsts
Prasību specificēšana			
Projektēšana			

2. Trasējamība sniedz šādus labumus		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	Uzlabo kvalitāti projektā		
	Ļauj pārvaldīt projektu		
	Ja kāda funkcija ir mainīta vairākkārt		
	Labāk var veikt kļūdu labojumus		
vidējs	Palīdz realizēt prasības programmas koda formā		
	Darbu pārņēmējam būs vieglāk saprast uz attiecas koda gabals		

Papildus viedokļi 2.jautājuma kontekstā	Svarīguma līmenis		
	zems	vidējs	augsts

Labāk var kontrolēt kļūdu labojumus			
Darbu pārņēmējam būs daudz vieglāk saprast, uz ko attiecas tests, prasība			

3. No trasējamības rīka sagaidu		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	Ērtu lietotāja saskarni		
	Jāuztur informāciju par ierakstu veidotājiem vai izpildītājiem		
	Saišu atbalsts/nodrošinājums		
	Brīvu meklēšanu pēc jebkuriem atribūtiem		
	Aptuveni to, kas pašlaik ir Tracelt		
	Izmaiņu pārvaldības atbalstu		
	Iespēju redzēt, kas, kad un ko ir darījis ar konkrētu vienumu.		
vidējs	Atgādinājumi par termiņu		
	Vēlams, lai tas būtu integrēts izstrādes rīkā(os)		
	Salīdzinājumā ar Tracelt - vienkāršāks un caurspīdīgāks sasaistes veids		

4. Tracelt ir labi izmantojams šādiem darbiem projektā		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	Problēmziņojumu (vai IP) reģistrēšanai		
	Problēmziņojumu (vai IP) apstrādei		
	Projekta statistikai		
	Testu izpildes reģistrēšanai		
	Testu un prasību pārklājuma kontrolei		
	Projekta vadības informēšanai par konstatēto kļūdu novēršanas gaitu		
	Pārraudzībai		
	Testu pierakstīšanai		
vidējs	Testu izpildes plānošanai		
	Lai sagatavotos auditam		
	Darba uzdevumu un atgādinājumu reģistrēšanai		

Papildus viedokļi 4.jautājuma kontekstā	Svarīguma līmenis		
	zems	vidējs	augsts
Atkļūdošanai			

5. Tracelt ir šādas nepilnības		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu

augsts	Nevar veikt brīvu meklēšanu pēc vārdiem un to kombinācijām		
vidējs	Nav lietotāja rokasgrāmatas		
	Nav iespēju meklēt vēsturē		
	Pietrūkst administratora rokasgrāmatas		
	Nevar veikt automātisku e-pastu pārsūtīšanu adresātam		
	Nevar veikt automātisku e-pasta izsūtīšanu cilvēkiem, uz kuriem attiecas ieraksts, ja tas mainās.		
	Filtru nosaukumam jābūt unikālam		
	Nevar pārskatīt visas pieejamās saites vienā ekrānā		
	Varēja būt vairāk iespēju uzstādīt noklusējumus		
	Lietotāji saraksta laukos ir ar reģistrācijas vārdiem, bet nevis ar vārdiem un uzvārdiem		
	Pietrūkst veiksmīgu projektu modeļu piemēru		

6. Vislabāk Tracelt atbalsta šādu projekta personu darbu		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	Testētāju grupas vadītājam		
	Testētājs		
	Projekta kvalitātes pārvaldnieks		
	Projekta pārvaldnieks		
vidējs	Sistēmanalītiķis		
	Programmētājs		

7. Tracelt vissliktāk piemērots šādu darbu veikšanai		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	Koda rakstīšana		

8. Tracelt vismazāk ir piemērots šādiem projekta darbiniekam		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	tehniskā atbalsta dienesta darbiniekam		
vidējs	dokumentētājam		
	sistēmanalītiķim		
	programmētājam		

9. Tracelt vislabāk piemērots šāda tipa projektiem		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	Testēšanas projektiem		
	Vidējiem projektiem		
	Izstrādes projektiem, kas ietver produkta pilnu dzīves ciklu		
	Lieliem projektiem		
	Uzturēšanas projektiem		
vidējs	Kur ir iesaistīti nopietni resursi		
	Jebkura izmēra projektiem		
	Kur tiek izmantots ūdenskrituma modelis		
	Kur ir atgriezeniskā saite		

Papildus viedokļi 9.jautājuma kontekstā	Svarīguma līmenis		
	zems	vidējs	augsts
Prototipēšanas projektos			
Projektiem, kas ietver vairākus dzīves cikla procesus			

10. Tracelt vissliktāk ir piemērots šāda tipa projektiem		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
vidējs	Kur nav atgriezeniskās saites		
zems	Maziem projektiem, kur ir vieglāk organizēt komunikāciju bez rīku starpniecības		
	Uzturēšanas vai palīdzības tipa projektiem		
	Tāda, kur nenotiek programmēšanas darbi		

11. Trasējamības uzturēšanā Tracelt galvenā priekšrocība salīdzinājumā ar teksta vai MS Excel formātu ir:		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	Nodrošina daudzlietotāju režīmu		
	Specializēta programma ar lielākām iespējām		
	Saišu mehānisms		
	Grupas vadītājs var kontrolēt lietotājus		
	Pieejamā informācija ļauj vadīt projekta darbu		
	Informācija ir klasificējama un ar filtriem ir aplūkojama dažādos veidos		
	Vieglāk veidot „kvalitātes" atskaites		
vidējs	Automatizē darbu		
	Vienkāršo darbu		
	Labāka filtrācija		
	Paaugstina lietotāju disciplīnu		
	Iespējams augsts reaģēšanas ātrums		
	Var veidot pielikumus		

12. MS Excel priekšrocības salīdzinājumā ar Tracelt ir		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
vidējs	Mazā projektā visu nepieciešamo var salikt vienā tabulā/ekrānā		
	Labāk pārraugāma tabulas veida informācija		

13. Ieteiktu Tracelt izmantot citos projektos, jo:		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
augsts	Kā interneta programmu var lietot no jebkuras vietas		
	Darbs ir iespējams vairākiem cilvēkiem reālā laika režīmā		
	Nav jāiegādājas licences		
	Ir iespēja pielāgot projekta vajadzībām		
	Interneta programmu var lietot bez sistēmas konfigurēšanas		
	Ir lielākas iespējas par MS Excel risinājumu		
	Ir pārbaudīts darbā lielā projektā		
	Ir laba rīka atbalsta komanda		
zems	Var izmantot ne tikai trasējamībai, bet arī datubāzes prototipa sagatavošanai		

14. Neieteiktu Tracelt izmantot citos projektos, jo:		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
vidējs	Nav skaidra uzņēmuma nostāja šī rīka tālākajai attīstībai		
zems	Mazu projektu gadījumā nav obligāti uzturēt trasējamību		

Papildus viedokļi 14.jautājuma kontekstā	Svarīguma līmenis		
	zems	vidējs	augsts
Tīmekļa programma lēnāka par lokāli uzstādītām programmām			

15. Tracelt nepieciešami šādi uzlabojumi:		Uzskatu, ka nepieciešams: *(atzīmēt ar "x")*	
Svarīguma līmenis		pazemināt svarīgumu	paaugstināt svarīgumu
vidējs	Uzlabota lietotāja saskarne, lai būtu vienkāršāk sasaistīt vienumus		

Papildus viedokļi 15.jautājuma kontekstā	Svarīguma līmenis		
	zems	vidējs	augsts
Paplašinātas grafiskās iespējas aplūkot saites starp vienumiem			
Mehānisms PZ lietotāja izvēlētai kārtošanai un grupēšanai			

Paldies!

11.4. TraceIt izmantošana projektos

Izziņa no a/s DATI par TraceIt izmantošanu projektos
[pievienota uz atsevišķas lapas]

DATI
Exigen Group

Akciju sabiedrība "DATI"
Reģ. nr.: LV 40003275598
Ganību dambis 17a
LV-1045 Rīga
Tālr.: 7067777
Fakss: 7381218

IZZIŅA
Rīgā

14.03.2005. Nr. 313

Latvijas Universitātes
Datorzinātnes nozares promocijas padomei

A/s DATI uzņēmumā RITI Programmatūras testēšanas un kvalitātes nodrošināšanas laboratorijā asistenta Mārtiņa Gilla vadībā 2001.gadā tika uzsākta eksperimentālā trasējamības rīka TraceIt izstrāde.

Laika periodā no 2001. līdz 2004.gadam rīks TraceIt tika izmantots šādos 22 a/s DATI projektos: PVIS, BCIS, SB, SB2, VID-DN, Transcend, CDMS, LTU, Housing (2 projekti), Musichop, Telehansa, RIWOS, NAIS, Prese-Tev, Hipo-DN, SAB-DIUS, UB-lietojamība un Finix (4 apakšprojekti).

Programmizstrādes projektu pārvaldnieki labprāt lieto TraceIt un atzīst, ka rīks būtiski uzlabo izstrādes disciplīnu, atvieglina procesus, kur trasējamība ir īpaši nozīmīga, un veicina programmproduktu kvalitāti.

A/s DATI prezidents, LZA goda doktors V.Lokenbahs

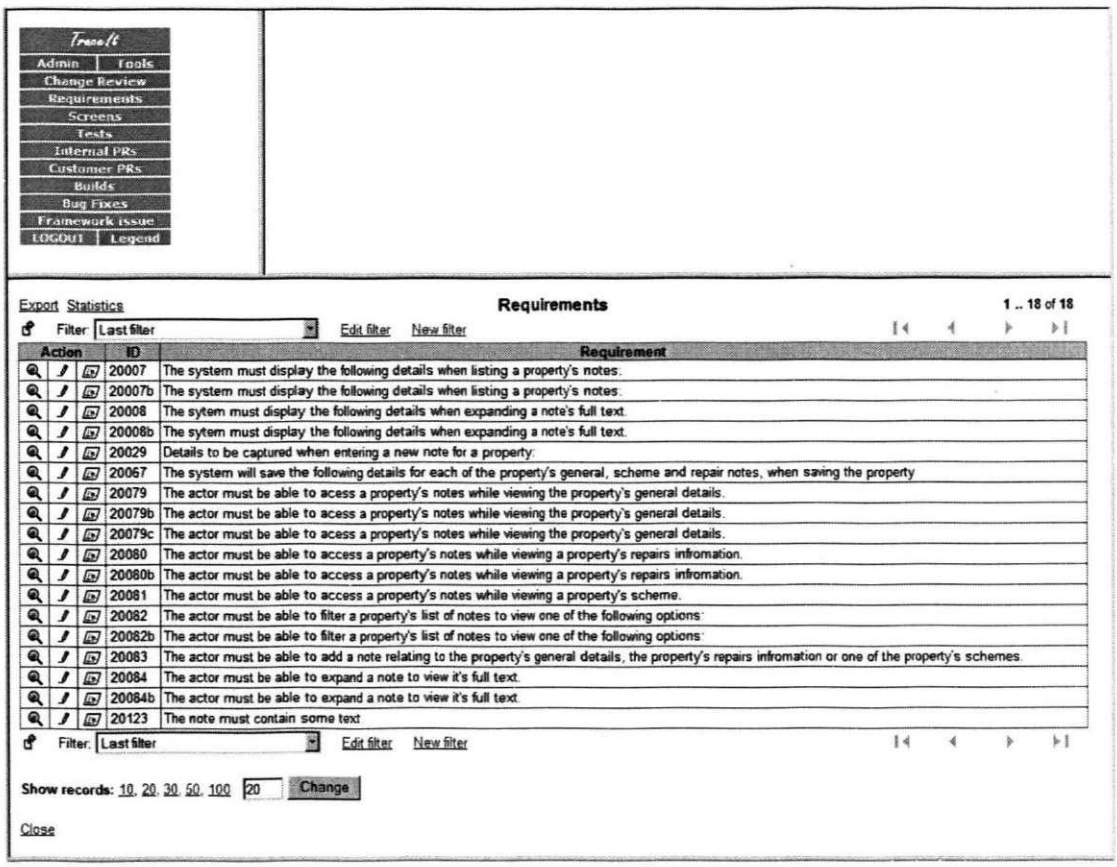
11.5. Tracelt funkcionalitāte

Funkcionalitātes satura rādītājs

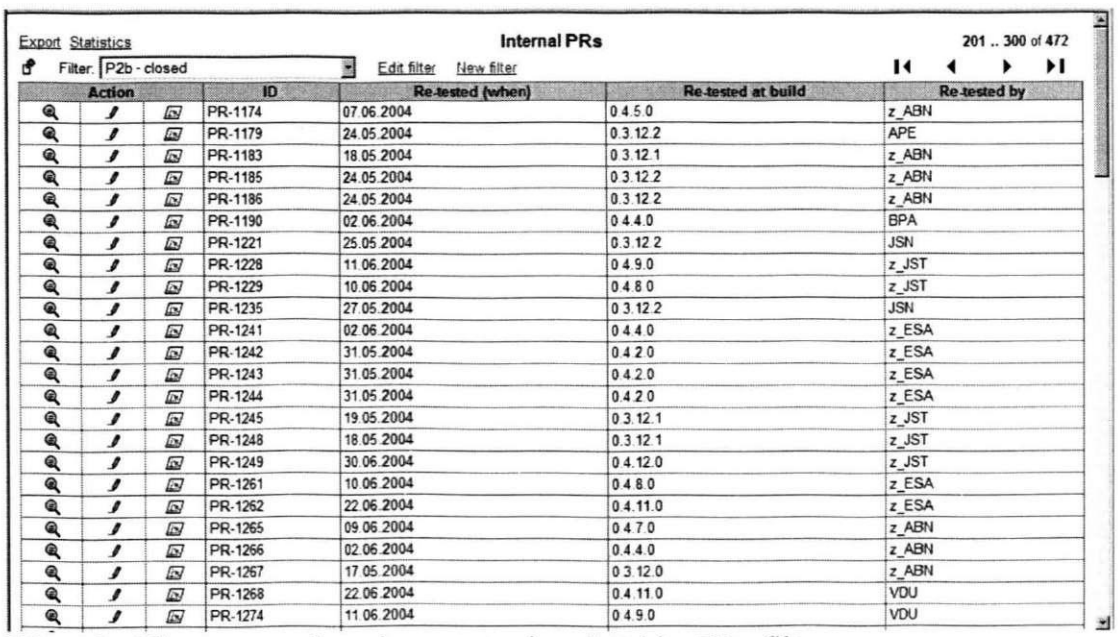
12.4. Informācija par aktīvajām sesijām

13. Pieejamā palīginformācija

Tracelt ekrāni – paraugs no reāla projekta



E1.attēls. Kopskats.



E2.attēls. Vienumu saraksts, izmantots viens brīvi izvēlēts filtrs.

Export Statistics

Internal PRs

1 .. 38 of 38

Filter: Res priority 2 Edit filter New filter

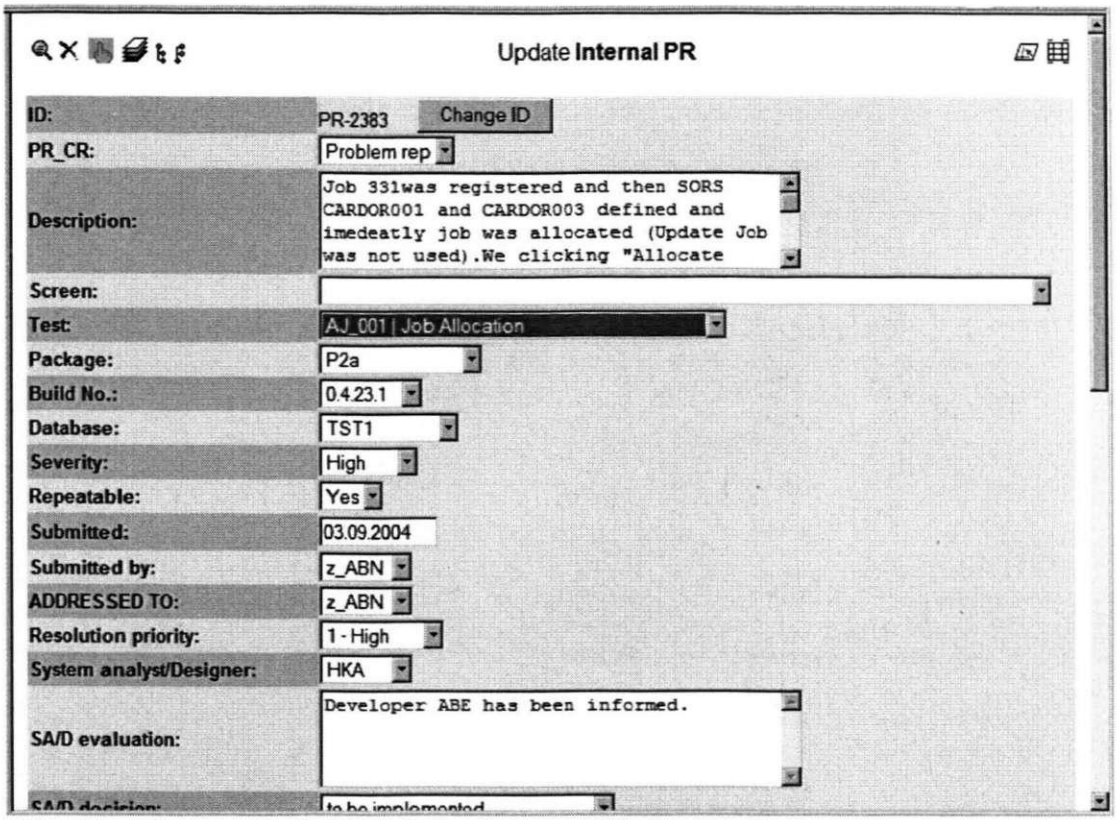
Action	ID	Description	Submitted by	SA/D decision
	PR-1543	Define Job - Search page. 1. User takes status OnAssessment. Button Search. Program returns list of jobs. 2. User goes to the last page in the grid (>>) and see that list includes job no.710. 3. User takes status All, job Number - 710. Button Search. Search result is empty. (This is because program sets Page 6 of 1). If user selects 1st page in the grid (<<) then job is in the list.	AAL	
	PR-1651	Property Enquiry. Then user select 'EventMaintenance' part from PropertyRepairsHistoryJobHistory screen - the navigation buttons disappears.	z_ESA	
	PR-1812	Register Job - Property Search - Item History - "View Item" in Data Menu item. After every selection of menu item, contents of Description fields in Item Details and Supply Item Details is duplicated.	AAL	
	PR-1867	When the Invoice search finishes with many entrys and pages (for example 17 pages) and user navigate to pages 15 or 16 or 17, then those pages does not have an any entry and system returns a masage "No results found". See attach.	z_DKE	
	PR-1910	Define Job page - banner - Repairs Status Info icon. Job no. 34. Repairs Status "CU" and Info icon are displayed in the banner. If user clicks the icon nothing happens.	AAL	to be implemented
	PR-1984	Compliance Management Complete compliance Plan page does not save the edited Job details (Inspected, Non-conformant, ITP Non-conformant. Click on save, then open the page again for the same Compliance plan - the data is not saved.	z_JRA	
	PR-2059	Allocate Job - Search page. Sequence of actions is as follows. We find a job. Allocate it. Program returns to search page - that is correct. But this page can't find anything more. If to start Allocate job from menu then find work again (1 time only again).	AAL	to be implemented
	PR-2072	Modify Invoice - search page. Please have a look at the image in the attachment: "Page 6 of 2". Please correct the error. Sequence of actions was as follows. 1. Select "Modify Invoice" from menu. 2. Press search. 3. Press Last page button. 4. Change number of rows for the page from default value to 20. 5. Press search button. NOTE: Agris Belte could give you advice in fixing the error if it will be needed.	AAL	to be implemented
	PR-2120	When the Invoice Line Notes in Data menu is selected in the "Invoice Override Reason" page nothing happens	APE	
	PR-2160	If the hold reason is selected in the "Cancel Job Search" screen then all jobs (not only jobs on hold) appears in search results. (Search results must appear as per requirement 081-00012)	APE	to be implemented
	PR-2175	user make invoice approve and certification (certification failed), then go to Invoice Audit Trail page. Page is displayed with entry for Invoice Certification Failed.¬ Invoice Reference = Invoice reference of the invoice just entered. Status Change Date (Audit Date = Date invoice certification failed) User Id = User Id of the user who approved the invoice User	TPE	to be implemented

E3.attēls. Tas pats vienumu tips, kas E2.attēlā, izmantots cits brīvi izvēlēts filtrs.

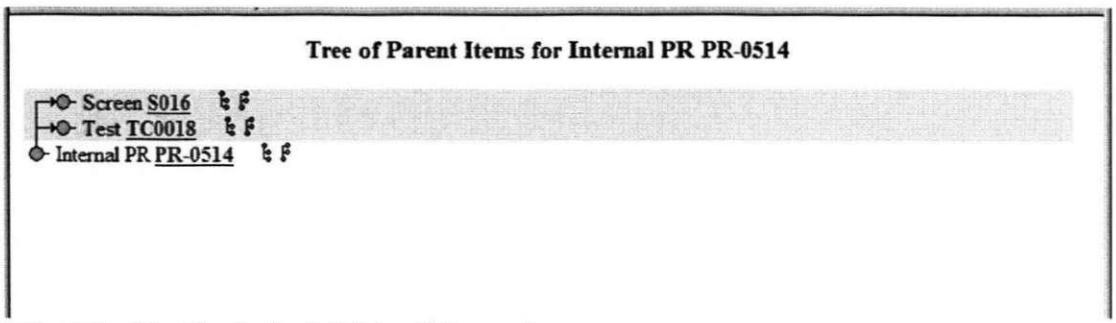
View Internal PR

ID:	PR-2468
PR_CR	Problem rep
Description	User ALLFUNC1. Bulk Print Pre-Inspection Ticket. Job Search screen - Inspector DDLB only lists ""All""
Screen	Print Pre-Inspection Ticket
Test	BP_PIT_001 |
Package	redefined P5a
Build No.	0.4.28.0
Database	TST2
Severity	Low
Repeatable	Yes
Submitted	23.09.2004
Submitted by	VDU
ADDRESSED TO	
Resolution priority	
System analyst/Designer	
SA/D evaluation	
SA/D decision	
Programmer	JMU
PRG evaluation	Problem solved. Please retest with build 0.4.31.0 or later.
SA/D/PRG evaluated	
PR/CR status	Closed
Re-tested (when)	11.10.2004
Re-tested at build	0.5.1.0
Re-tested by	VDU
External recipient evaluation	
External recipient	
Model version	
Model	
Reqs document (file)	
Sub-system	
Comment	
Status:	Actual
Attachments:	

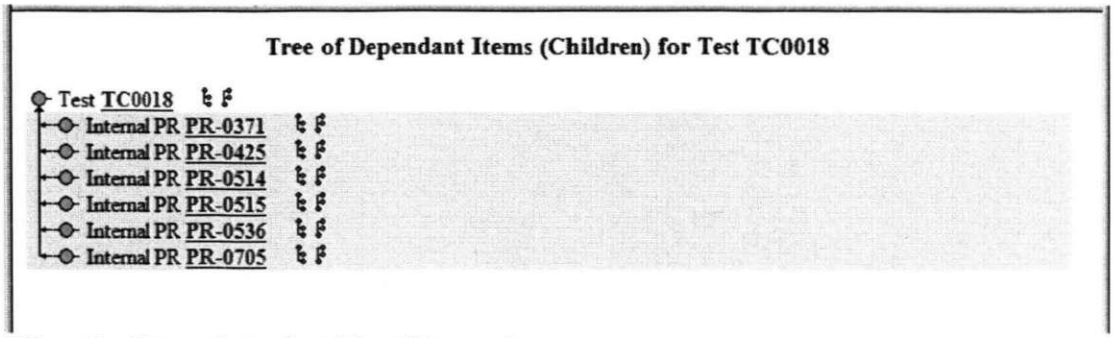
E4.attēls. Vienuma individuālais ieraksts – apskates režīms.



E5.attēls. Vienuma individuālais ieraksts – rediģēšanas režīms.



E6.attēls. Vecāku koks brīvi izvēlētam vienumam.



E7.attēls. Bērnu koks brīvi izvēlētam vienumam.

Relations

From *Internal PR* **PR-0514** to its parents:

Screen	ID	Rel.name
S016		is related to

Test	ID	Name	Last Executed by	Build No	Execution Status	Rel.name
TC0018	Modify Property (Property Class 7)		z_DKE	0.4.17.12	Pass (unconditional)	initiated by

Internal PR	ID	Build No.	Severity	Submitted	ADDRESSED TO	PR/CR status	Re-tested (when)	Rel.name

From children to *Internal PR* **PR-0514**:

Internal PR	ID	Build No.	Severity	Submitted	ADDRESSED TO	PR/CR status	Re-tested (when)	Rel.name

Framework issue	ID	Rel.name

Bug Fix	ID	Description	Fixed (when)	Rel.name

Cancel

E8.attēls. Saišu logs brīvi izvēlētam vienumam.

Internal PRs : fields ?

To item types Add new field To field types

Action	Field name	Influences Pendings	Field type	Description	Physical type
	PR_CR	No	PR_CR	Problem Report, Change Request	Enumerated
	Description	Yes	Description	Description of an item	Fixed (Description)
	Screen	No	Screen	Relation from PR to Screen where the problem is located	Relation (Single)
	Test	No	Test	Relation from PR to Test which caused the problem	Relation (Single)
	Package	No	Package	Packages of Housing application	Enumerated
	Build No.	No	BUILD	Build number of Housing application	Enumerated
	Database	No	database	Instance of database	Enumerated
	Severity	No	Severity	Problem severity	Enumerated
	Repeatable	No	YN	Yes, No	Enumerated
	Submitted	No	Date		Date
	Submitted by	No	User		User
	ADDRESSED TO	No	User		User
	Resolution priority	No	Priority	Priority of problem resolution	Enumerated
	System analyst/Designer	No	User		User
	SA/D evaluation	No	Long text		Text
	SA/D decision	No	SA/D decision	SA/D decision	Enumerated

E9.attēls. Vienuma tipa definīcijas paraugs.

Item History

Created on: 10.05.2004 12:29 (GMT+2) **by** BPA (Bosko Pavlovic)

	User	Changed at	Field	Old value	New value
	BPA	09.06.04 11:27:33	ADDRESSED TO:	*BPA*	
			PR/CR status:	*Open*	Closed
			Re-tested (when):	*00.00.0000*	09.06.2004
			Re-tested at build:		0.4.7.0
			Re-tested by:		BPA
			External recipient evaluation:		fixed.
	ABE	02.06.04 16:31:14	ADDRESSED TO:	*ABE*	BPA
			Programmer:		ABE
			PRG evaluation:		Corrected
			SA/D/PRG evaluated:	*12.05.2004*	02.06.2004
	HKA	12.05.04 10:00:02	ADDRESSED TO:	*HKA*	ABE
			System analyst/Designer:		HKA
			SA/D evaluation:		To be fixed.
			SA/D decision:		to be implemented
			SA/D/PRG evaluated:	*00.00.0000*	12.05.2004

E10.attēls. Ieraksta apstrādes vēstures apskate.

Roles

	Role name
	Administrator
	User
	Tester
	Team leader
	Developer
	Framework team
	import
	Customer tester - LV
	Builder
	Installer
	Customer tester - AU

New record | Cancel

E11.attēls. Lietotāja lomu definēšanas logs.

Role item type permissions

Role: Team leader

Does not belong to		Belongs to
Delete Builds	>>	Attach Builds
Delete Changes	<<	Attach Changes
Delete DATI PR/CR		Detach Builds
Delete Framework issue		Detach Changes
Delete Requirements		Edit Builds
Delete Screens		Edit Changes

Close

E12.attēls. Tiesību piešķiršana lomai.