TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Siim Medijainen
130379IABMM

TEHNILISE VÕLA PRIORITISEERIMINE SQALE MEETODIGA ETTEVÕTTE NÄITEL

magistritöö

Juhendaja: Deniss Kumlander
vanemteadur

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Siim Medijainen

08.05.2017

Annotatsioon

Magistritöö eesmärk on mõõta ning prioritiseerida ettevõtte projektides sisalduv tehniline võlg kasutades ning adapteerides selleks SQALE meetodit. Selle käigus mõõdetakse tööriista SonarQube kasutades projektides esinevat tehnilise võla hulka ning uuritakse selle võimalikke tekkepõhjuseid. Uuritakse, kas SQALE meetodil leitud tehnilise võla prioriteedid vastavad eksperthinnangutele ning parandatakse tehnilise võla prioritiseerimise protsessi.

Magistritöö tulemuseks on antud ettevõtte kontekstis täiendatud metoodika, mida saab kasutada tehnilise võla prioritiseerimiseks PHP-s kirjutatud rakendustes. Lisaks leitakse tulemusena, et enamik arenduse käigus tekkinud tehnilisest võlast satub sinna teadlikult ning tuleneb praktilisest vajadusest. Välja pakutakse ideid protsessi kujundamiseks, kuidas iteratsioonides töötavad tarkvara arendavad meeskonnad saavad prioritiseerida ning vähendada tehnilist võlga süstemaatiliselt.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 66 leheküljel, 5 peatükki, 11 joonist, 13 tabelit.

Abstract

Prioritising technical debt based on company x using SQALE methology

The aim of the thesis is to measure and prioritise technical debt in the projects of a company, using SQALE method and adapting it to the company needs. In order to achieve that goal technical debt is measured using SonarQube software, as an addition the possible causes of thechnical debt are investigated. It is researched if the results from prioritising technical debt with SQALE method are complying to the optinion of experts, the process of prioritising technical debt is improved on the way.

The result of current master's thesis is an improved methodology in current company's context, which can be used for evaluating technical debt in applications written in PHP. It is also found that most of the technical debt accumulating during the development process is put there deliberately by developers and is driven by the pragmatic need. As a result some ideas to create a process how software development teams working in iterations can prioritise and reduce the technical debt in their projects is proposed.

The thesis is in estonain and contains 66 pages of text, 5 chapters, 11 figures, 13 tables.

Lühendite ja mõistete sõnastik

SQALE	Tarkvara elutsüklil põhinev kvaliteedi hindamine. *Ing k Software Quality Assessment based on Lifecycle Expectations*
PHP	Programmeerimiskeel, hüperteksti preprotsessor *ing k Hypertext Preprocessor*
OWASP	Avatud veebirakenduste turvaprojekt *Ing k Open Web Application Security Project*
SBI	SQALE äriindeks *ing k Sqale business index*
SQI	SQALE kvaliteediindeks *ing k SQALE quality index*
PSR	PHP keele standardite soovitused
CWE	Levinud turvaaukude kataloog *ing k Common weakness enumeration*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Enamikes tehnilist võlga käsitlevates allikates viidatakse esmakordsele mainimisele Ward Cunninghami poolt aastal 1992[1], mis kirjeldas projekti koodi esimese iteratsiooni väljastamist justkui võla võtmisega, mis kiirendab üldist arendust ning tuleb kiirelt tasuda kõnealuse koodi ümberkirjutamisega. Mitte-ümberkirjutamine peidab endas ohtu hilisemate arenduste suuremas keerukuses ning ajakulus. Tänapäevasemates käsitlustes on tehnilise võla kontseptsioon muutunud, ning see kirjeldab kõikvõimalikke puudujääke nii süsteemide lähtekoodis, arhitektuuris, infrastruktuuris jmt valdkondades. Tehnilise võla uurimine on viimasel kümnendil plahvatuslikult kasvanud ning arvestades selle metafooriga seostatud valdkondi - uuring[2] eristab 10 erinevat liiki tehnilist võlga ning mõisteid – sama allikas pakub 24 erinevat tehnilise võlaga seotuvat mõistet, jätkub see tõenäoliselt edaspidi.

Vanemate „legacy" tarkvaraprojektidega käib praktiliselt alati kaasas märkimisväärne kogus tehnilist võlga. Mõnikord on projekt saadud päranduseks varasemalt ettevõttest lahkunud arendajate käest. Sellistes olukordades on tihti vaja anda hinnang hetkeolukorrale tarkvara kvaliteedi mõistes ning kaardistada kriitilisemad kohad lähtekoodis ülevaate saamiseks.

Tehnilise võla põhiline dilemma seisneb selle nähtamatuses tarkvara tellijale, hetkel korrekteselt töötav tarkvara võib olla arendajatele sedavõrd probleemne, et edasised arendused selle juures võivad osutuda raskeks või võimatuks.

1.1 Ülesandepüstitus

Autorile teadaolevalt ainuke tehnilist võlga mainiv töö TTÜ digiraamatukogust [21] käsitleb tehnilise võla mõistet ning antimustreid, mis viivad tehnilise võla tekke ning kogunemiseni. Tehnilise võla prioritiseerimiseks on olemas mitmeid erinevaid strateegiaid, tehnilise võla mõiste kaardistamisega seotud allikas[2] viitab 5 uurimustöö baasil neljale võimalikule kategooriale: kulu vs väärtuse analüüs sh SQALE meetod[8],

kõrgema võlaga (lävendipõhine) komponentide prioritiseerimine[11], kõikide defektide üle portfoolio pidamine[31] ning kõrgema intressimääraga võla prioritiseerimine. Samas kui allikaid, mis käsitlevad nende metoodikate praktilist rakendamist on raske või võimatu leida. Käesolev töö keskendub tehnilise võla määramisele ning prioritiseerimise taktikale, kasutades kulu vs väärtuse hindamiseks SQALE meetodit. Eesmärk on välja selgitada meetodi praktilisel rakendamisel esilekerkivad probleemid ning anda hinnang tulemuste aktuaalsusele tehnilise võla prioritiseerimise osas.

Olles ise kahe arendusmeeskonna tehniline juht ning töötades projektidega, mis on võrdlemisi pikad (5-6+ aastat) ning kus peaks olema ülioluline rõhk hallatavusel. Näeb töö autor oma igapäevatöös korduvalt olukordi, kui tehniline võlg koguneb piirini, kus arendajad keelduvad mingisse süsteemi osasse muudatusi tegemast, kuna on suur oht, et ühe muudatuse sisseviimine põhjustab suure tõenäosusega kriitilisi probleeme mõnes teises osas. Ehk tehnilise võla termineid kasutades on mingi osa süsteemist pankrotis. Sellest tuleneb vajadus töötada välja protsess, millega oleks võimalik tuvastada tarkvaraprojekti juures kogu tehnilise võla hulka ning tehnilise võla mõistes huvi pakkuvaid süsteemi osiseid, kus võlg on “kuhjuma” hakanud. Teisalt tuleb arvestada, et refaktoorimiseks on ressursid piiratud ning see sunnib meid otsima kohti, mis võimalikult väikese ressursside kuluga annaks võimalikult rohkemat ärilist väärtust.

Käesoleva magistritöö ülesandeks on mõõta tehnilise võla kogus antud ettevõtte projektides ning leida prioriteetsed kohad selle vähendamiseks SQALE meetodit kasutades ning seda meetodit vajadusel adapteerides. Meetodi tulemusi kontrollitakse projektidega töötavate arendajate eksperthinnanguga. Ülesande käigus peame leidma vastused järgmistele küsimustele:

- Kui suur on projektides hetkel tehnilise võla kogus?
- Millest peaksid olema prioriteedid tehnilise võla vähendamisel?
- Kas SQALE meetodil tehnilise võla prioritiseerimine vastab eksperthinnangu nägemusele tehnilise võla prioriteetidest ?
- Mis on koodis leiduva tehnilise võla peamine tekkepõhjus või tekkepõhjused?

1.2 Metoodika

Tehnilise võla koguse mõõtmiseks kasutatakse lähtekoodi staatilist ning dünaamilist analüüsi SonarQube tööriista kasutades. Leitud tehnilise võla prioritiseerimiseks kasutatakse SQALE meetodit. SQALE meetodi adapteerimiseks antud ettevõtte projektide puhul ning tehnilise võla tekkepõhjuste selgitamiseks kasutatakse projekti arendajatega läbiviidud poolstruktureeritud intervjuusid.

1.3 Töö struktuur

Magistritöö koosneb kolmest sisulisest osast. Esimeses osas antakse üldine ülevaade tehnilise võla mõistest ning taustsüsteem mõistmaks, mida mõistame terminiga "tehniline võlg" ning ülevaade sellest, kuidas on seda senini käsitletud.

Teises osas antakse teoreetiline ülevaade rakendatavast SQALE meetodist ning käsitletakse selle alternatiive.

Kolmandas osas analüüsitakse kahe projekti lähtekoodi staatilise analüüsi vahenditega ning leitakse SQALE meetodit kasutades olulisemad kitsaskohad. Tulemuste edukust kontrollitakse, viies läbi intervjuud arendajatega, uurides, mil määral on meetodi rakendamine edukas olnud.

2 Tehnilise võla mõiste käsitlus ning taustsüsteem

2.1 Tehnilise võla mõiste

Tarkvaraprojektides koguneb tehniline võlg juhul, kui pikaajalised eesmärgid vahetatakse lühiajaliste eesmärkide vastu. Lähtekoodiga seotud tehniline võlg võib koguneda läbi aja ning muuta tarkvara arenduse pikas perspektiivis võimatuks. Tehnilise võla puhul on tihti analoogiana toodud finantsmaailma vastavaid termineid näiteks – enamik allikaid mainivad intressi [26]. Võetakse "laen", võimaldamaks koheselt midagi saavutada arvestades, et hiljem tuleb selle eest maksta teatud intressi. Ka on sarnane pankroti[3] mõiste - kui laenu on võetud niivõrd suurel hulgal, et ostujõud läheneb nullile ning kogu olemasolev ressurss kulutatakse intresside tagasimaksetele.

Tehnilisest võlast rääkides tuleb alati mainida kompromisse - tuleb vastu võtta otsus, millised osad süsteemist on sedavõrd olulised, et nendes tehnilise võla võtmine võib hilisemate arenduste lisades tuua rohkem kahju kui kasu. Akbarinasaji[28] kirjeldab seda valikut kui tasakaalu kolme komponendi vahel: aeg, kvaliteet ning hind. Nendest kolmest komponendist on meil võimalik valida kaks.

Olenevalt projektide tüübist, võib mõnikord tehnilise suure võla võtmine olla põhjendatud. Näiteks mõnikord juhtub, et loodud funktsionaalsust ei hakatagi kunagi klientide poolt kasutama. Sellisel juhul ei oma loodud tarkvara sisemine kvaliteet tõepoolest suuremat tähtsust. Näitena võib tuua ka mõne võib-olla tehniliselt lihtsama süsteemi, mis võetakse kasutusse, mis täidab oma nõuded ning mida enam kunagi tarkvara elutsükli jooksul ei täiendata.

Tehniline võlg on enamasti tulemuseks, kui proovitakse leida tasakaalu parimate praktikate vahel süsteemi luues ning kõigi teiste eduka tarkvaraprojektiga kaasas käivate mõistete vahel: lanseerimiskuupäev, kasutada olevate tööriistade hind ning kaasatud tarkvara loojate oskused. Näiteks puudulik või sootuks puuduv dokumentatsioon, aja

puudusel kirjutamata jäänud ühiktestid ning muu säärane. Enamasti on see lõppkasutajale nähtamatu, kuid tarkvara pikaajalisele hallatavusele ülitähtis komponent.

Enamik tehnilist võlga käsitlevad allikad nõustuvad, et tehnilist võlga ei saa liigitada binaarselt positiivseks või negatiivseks nähtuseks. Antud mõistet käsitlevates allikates tuuakse väga tihti võrdluseks laenu või võla mõistet majandussektorist - laenuga kaasnevad intressimaksed, mis tagastades koos põhiosaga teevad kokku rohkem, kui esialgne laenusumma. Sarnasus on ka selles, mis juhtub juhul kui laenu tagasi ei maksta - tulemuseks on pankrot või mõne väärtusliku tagatisena kasutatud eseme võõrandamine. Erinevustena võib välja tuua konkreetse maksegraafiku puudumise - tehnilise võlaga seotud intressi tuleb maksta juhul, kui keegi puutub kokku tehnilisest võlast tulenevate puudustega - olgu selleks refaktoorimine, enne uue funktsionaalsuse lisamist, helpdeskile kulutatud aeg näiteks halvasti disainitud kasutajaliidese puhul[3].

Lisaks on veel mõned silmapaistvad erinevused finantsmaailmaga - tehnilise võla puhul on ülimalt tõenäoline, et võla tagasimaksjateks ei ole need samad inimesed, kes on selle võla esialgu tekitanud. Reaalne on ka situatsioon, kus võlga võib tagasi maksma hakata ka hoopis teine ettevõte. See võib julgustada võtma järgmistes projektides rohkem tehnilist võlga. Sellist olukorda soodustab näiteks ettevõtte sisemised poliitikad, mis väärtustavad kiiret implementatsiooni, mis on pahatihti pikaajalise hallatavuse vastand. Erinevused majanduses käibel oleva “võla” mõistega on veel selles, et kogu laenu ei pea tõenäoliselt kunagi tervenisti tagasi maksma. See tähendab, et mõned süsteemi osad võivad elada koos tehnilise võlaga kogu oma elutsükli, ilma, et sellelt mingit intressi tasuma peaks. Samuti pole mõtet tagasi maksta võlga, kui sellelt tasutav “intress” ei ületa tarkvara elutsükli jooksul põhiosa väärtust.

Tuleb meeles pidada, et tehniline võlg ei ole seotud ainult programmi koodi või programmeerijatega. Täpselt samasugust mustrit võib täheldada ka teistel ametitel: näiteks kui puhkusele siirduv töötaja ei ole oma tööülesandeid lõpetanud või teistele delegeerinud ning puudub ülevaade, kui kaugele on nende ülesannetega jõutud - probleemide tekkides peavad kolleegid nägema ekstra vaeva et ennast toimuvaga kurssi viia ning tõenäoliselt tuleb neil pooleli olevad ülesanded ka lõpetada.[3]

Ettepanekud omada täielikult tehnilise võla vaba süsteemi pole autori hinnangul kuigi otstarbekad, kuna sellisel viisil tarkvara loomine on liigselt ressursinõudlik, ning

praeguse agiilsete arendusmeetodite juures - (iga iteratsiooni lõppedes võiks olla olemas töötav prototüüp uuest funktsionaalsusest) jääks toode ilmselt mõistlikul ajal turule viimata.

Sellise ressursi kasutamine, mida oleks vaja tehnilisest võlast täielikult loobumiseks ei ole tingimata tihedalt seotud projekti õnnestumisega [16]

2.2 Tehnilise võla liigid

Tarkvara arendusega seotud tehnilist võlga on siiani liigitatud 10 erinevasse kategooriasse, selline jaotis on saadud tehnilise võlaga seotud teadustööde kaardistamise uuringust, mille käigus kaardistati 1992-2013 aastatel ilmunud ning tehnilist võlga käsitlevate mõistete kasutust[2]:

1. Nõuetega seotud tehniline võlg
2. Arhitektuuri tehniline võlg
3. Tehniline võlg, mis on seotud disainiga
4. Lähtekoodi tehniline võlg
5. Testimisega seotud tehnilise võla käsitlus
6. Tarkvarapakettide ehitamisega (build) seotud tehniline võlg
7. Infrastruktuuri tehniline võlg
8. Versioneerimisega seotud tehniline võlg
9. Defektid, bugid
10. Üldine tehniline võlg

Konkurentsitult kõige rohkem on uuritud lähtekoodi ning testimisega seotud tehnilist võlga, peamiselt seetõttu, et selle tuvastamiseks on olemas lai valik erinevaid tööriistu

ning üldiselt on koodiga seotud tehniline võlg kõige arusaadavam. Ülejäänud derivaadid tehnilisest võlast on kas abstraktsema definitsiooniga (nõuetega seotud tehniline võlg) või ei oma tarkvaratootele otsest kvaliteediga seotud mõju. [2]

2.3 Tehnilise võla peamised tekkepõhjused

Tehnilise võla tekkepõhjuseid on erinevaid: see võib tekkida teadlikult näiteks ärilise otsusena, et toode peab valmima mingiks varem määratud tähtajaks. Etteantud tähtaja saavutamiseks implementeeritakse ühe lahendusena mõned funktsionaalsuse osad selliselt, et näiteks mõningad tarkvara arhitektuuriga seotud häid tavasid ignoreeritakses, vaid selleks et oleks võimalik tooda või funktsionaalsus kiiremini turule(toodangusse) saada. Järgnevalt toob autor ära mõnedes käsitlustes pakutavad potentsiaalsed tehnilise võla tekkepõhjused[4] Selline tekkepõhjuste jaotis on piisavalt üldine, et sobida kirjeldama praktiliselt igat tüüpi tehnilist võlga, selle hulgas ka meid huvitavat lähtekoodist tulenevat võlga:

Praktiline vajadus - Iseloomulik näiteks väga noortele ettevõtetele (Startupid jms alustavad ettevõtted). Näiteks esineb oht, et kui antud toode ei jõua kindlaks määratud ajal turule, tuleb kogu ettevõttel ilmselt töö lõpetada. Kuna tootest saadav kapital läheb koheselt kasutusse ning on vajalik element ettevõtte jätkusuutlikkuseks. Pragmatismist tulenevat tehnilist võlga eristab prioritiseerimisest tulenevast võlast selle lähtesuund, mis antud juhul on ettevõtte seest väljapoole - võlg tekib ettevõtte enda sisemistest otsustest ja protsessidest[4]. Sisemiste faktorite hulka kuuluvaks loetakse veel vananenud, ning kolmanda osapoole toodetud teeke. Selline tehniline võlg nõuab intressi näiteks juhul, kui tekib vajadus *legacy* teeke uuendada[27].

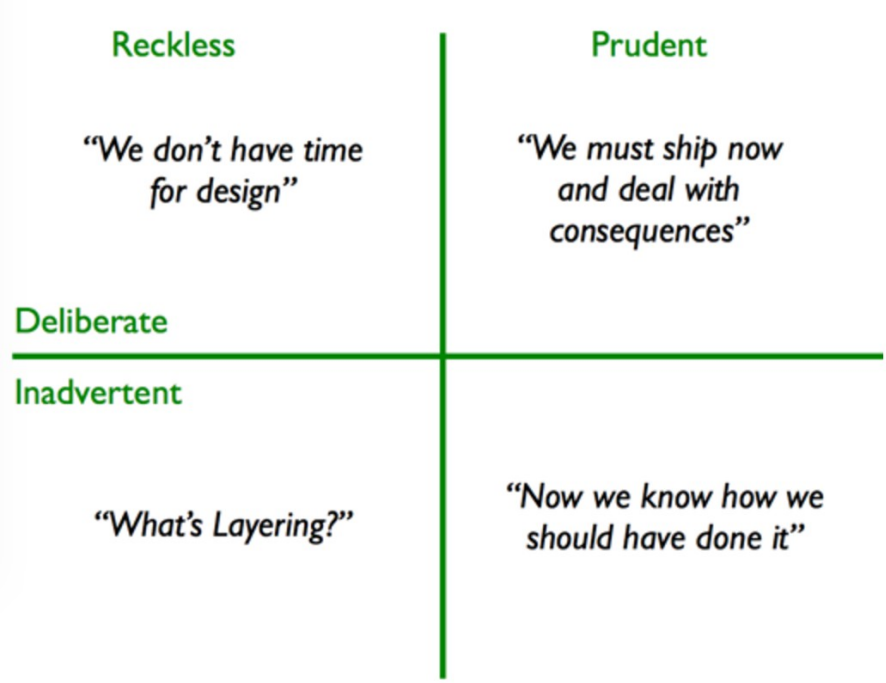
Prioritiseerimine - Üldine tendents tehnilise võla tekkel prioritiseerimise läbi on asjaolu, et tarkvara arendusega tegelevad meeskonnad otsustavad funktsionaalsuse implementeerimise kasuks, juhul kui teisel kaalukeelel on loodud üleüldine kvaliteet. Veel üks näide prioriteetide muutusest tekkiva tehnilise võla kohta on näiteks hilisemate halduskulude kirjutamine mõne teise projekti eelarvesse või lihtsalt erinevale reale raamatupidamises. Seega on siinkohal väga tähtis saada toode õigel ajal turule. See võib tõsta kliendis ootusi ka järgmistele arendusiteratsioonidele - kuna esialgne toode valmis suhteliselt kiiresti on loogiline järeldus kliendi poolelt, et ka järgnevad nõuded

täidetakse vähemalt sama kiiresti, mis omakorda seab teatud surve arendajatele ning nad tõenäoliselt tekitavad tehnilist võlga veelgi juurde.[4] Prioritiseerimise alla kuulub ka igasugune leppetrahvidega määratletud projekti üleandmise tähtaeg[27]

Suhtumine ning hoiak - autori hinnangul väga tähtis tehnilise võla tekkepõhjus. Kui tarkvara loovad inimesed on selle kvaliteedi suhtes ükskõiksed või kui valitseb mentaliteet, et töötavat asja ei ole mõtet "parandama" hakata - on tehniline võlg kiire kogunema. Samuti võib juhtuda, et mingis kindlas keeles väga osavad arendajad implementeerivad mõnes teises keeles (millest nad mingil põhjusel lugu ei pea) funktsionaalsust mitte nii kvaliteetselt. Tõenäoliselt on sellise tekkepõhjuse likvideerimine üks keerulisemaid, kuna tõenäoliselt tuleb välja vahetada töötaja koos oma suhtumiste ning hoiakutega. [4]

Teadmatus ning eksimused - Selle tekkepõhjuse võib jagada kaheks - eksimused, mis toodavad tehnilist võlga juurde. Näiteks vale disainimustri kasutamine, kuna lihtsalt ei tunta antud ülesande jaoks sobivamaid mustreid. Teisena teadmatus tehnilisest võlast üldiselt - ei teadvustata et mõne valiku puhul tuleb hiljem maksta "intressi", mis võib ajaliselt olla isegi kulukam, kui kohe õigesti käituda. See tekkepõhjus on eriti ohtlik, kui peaks juhtuma kokku terve meeskond selliseid arendajaid, kes ei teadvusta tehnilise võla olemasolu ning ei oska ka üksteist sellistel puhkudel aidata.[4] Mõned allikad[27] jagavad põhjusena teadmatuse veel 4 erinevasse alamkategooriasse: kogenematus, puudulikud teadmised valdkonnas, teadmatus (eelkõige tarkvara disaini osas) ning hoolimatus.

Martin Fowler on tekkepõhjusena välja pakkunud pisut lihtsama alajaotuse, sellist jaotist on hakatud kutsuma Fowleri kvadrandiks[17]. Selle lähenemise kohaselt jagatakse tehnilise võla tekkepõhjused nelja jaotisse:



Joonis 1. Fowleri kvadrant

Lühidalt jagatakse võla tekkepõhjus horisontaalselt tahtmatuks ning tahtlikuks ning vertikaalselt lohakaks ning ettevaatlikuks/hoolivaks.

2.4 Tehnilise võla haldamise puudumine

Antud tarkvaraettevõtte kontekstis tegeletakse tehnilise võla haldamisega kaudselt. Kogu kood, mis toodangusse läheb vaadatakse alati üle (code-review) vähemal ühe meeskonnakaaslase poolt. Samas on koodi ülevaatuste kvaliteet meeskonniti kõikuv, ning mõnikord võib meeskond tehnilise võla teadliku otsusena vastu võtta, kuid pärast seda otsust juhtub tihti, et tekitatud tehniline võlg ununeb kuni selle korrani, kui sama koodi uuesti muuta tuleb.

Üheks tehnilise võlaga seonduvatest probleemidest on puudulik haldus ning äriprotsessid kuidas peaks toimima tehnilise võla haldus ning riskide maandamine,

samuti prioriteetide puudumine tehnilise võla käitlemisel. Kogenenumad arendajad on enamasti võimelised mainima, milliseid tehnilisi järeleandmisi on parasjagu mõne komponendi juures tehtud. Võimalik, et ka märge selle kohta on tehtud lähtekoodi juurde koodi kommentaarina või isegi luuakse projekti *backlog*-i seda puudutav tööülesanne tulevikuks, kuid organisatsioonis tervikuna puudub üheselt defineeritud protsess, kuidas tehnilise võlaga käituda. Näiteks võib see näiteks sõltuvalt projektijuhi suvast jäädagi oma aega ootama. Halduse ning prioritiseerimise puudumisega kaasnevad suurenenud kulud tarkvara haldusfaasis. Töö autor peab vajalikuks mainida, et tehnilise võla haldamise puudumine antud organisatsioonis ei ole binaarselt negatiivne nähtus, kuna paljude projektidega ei tulegi pärast esimesi iteratsioone rohkem tegeleda. Probleem on pigem projektidega, mille raames toimub pidev arendus, ehk vajadus varem rajatud maja vundamendile ehitada uusi korruseid.

2.5 Tehnilise võla mõju tarkvara arendusele ning peamised riskid

Üheks suuremaks tehnilisest võlast tulenevaks riskiks on tehnilise võla olemasolu mitte teadvustamine projekti juhtimise tasandil, mis võib päädida ekstreemsematel juhtudel ka arendustegevuse seiskumisega või vajadusega mingeid suuremaid süsteemi osiseid nullist uuesti arendada. See võib tekkida näiteks juhul, kui esialgse süsteemi loojateks pole samad isikud/meeskonnad, kui need kes pärast esmast lansseerimist seda süsteemi haldama ning edasi arendama hakkavad.

Kui esmase implementatisooni käigus on projekti tarnetähtaegadest tulenevalt kiirustatud, kuid süsteem on edukalt lansseeritud, siis võib juhtuda, et hiljem süsteemi haldama hakkavad arendajad avastavad, et käesoleva süsteemi muutmine osutub äärmiselt keeruliseks.

Mõningad käsitlused jagavadki arendajad kahte gruppi: esialgse funktsionaalsuse loojad ning haldajad, kes hakkavad tarkvaraga tegelema selle lansseerimisjärgses etapis. Tihti tunnustatakse insenere, kes loovad esialgse töötava prototüübi suhteliselt kiire ajaga, kuid kogu see näiline edu tuleb enamasti nende arvelt kes hiljem selle projekti haldusega tegelema peavad. Võimalik isegi, et esialgse prototüübi loojatel puudub üldse kogemus tarkvara haldamises. See on võimalik selle tõttu, et projekti algfaasis pole tehniline võlg eriti nähtav kellelegi teisele peale arendajate enda. Üheks taolise riski

maandamise võimalikuks on kaasata projekti esimesse faasi arendajad, kellel on olemas kogemused tarkvara haldamises.[3].

Mõningad allikad [4] jagavad tehnilise võlaga seotud riskid selgetesse kategooriatesse:

Psühholoogiline efekt arendajatele - Lisaks palju käsitletud majanduslikule aspektile on tehnilise võla kogunemisel ka negatiivne psühholoogiline mõju meeskonnale. Tihti seetõttu, et olemasoleva tarkvara ümbertöötamine (refactoring) pole pahatihti nii põnev ülesanne, kui uue funktsionaalsuse loomine või muud arendusega seotud väljakutsed. Kõigile tarkvara kvaliteeti tõsiselt võtvate arendajate moraalile mõjub tehnilise võla kogunemine (isegi lühiajaliselt) demoraliseerivalt. [4]

Produktiivsuse langus - tehnilise võla põhiline idee on suurendada produktiivsust lühiajaliste eesmärkide saavutamiseks. Siin on kohane tuua võrdluseks finantsmaailma vastavad terminid - kui on võetud kõrge intressiga laen, tuleb suur osa kapitalist kulutada selle intressi tasumiseks ning üldine ostujõud väheneb. Ostujõu vähenemist saab võrrelda uue funktsionaalsuse lisamisega - enamus olemasolevast ressursist kulutatakse varem tehtu ümberkirjutamisele, enne kui on võimalik mingit uut funktsionaalsust lisada. [4]

Mõju kvaliteedile - üldine tehnilise võla mõju kvaliteedile on negatiivne. Esiteks võib tehniline võlg süsteemis defekte juurde tekitada, ning teiseks aitab tehnilise võla kogunemine kaasa vigade avastamise keerukusele - näiteks automaattestid (unitTest), mis aja kokkuhoiu tõttu on jäänud kirjutamata. Enim kannatavate omaduste hulka kuuluvad tarkvara skaleeruvus, hallatavus, adaptiivsus ning kasutatavus.

Mõju kogu projektile (projekti risk) - Tehniline võlg suurendab ebakindlust ükskõik missuguste muudatuste sisseviimisel projekti. Mida kõrgem on tehnilise võla tase, seda raskem on ette ennustada töö hulka mis tuleb uue funktsionaalsuse implementeerimiseks teha. Samas tehnilise võla kogunemine võib lühiajaliselt vähendada projekti ebaõnnestumise riski, siin võib paralleele tuua praktilise vajadusega seotud tekkepõhjusega - kui ei õnnestu toote esimest iteratsiooni teatud ajal turule viia, on projekt läbi kukkunud.[4,10]

3 Teoreetiline ülevaade tehnilise võla prioritiseerimisest

3.1 Tehnilise võla hetkeolukorra kaardistamise vajadus

Hetkeolukorra kaardistamine on vajalik etapp üldises tehnilise võla haldamise protsessis. See aitab võtta vastu otsuseid uue funktsionaalsuse planeerimise suhtes ning aitab vastata küsimusele - kas enne uue funktsionaalsuse loomine on käesoleva koodibaasi juures üldse otstarbekas. Kaardistatud olukord aitab vastu võtta taktikalisi otsuseid - tõenäoliselt ei ole mõtet luua funktsioone juurde, kui selgub et antud komponendile kulutatavast ajast kuluks suure tõenäosusega 90% varem akumuleerunud tehnilisele võlale. Enamik allikad (näiteks [4,5,6,7]) viitavad käesoleva olukorra kaardistamisele, kui olulisele komponendile tehnilise võla üldise haldamise juures. Hetkeolukorra kaardistamine annab ka suurepärase võimaluse projektide võrdluseks - kui oleme võimelised määrama tehnilise võla hetkeseisu mingi väärtusega, on need väärtused projektiüleselt võrreldavad.

3.2 Teoreetiline ülevaade SQALE meetodist

SQALE meetodit mainiti autorile teadaolevalt esimest korda 2009 aastal [20]. SQALE meetod on lühendatult tarkvara elutsüklite ootustel põhinev tarkvara kvaliteedi hindamise metoodika. SQALE meetod lubab pakkuda täpse mudeli, millega hinnata tehnilise võla suurust ning võimaldab seada konkreetseid prioriteete kõnealuse projekti parendamisel. SQLAE meetod üritab hinnata nõudeid, mida esitatakse tarkvarale kogu tarkvara elutsükli käigus

3.2.1 Kvaliteedimudel ning indeksid

SQALE meetodi rakendamiseks on esmalt vajalik koostada nimekiri mittefunktsionaalsetest nõuetest, mis määravad ära korrektse koodi. Sellist nimekirja kutsutakse kvaliteedimudeliks ning selle abil on võimalik koodist tulenevale tehnilisele

võlale hinnanguid anda. Kvaliteedimudel on miski, millel arendustegevusega seotud meeskondadel on võimalik oma töös juhinduda, see peab koosnema selgetest ning kontrollitavatest nõuetest.

Näited mittefunktsionaalsetest nõuetest:

- muutujatele ei omistata väärtust tsükli sees
- funktsiooni tsüklomaatiline keerukus ei ületa mingit kindlat väätust
- tarkvara pakettide vahel pole ringsõltuvust jms

Mittefunktsionaalse nõuete mittetäitmise parandamine (leevendamine). Järgnevalt tuleb iga mittefunktsionaalse nõudega seostada tegevus, mis on vajalik selle "rikkumise" leevendamiseks. Tehnilise võla suurust saab hakata hindama, kui korrutada leitud täitmata mittefunktsionaalse nõude arv leevendamiseks ette nähtud ajaga, sel moel saab teada absoluutse aja, mis kulub nende nõuete täitmiseks.

Näiteks ringsõltuvuse eemaldamine kahe tarkvarapaketi vahel omab leevendamiseks ettenähtud aega 40 minutit iga sõltuvuse kohta. Kui meie projektis leidub 6 sellist sõltuvust, võib hinnata selle konkreetse nõude leevendamisele kuluva aja 4 tunnile.[8,9]

Kvaliteedi indeks (SQI)

SQALE mudeli täpsus on otseses sõltuvuses sellest, kui täpselt on defineeritud kvaliteedimudelile mittevastavuste parandamisele kuluv aeg. Järgnevalt jooksutatakse projekti kood läbi staatilise analüüsi tööriistade ning seostatakse leitavate puuduste arv nendega seotud leevendavate tegevuste ajaga. Tehnilise võla hulga määrab ära kõikide nende tegevuste summa ning seda summat nimetatakse SQALE kvaliteedi indeksiks (SQI). Seda tulemust on näiteks võimalik jagada projekti koodi hulgaga, saades tulemuseks tehnilise võla tiheduse, mis võib omakorda vajalik olla erinevate meeskondade või organisatsioonide poolt arendatavate projektide omavahelises võrdluses. [8,9]

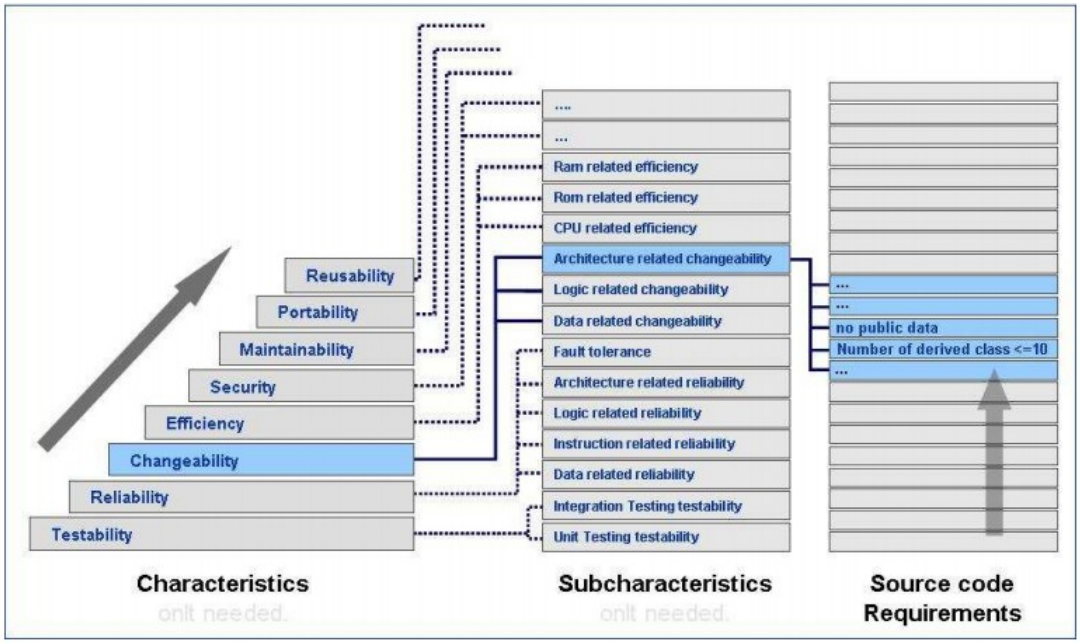
SQALE ärilise mõju indeks

Ärilise mõju indeks kujutab endast kõikide nõuete rikkumiste arvväärtust kokku kogutuna ning jagatuna nende nõuetega seotud leevendamiseks kuluva ajaga. Ärilise mõju indeks näitab, millised on äriliselt olulised rikkumised, mis koodi analüüsi käigus välja on leitud. Ärilise mõju indeks annab infot selle kohta, millised rikkumised on prioriteetsemad kui teised.[8] Teisisõnu võimaldab see meil selekteerida minimaalse ajakuluga saavutatava maksimaalse tulemuse tehnilise võla vähendamisel. Nõuete rikkumise arvväärtused on kirjeldatud järgmises peatükis.

3.3 Tehnilise võla analüüs SQALE mudelis

SQALE mudel pakub lisaks kvaliteedi indeksile ka teisi näitajaid ning suhtarve, et tehnilist võlga oleks võimalik analüüsida ning pakub tehnilist loogikat otsuste vastuvõtmiseks. SQALE mudel tugineb kaheksale põhilisele tarkvara kvaliteedi tunnusele, nendeks on:

- Taaskasutatavus
- Porditavus
- Hallatavus
- Turvalisus
- Efektiivsus (jõudlus)
- Muudetavus
- Töökindlus
- Testitavus



Joonis 2. Tarkvara karakteristikud SQALE meetodis[20]

Need tunnused on oma prioriteetidelt üksteist katvad - kõige tähtsam nendest on testitavus, kõik nimekirjas üleval pool asuvad tunnused toetuvad allpool asuvatele. Näiteks pole eriti kerge teha mitte testitavat komponenti töökindlaks jne.

Eelpool kirjeldatud mittefunktsionaalsed nõuded tuleb seostada nende 8 tunnusega, kui peaks juhtuma et mõni nõue on seostatav mitme erineva tunnusega, käsib metoodika seda seostada madalaima võimaliku tunnusega.

Seejärel on võimalik jagada esialgne tehnilise võla hulk - SQALE kvaliteedi indeks nende tunnuste vahel, mis omakorda annab analüüsijale jaotuse, milliste tunnuste alla on kogunenud enim tehnilist võlga. Sellise jaotuse pealt on võimallik juba mõningaid otsuseid vastu võtta - näiteks porditavusega seotud tehnilise võla hulka on võimalik ignoreerida, kui ettevõttel pole lähiajal plaani liikuda oma funktsionaalsusega mõnele teisele tehnilisele platvormile.

Kuna varasemalt järeldasime, et kogu tehnilise võla tagasimaksmine ei pruugi paljudel juhtudel olla otstarbekas, pakub SQALE meetod välja vastupidise funktsiooni nõuete mitte-leevendamise suhtes[8,12].

Järgnevalt jagatakse nõuded nende rikkumiste tõsiduse järgi kategooriatesse. Näiteks identsete koodilõikude (suure tõenäosusega kopeeritud) kategoriseeritakse kõrge prioriteediga nõuete hulka võrreldes näiteks koodi treppimata jätmisega.

Igale nõudele lisatakse ka suhteline arvväärtus, mis kujutab selle rikkumise suhtelist tähtsust. Otsesed väärtused siinkohal rolli ei mängi, tähtis on nende väärtuste omavaheline suhe. Näiteks muutujate nimetuse heade tavade vastu eksimine võib olla tuhandeid kordi väiksema väärtusega, kui nulliga jagamine, mis tekitab otsese veaohu.

Näitena võib jagada nõuete rikkumised 5 kategooriasse, omistades neile ka suhtelise arvväärtuse[8]:

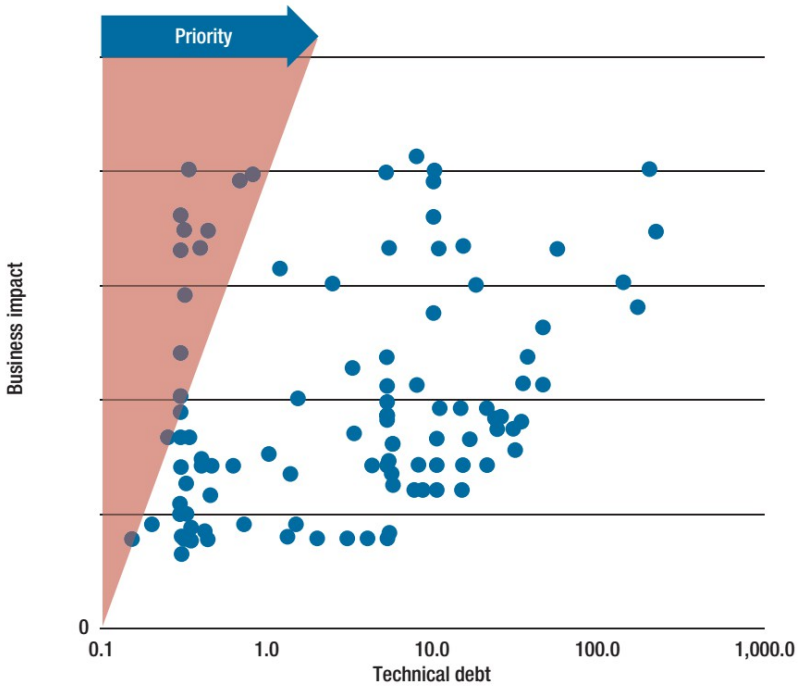
- Kriitiline (5000)
- Kõrge (250)
- Keskmine (50)
- Madal (15)
- Minimaalne (2)

Siinkohal peab mainima, et erinevad allikad pakuvad rikkumiste kategooriatele erinevaid arvväärtusi, lisaks sellele on nende arvväärtuste suhe erinev. SQALE juhend [9] pakub välja väärtused vastavalt (20,5,1,0.1,0.01)

3.3.1 SQALE meetodi teoreetiline rakendamine

Et piiratud ressurssidega probleeme prioritiseerida, tuleks tegeleda esmajärjekorras nende puudustega, mis on ärilises mõttes kõige kriitilisemad. Kui SQALE kvaliteediindeks näitas, et meie hüpoteetilisel projektil oleks vaja 20 päeva, et likvideerida kogu tehniline võlg, kuid meil on raha või aega ainult 5 päeva ulatuses, tuleb leida need puudused, mis on ärilisest seisukohast kõige väärtuslikumad. Selleks

paigutatakse puudused graafikule nii, et kvaliteediindeks satub X-teljele ning ärilise mõju index Y-teljele.



Joonis 3. Tehnilise võla prioritiseerimine[8]

Prioriteet selgub, kui lugeda graafikut diagonaalis vasakust ülemisest nurgast parema alumise nurgani. Sisuliselt seisneb prioritiseerimine kvaliteedi indeksi ning ärilise mõju indeksi suhte leidmises, kuna meie huvides on leida probleemid, mis omavad maksimaalset ärilist mõju.[9]

3.4 Ülevaade kaardistamiseks valitud tööriistast SonarQube ning võimalikud alternatiivid

SonarQube (varasema nimetusega Sonar) on avatud lähtekoodiga tarkvara, mis on mõeldud tarkvara kvaliteedist hinnangu saamiseks. SonarQube (edaspidi SQ) on

kompleksne tööriist, mis kasutab oma tööks sisendina mitmete erinevate koodi kvaliteedi eri aspekte hindavaid tööriistu, nagu näiteks Checkstyle, PMD jmt. SQ-s koodi analüüsimiseks kasutatud meetodid pole antud tööriista spetsiifilised [11]. SQ on võimalik integreerida hõlpsasti erinevate *Continuous Integration* lahendustega, et saada pidevalt tagasisidet koodi kvaliteedi kohta. Kuna SQ salvestab koodis leiduvad probleemid andmebaasi ning seob need tarkvara vastava versiooniga, siis on võimalik probleemide monitoorimine ajas ning teavitada, kui seisukord on muutunud halvemaks võrreldes eelmiste tulemustega.

SQ tööpõhimõte seisneb lähtekoodi staatilises ning dünaamilises analüüsimises, analüüsi käigus kontrollitakse koodi vastavust SQ andmebaasis defineeritud reeglitele, reeglid ise on koostatud nii, et need oleks vastavuses tarkvara arenduse parimate praktikatega – näiteks tunneb SQ ära koodi duplitseerimisega seotud probleeme, võimalikke vigu (bugs) ning liialt keerukaid konstruktsioone. Reeglite baasi on võimalik laiendada, kasutades kolmanda osapoolte pistikprogramme. Iga reegli juurde kuulub tema prioriteet skaalal 1-5, minimaalsest kuni kriitiliseni. Lisaks kuulub iga reegel kindlasse kategooriasse – vastavalt kas üldised koodi standardid, jõudlusega seotud probleemid, turvariskid, disainivead jms.

Kaardistamiseks alternatiivseid lahendusi otsides jõudis töö autor järeldusteni, mida kinnitab magistritöö [24], milles uuritakse koodi kvaliteeti enne ning pärast koodi kvaliteediga seotud nõuete meeldetuletamist arendajate seas. SQ plussideks on kõikide reeglite klassifitseerimine ning tulemuste salvestamine homogeenselt ühises andmebaasis, millega on võimalik tänu veebiliidesele hiljem mitmesuguseid andmetöötlus operatsioone läbi viia. See võimalus on pakkunud paljudele uurijatele võimaluse analüüsida SQ poolt objektiivselt tuvastatud reeglitele mittevastavusi erinevatelt tahkudelt – näiteks tehnilise võla suhe ekspertarvamustega [7]. Uurimustöö mis võrdleb nelja lähenemist tehnilise võla tuvastamisel eristab[12] SQ ülejäänud koodi staatilise ning dünaamilise analüüsi tööriistadest just konkreetsetele probleemsetele kohtadele viitamine. Tööriistad enne SQ võimaldasid tuvastada probleeme üldiselt, mitte seostades neid konkreetsete ridadega lähtekoodis.

SQ negatiivseteks omadusteks on failipõhine töörežiim, mis vähendab duplitseeritud koodiga seotud probleemide raporteerimist [24], kuna pole võimeline tuvastama

dubleeritud koodi üle kogu projekti, vaid ainult käesoleva faili ulatuses. Eisenberg[11] on seadnud kahtluse alla tehnilise võla ajalise hindamise SQ poolt, kuid möönab et ülejäänud andmed ning meetrika on kasutatav. Ka käesolevas töös ei oma ajaline faktor projekti kohta muud tähtsust, kui vaid võrdlusmoment kahe erineva projekti vahel. Negatiivseks võib veel lugeda aspekti, et SQ on originaalis JAVA programmeerimiskeelele pühendatud tööriist ning tõenäoliselt parima dünaamilise ning staatilise analüüsi tulemused on võimalikud just JAVA keelt kasutades [31]. Siiski on võimalik erinevate pistikprogrammide abil laiendada funktsionaalsust ka teistele keeltele.

SonarQube on antud töös eelistatud tema otsestele konkurentidele Metrixwarele ning Squoringule peamiselt seetõttu, et tegu on vabavaralise tarkvaraga ning SQ toetab PHP programmeerimiskeelt.

4 Tehnilise võla kaardistamise protsess ja tulemused

4.1 Analüüsimiseks kasutatud tarkvara valik

Antud töös läbiviidud analüüs tugineb kahele projektile, mille puhul pikaajaline tarkvara hallatavus on eriti kriitilise tähtsusega. Tegemist on projektidega, mis on ajaliselt kestnud üle 4 aasta ning milles lisaks uue funktsionaalsuse loomisele on suur osa haldamisel - selleks on nii operatiivne haldamine erinevatele koodiga seotud kriitiliste tõrgete tekkimisel igapäevaselt, kui ka pikaajaline haldus.

Mõlemad projektid kujutavad ennast e-kommerts keskkonda, mis on keskendunud ehitusmaterjalide ning erinevate kodu- ning aiakaupade müügile. Rakenduste back-end on kirjutatud PHP-s ning kasutavad tööks Magento Enterprise Edition raamistikku.

Kuigi kasutatud raamistik võimaldab ka ilma mingi kolmanda osapoole lisata veebipoodi hallata on valitud projektid väga palju enamat kui lihtsad veebipoed, olles integreeritud kümnete erinevate süsteemidega üle SOAP ja REST liideste abil ning omades kümneid suuremaid ning väiksemaid mugandusi vastavalt klientide vajadustele. Projektide sarnasus seisneb selles, et tegu on sama kliendiga, kellele on eri riikide jaoks loodud funktsionaalsus ning integratsioonid vastavalt vajadusele. Projektidega tegelev meeskond on alates projekti algusest 100% ulatuses välja vahetunud selliselt, et esialgselt projektiga tegelenud meeskonnast pole käesolevaks hetkeks enam keegi projektide juures tegev.

Lihtsustamise huvides nimetab autor edaspidi projekte Projekt A ning Projekt B, kuna antud töö käsitluses ei oma konkreetsed projektid ning kliendid suuremat tähtsust.

Projekt A lühikirjeldus

- Koosneb 140 moodulist
- Ca 2.1 miljonit rida koodi (back-end, äriloogika)

- 10-liikmeline meeskond
- 7 arendajat

Projekt B lühikirjeldus

- Koosneb 124 moodulist
- Ca 1.7 miljonit rida koodi
- 7-liikmeline meeskond
- 5 arendajat

Mõlemad projektid on võrdlemisi pikaajalised, vastavalt Projekt A - 6 aastat ning Projekt B 5 aastat, mis teeb nendest autori hinnangul tehnilise võla käsitluse mõistes huvitavad uurimisobjektid.

Analüüsiks on tehtud valik teatud projekti osadest - kuna tegemist on Magento raamistikule toetuvate projektidega, pole töö autori hinnangul põhjendatud Magento baas-lähtekoodi analüüs, küll aga võiks see anda ainet mõnele järgnevale uurimustööle.

Analüüsimiseks on tehtud valik moodulitest, mida on viimase kahe aasta jooksul enim kordi muudetud. Idee tuleneb mõnes käsitluses välja pakutud ideest, et tehnilise võla käsitluses on lisaks võla enda hulgale ning ärilisele väärtusele oluline ka tõenäosus, et vastavat komponenti tuleb üldse tulevikus muuta[18].

Mõlemast projektist on valitud 15 moodulit, järjestades need kahanevalt vastavalt viimase kahe aasta jooksul tehtud muutustele. Selleks on kasutatud Mercurial versioonihaldussüsteemi sisseehitatud halduskäsku log.[13]

Projekt A:

Tabel 1: Projekt A viimati muudetud moodulid

Moodul	Muutusi viimase 2a jooksul	Koodi kokku(read)
ProjectA_theme	3482	87998
ProjectA_cms_common	2157	104926
ProjectA_checkout_commo n	991	10435
ProjectA_halle	672	11829
ProjectA_checkout	650	5996
ProjectA_theme_seamless	483	27580
ProjectA_se	469	11191
ProjectAB_klarna	450	29625
ProjectAB_common	440	9526
ProjectAB_multioptionfilter	373	5114
ProjectA_seshipping	332	6636
ProjectAB_b2b	293	12407
ProjectA_collectatstore	210	2651
ProjectA_bookemployee	195	6033
ProjectAB_voucher	167	1748

Projekt B:

Tabel 2: Projekt B viimati muudetud moodulid

Moodul	Muutusi viimase 2a jooksul	Koodi kokku(read)
ProjectB_theme_dk_ee	1843	57005
ProjectB_checkout_commo n	996	10435
ProjectB_common_dk	754	15115
ProjectB_checkout	649	5561
ProjectB_bookanemployee	625	18712
ProjectAB_klarna	506	29625
ProjectAB_common	456	9523
ProjectAB_multioptionfilter	380	4399
ProjectAB_b2b	298	11729
ProjectB_collectatstore	205	2515
ProjectAB_storelocator	198	4555
ProjectAB_voucher	167	1748
ProjectB_menu	148	1064
ProjectB_rinvoice	110	24644
ProjectB_dkshipping	110	2176

Tehnilise võla suurus väljendab ennekõike muudatuste tegemise hinda - kui muudatusi pole vaja teha, pole vaja ka selle eest mingit hinda maksta. Seetõttu pole oluline, kui suur osa tehnilist võlga tuvastatakse nendes süsteemi osades mida aktiivselt ei

muudeta[18]. Mitmed allikad [25,6,10] nõustuvad, et nendel süsteemi osadel on suurem tõenäosus muutuda ka tulevikus, mida minevikus on aktiivselt muudetud.

4.2 Lähtekoodi analüüs tööriistaga SonarQube

SonarQube on oma olemuselt vabavaraline tööriist, mis on kõikidele huvilistele vabalt allalaadimiseks saadaval. Käesolevaks hetkeks (27.03.2017) on uusim tarkvara versioon versiooninumbriga 6.3. Antud töös on kasutatud pisut vanemat versiooni 5.6.6 Long-Term Support (LTS), kuna LTS versioonile on saadaval rohkem lisafunktsionaalsust pakkuvaid pistikprogramme. Lisaks baastarkvarale on lisatud pistikprogrammid:

- SonarPHP - PHP-s kirjutatud lähtekoodi analüüsimiseks
- Sonargraph - lisab funktsionaalsust erinevate graafikute joonistamise näol
- Sonargraph Integration

SonarQube koodi analüüsiv reeglibaas sisaldab endast 127 reeglit ehk mittefunktsionaalset nõuet, millest igale on omistatud selle nõude rikkumise leevendamisega kaasnev aeg. Reeglid on kategoriseeritud ning enamik omavad viiteid oma algupära kohta. Leidub viited OWASP-ile, PSR-2-le, CWE-le jmt. Lisaks on määratud igale mittefunktsionaalsele nõudele selle nõude rikkumisel prioriteet.

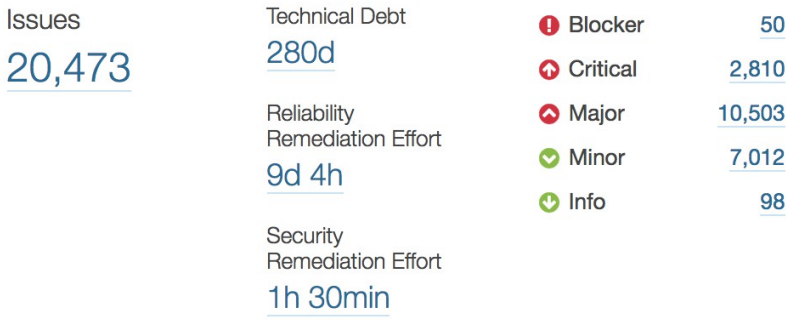
Kogu SonarQube poolt pakutav reeglibaas käidi esmalt läbi, et korrigeerida nõuete prioriteeti.

Selliste tingimuste prioriteete, mida autori hinnangul on võimalik automaatselt parandada/refaktoorida, kasutades arenduskeskkonna (IDE) sisseehitatud vahendeid vähendati 1-2 palli võrra. Reeglite prioriteeti, mis olid seotud testitavuse või üldise keerukusega tõsteti vastavalt. Kõrgeima prioriteediga (5 - blocker) väärtus omistati reeglile, mis tuvastas tsüklomaatilise keerukuse tõusmise üle lubatud taseme.

4.2.1 Projekti A analüüs (1 iteratsioon)

Esimeses iteratsioonis, (mis toimus enne reeglibaasi korrigeerimist) leiti projekti juures 20 473 probleemi ning tehnilise võla hulk - SQI võrdus 280 päevaga. Mis tähendab, et

ühel inimesel kuluks tehnilise võla täielikuks leevendamiseks projektis üle ühe kalendriaasta (arvestades puhkepäevi ning riiklike pühasid). SQI kogusumma näitab autori hinnangul tehnilise võla prioritiseerimise olulisust. Kuna kogu võla läbitöötamine ei ole antud projekti juures praktiliselt võimalik ega täida oma otstarvet [8,12].

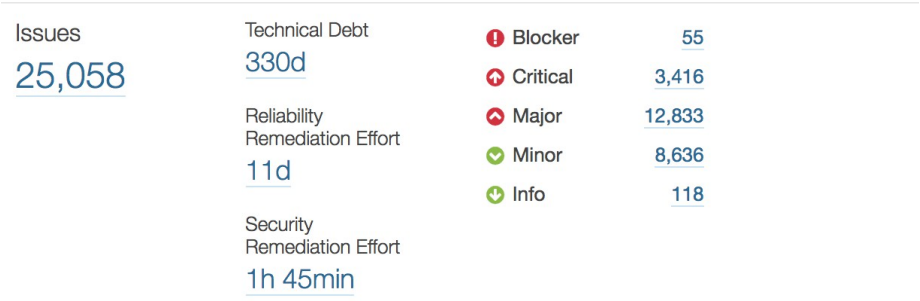


Joonis 4. Projekti A tehnilise võla hulk (iteratsioon 1)

Lisaks selgub analüüsist, et tehnilise võla suhe on 4.9%, mis tähendab kogu tehnilise võla suhet olukorraga, kui peaks kogu analüüsitud koodi nullist uuesti kirjutama[15]. Väärtus alla 5% pälvib projekti hallatavuse väärtuseks hinde “A”.

4.2.2 Projekti B analüüs (1 iteratsioon)

Teist projekti analüüsides leiti 25 058 mittefunktsionaalse nõude mittetäitmist. Teine projekt sai kogu tehnilise võla leevendamise ajaliseks hinnanguks 330 tööpäeva. Tulemus on pisut ootamatu, kuna Projekt B on mahult väiksem kui A. Siinkohal peab autor vajalikuks ära märkida SQI väärtust erinevate projektide võrdluses.



Joonis 5. Projekti B tehnilise võla hulk (iteratsioon 1)

Tehnilise võla suhe projektis B omab väärtust 5.3%, mis annab tulemuseks B". Projekti B üldiseks keerukuseks on 22 392.

4.2.3 Projektide analüüsimisega seotud probleemid ning kitsaskohad

Esialgsest reeglibaasist on autor katsetamise käigus deaktiveerinud 13 reeglit, kuna sisuliselt esines antud nõuete mittetäitmisi igas projekti failis ning nõuded ei olnud autori arvates põhjendatud, kuna olid vastuolus ettevõttes kehtivate koodistandartitega.

Siinkohal peab töö autor vajalikuks tuua ära nimekirja reeglitest, mis pärast esimest iteratsiooni tulemusi analüüsides deaktiveeriti.

1. *Algavad loogelised sulud peavad algama uuelt realt*. SQ kehtib see nõue nii meetodite, kui ka loogiliste operaatorite puhul. Ettevõtte koodistandard näeb aga loetavuse huvides ette meetodite puhul loogelist sulgu uuel real ning loogiliste operaatorite puhul samal real.

2. *Algavad loogelised sulud peavad algama sama rea lõpust*. Antud reegel on vastupidine eelmisele, kuid antud projektide puhul mittesobiv. Sarnaselt reegli 1 põhjendusele on see vastuolus ettevõttes kehtivatele koodi kirjutamise standarditele.

3. *Klassinimed peavad vastama regulaaravalisele ^[A-Z][a-zA-Z0-9]*$* ehk peaksid PSR-2 soovitusele vastavalt koosnema vaid suurtest- ning

väiketähtedest, kuid kuna Magento 1 raamistik on valminud enne selliseid kokkuleppeid, ei peeta seal antud nõudest kinni, ning klassinimed on ühtlasi ka klasse sisaldavate failide automaatse laadimise hõlbustamiseks. Samuti on muutujate- ning klasside nimetuste osas erinevusi, Magento raamistik, mis omakorda toetub Zend raamistikule, kasutab muutujate ning funktsioonide nimetustes Zend raamistiku koodistandardeid [14].

4. *Klassi muutujate nimed peavad vastama regulaaravalisele ^[A-Z][a-zA-Z0-9]*$* deaktiveerimise põhjendus on sarnane reeglile 3.

5. *Süsteemi failinimed peavad vastama avaldisele [a-z][A-Za-z0-9]+.php* Antud reegel on vastuolus Magento 1 poolt rakendatavale koodistandardile, kuigi nõude fookus on tegelikult piirangul kasutada vaid alamosa ASCII märgistikust[32], esitab see lisaks ka nõude failinime algusele väikse tähega, mis antud raamistikku kasutades on võimatu.

6. *Failide lõpus peab olema üks tühi rida* PSR-2 [33] soovitusel peaks olema iga faili lõpus üks tühi rida. Antud standardi soovitust ei ole ettevõttes varem jälgitud on see reegel deaktiveeritud.

7. *Funktsioonide nimed peavad vastama regulaaravalisele ^[a-z][_a-zA-Z0-9]*$* Sarnaselt reeglitele 3 ja 4 on siinkohal erinevused Zend raamistiku koodistandarditega.

8. *Funktsioone ning muutujaid ei tohiks defineerida väljaspool klasse*. Kuna projekti hinnatava koodi hulgas oli ka kujundusmalle (template), siis deaktiveeris autor reegli, mis keelas väljaspool klasse funktsioonide ning muutujate deklareerimise, antud reegli võib ka liigitada false-positive hulka, kuna tegelikult käib mallide realiseerimine rakenduse klasside sees ning uued kohalikud muutujad lihtsustavad nendes mallides andmete taaskasutust ning üldist arusaadavust.

9. *Vähemalt 65% koodi ridadest peab olema kaetud ühiktestidega*. Kuna tegemist on *legacy* projektidega, millel valdavas enamuses puuduvad ühiktestid sootuks, oleks antud reegli all rikkumisena ära toodud enamus koodist.

10. *Iga koodi haru peab olema kaetud ühiktestidega vähemalt 65% ulatuses*. Sisuliselt sama nõue(9), SQ võimaldab mõningate versioonihaldus tarkvarade kasutamise korral ka projekti erinevaid harusid analüüsida.

11. *Kommentaaride tihedus koodis peab olema vähemalt 25%.* Antud reegel läheb vastuollu tänapäevaste koodikirjutamise praktikatega. Siinkohal ühtib autor Robert C Martini arvamusega [34], et kood peaks olema ise-dokumenteeruv nii suures ulatuses kui võimalik ning kommentaare tuleks kasutada säästavalt, vaid seal, kus see on ilmtingimata vajalik. Seetõttu ei saa nõuda kvoodiga kommenteerimise tihedust.

12. *Vigane või puuduv kopeerimisõiguste deklaratsioon faili päises*. Kuna töö eesmärkide hulka kuulub lähtekoodiga seotud tehnilise võla prioritiseerimine, ei kuulu õiguslikud küsimused otseselt meid huvitava tulemi alla.

13. *Lähtekood ei tohiks sisaldada kasutajanimesid või paroole*. Kuigi reegel on igati põhjendatud turvalisuse seisukohast [35], on selle implementatsioon SQ kontekstis puudulik - näiteks mõne muutuja või konstandi väärtusest leiti ükskõik missugusel kujul sõna “password”, liigitati see OWASP-ile viidates koheselt turvariski alla. Põhjendusega, et paroole ei tohiks muutujate väärtusteks omistada, kuigi reaalselt oli tegu viidetega süsteemi konfiguratsioonile, mille pealt oli võimalik andmebaasist parooli pärida.

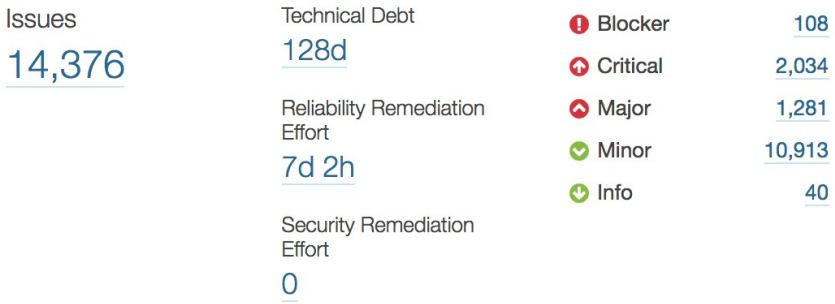
Seega on autori hinnangul käesolevate reeglite deaktiveerimisega ära hoitud *false-positive* juhtumeid ning tehnilise võla hindamise kogu protsesist on eemaldatud liigset müra. Lisaks oli autori hinnangul mõnede reeglite prioriteet seatud vaikimisi liiga kõrgeks, see puudutas eelkõige kõikvõimalikke ühe sõne kordusi, mida oleks võinud defineerida kui konstanti. Läbi käidud kõik reeglid ning vajadusel korrigeeritud reeglite prioriteete (vt Lisa 2).

Lisaks eelpoolmainitud reeglite hindamisele ning võimalikule deaktiveerimisele on üldiselt hinnangu andmisest välja jäetud kõikvõimalikud ühik-, funktsionaalsustestid ning testimisega seotud raamistikud, mis kuulusid moodulite algsesse koosseisu.

Eraldi väärib äramainimist, et SQ polnud võimeline analüüsima disainimalle, milles oli kasutatud underscore js javascripti teegi tage(<%,%>), kuna need kutsusid esile veateate parsimisel, sest sarnased tage oli võimalik kasutada ka PHP keeles enne versiooni 7.

4.2.4 Projekti A analüüs (2 iteratsioon)

Pärast reeglibaasi korrigeerimist tehtud analüüsi käigus leiti projekti A juures 14376 probleemi, ehk kolmandiku võrra vähem. SQI langes 128 päevale, mis on endiselt märkimisväärne kogus.



Joonis 6. Projekti A tehnilise võla hulk (iteratsioon 2)

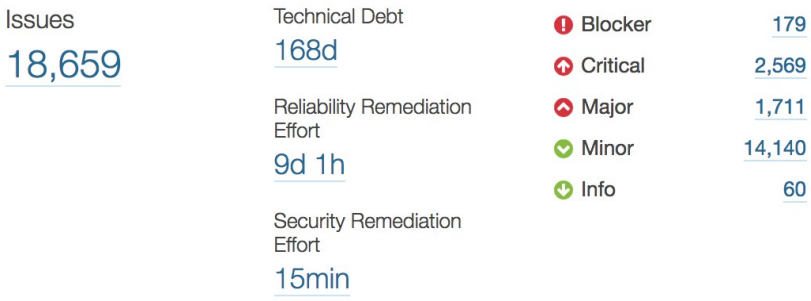
Muutunud on ka vigade jaotus prioriteetide järgi (vt joonis). Kõige kriitilisemate (5 - Blocker) vigade arv on suurenenud, nagu ka prioriteediga 2 (Minor) vigade arv. Mõlemad muutused tulenevad autori hinnangul taseme 3 (Major) rikkumiste ümberhindamisest.

Suure tõenäosusega on tulemuseks ka ärilise mõju indeksi jaotuse muutumine.

Tehnilise võla suhe on langenud 2.3%-ni, duplitseeritud koodi hulka kogu koodi mahust hindab SonarQube 6.1%. Koodi keerukuse suhtarvuks on 18810.

4.2.5 Projekti B analüüs (2 iteratsioon)

Teise iteratsiooni tulemusena leiti projektist B 18 659 probleemi. Tehnilise võla kogumäära (SQI) väärtuseks on 168 päeva. Leitud vigade prioritiseerimises leiab aset sarnane liikumine, mida oli märgata ka projekti A puhul: pisut on suurenenud vead prioriteetidega 5 ja 2 ning 3(Major) osakaal on tunduvalt vähenenud (vt joonis).



Joonis 7. Projekti B tehnilise võla hulk (iteratsioon 2)

Tehnilise võla suhe on projektis B langenud 2.8%-ni. 7.6% kogu projekti koodist on duplitseeritud ning kogu projekti keerukuse suhtarv on 21513.

4.2.6 Järeldused analüüsidest

Leitud suhtarvud on baasandmed andmebaasi salvestatud rikkumistele ning rikkumiste kategooriatele. Suhtarvude konkreetne arvuline väärtus sobib lisaks ärilise indeksi arvutamiseks ka kahe projekti omavaheliseks võrdluseks, nagu kirjeldatud peatükis (3.2.2). Kuna mõlemat projekti on hinnatud samade kriteeriumite järgi. Töös näidisena käsitletud kahe projekti kontekstis võime teha järelduse, et tehnilise võla suhtes on projekt B selgelt suuremate puudujääkidega – jagades tehnilise võla 168 päeva koodibaasi suurusega 1700 tuhat koodirida (0.0988) on selge, et tehnilise võla tihedus on suurem kui projekti A 128 päeva jagatud 2100 tuhat koodirida (0.0609). Ehk tuhande rea koodi kohta on tehniline võlg projektis B hinnanguliselt 30% suurem.

4.3 Tehnilise võla prioritiseerimine

Et saada ettekujutus, kuidas tuleks prioritiseerida projektides leiduva tehnilise võla vähendamist, tuleb SQ poolt salvestatud andmetega teha mõned operatsioonid. Standardlahenendusena ei paku SQ ärilise mõju indeksit. Küll aga on tarkvaral olemas veebiteenus, millelt on võimalik pärida kõiki koodis leiduvaid probleeme, koos nende peale kuluva leevendusaja, prioriteedi ning muude andmetega. Sellest tulenevalt kirjutas töö autor skripti [22], mis SQ veebiteenust kasutades kogub kokku kõik projekti juures leitud probleemid ning grupeerib need failide kaupa, summeerides kogu leevendusaja faili kohta ning summeerides ka prioriteedile vastava faktori. Siinkohal mainib autor, et ärilise mõju faktori väärtustena kasutati allikas [8] leiduvaid faktorite väärtuseid:

Category	Description	Sample type	Nonremediation factor
Blocking	Could result in a bug	Division by zero	5,000
High	Will have a high direct impact on the maintainance cost	Copy and paste	250
Medium	Will have a medium potential impact on the maintainance cost	Complex logic	50
Low	Will have a low impact on the maintainance cost	Naming convention	15
Report	Has very low impact—it's just a remediation cost report	Presentation issue	2

Joonis 8. Prioriteetidega seotud suhtarvud[8]

Skripti väljundiks oli CSV formaadis fail, mida oli võimalik kontoritarkvaraga lugeda ning mille iga rida vastas failile projektis, millele oli lisatud kogu leevendusaeg minutites ning prioriteedifaktor summeerituna. Lihtsuse huvides on tulemusena saadud klasside nimetused muudetud vastavalt koodile **Class_<Järjekorranumber - (prioriteet)><Projekti nimi>_<iteratsioon>**. Ehk siis Class_1A on SQALE meetodi rakendamise tulemina projekti A kõige prioriteetsem - ehk kõige suurema tulu/kulumi suhtega klass, mida oleks äriliselt kõige otstarbekam refaktoorida.

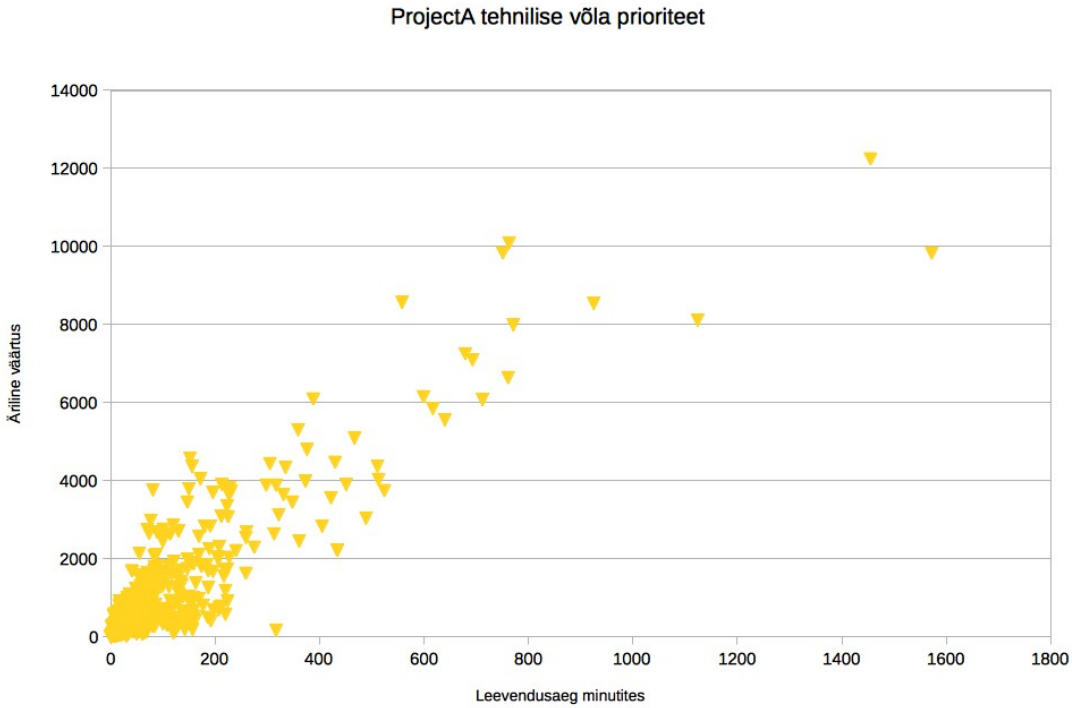
4.3.1 Projekti ärilise mõju indeksid

Järjestades projekti failid ärilise mõju indeksi järgi, oli tulemuseks märksa ühtlasem jaotis, kui toodud näidetes allikates [8] ning [9]. Graafikult tundub, et enamus projekti failides sisalduva tehnilise võla hulk on võrdelises seoses võla vähendamiseks kuluva

leevendusajaga. Teisisõnu ei eristu oluliselt, et mõne failiga seotud tehnilise võla ja leevendusaja suhe oleks astmeliselt erinev. Töö autor ootas ärilise mõju indeksitelt märksa volatiilsemat jaotust, koos konkreetselt välja paistvate ekstreemumitega.

Tabel 3: Projekti A klasside prioriteet SQALE perspektiivist

Moodul	Id	Äriline tähtsuse suhtarv	Leevendusaeg minutites	SBI
ProjectAB_cms_common	Class_1A_1	3640	227	16.03
ProjectAB_checkout	Class_2A_1	3345	223	15.00
ProjectAB_klarna	Class_3A_1	4435	305	14.54
ProjectAB_b2b	Class_4A_1	10092	763	13.22
ProjectAB_common	Class_5A_1	3885	298	13.03
ProjectA_seshipping	Class_6A_1	4795	376	12.75
ProjectAB_b2b	Class_7A_1	3884	317	12.25
ProjectA_klarna	Class_8A_1	7250	679	10.67
ProjectA_se	Class_9A_1	4470	430	10.39
ProjectAB_klarna	Class_10A_1	7985	771	10.35
ProjectAB_klarna	Class_11A_1	6147	599	10.26
ProjectAB_klarna	Class_12A_1	7095	693	10.23
ProjectA_halle	Class_13A_1	3450	348	9.94
ProjectAB_cms_common	Class_14A_1	3125	322	9.70
ProjectA_se	Class_15A_1	5844	617	9.47



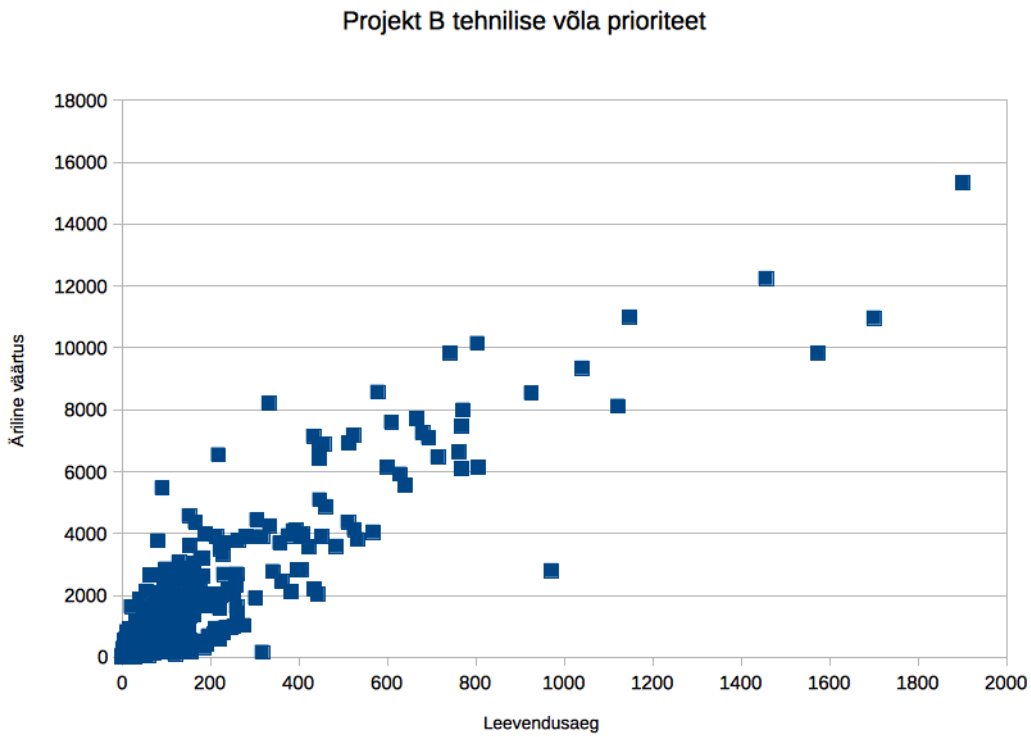
Joonis 9. Projekti A tehnilise võla prioriteedi graafik

Tehnilise võla prioriteedi graafikul kujutab iga andmepunkt ühte projekti faili(klassi). Y-teljel on ära toodud ärilise tähtsuse suhtarv ning X-teljel iga faili peale kuluv leevendusaeg minutites. Nagu peatükis 3.3.1 mainitud, tuleb antud graafikut lugeda diagonaalis, alustades vasakust ülemisest nurgast, sel viisil saame ärilise väärtuse ning leevendusaja suhte. Enamik tehnilise võla prioriteetidest on koondunud graafiku vasakule nurka ehk tegu on madala ärilise väärtuse ning madala leevendusajaga probleemidega. Et saada prioriteetidest paremat ülevaadet, kui seda on võimalik graafikult välja lugeda, on suurema ärilise väärtuse ning leevendusaja suhtega klassid eraldi ära toodud tabelis.

Tabel 4: Projekti B klasside prioriteet SQALE perspektiivist

Moodul	Id	Arilise tähtsuse suhtarv	Leevendusaeg minutites	SBI

ProjectAB_common	Class_1B_1	2590	96	26.97
ProjectB_bookanemployee	Class_2B_1	2610	155	16.83
ProjectAB_checkout	Class_3B_1	3330	228	14.60
ProjectAB_klarna	Class_4B_1	4435	305	14.54
ProjectB_storelocator	Class_5B_1	3780	262	14.42
ProjectAB_common	Class_6B_1	2630	183	14.37
ProjectB_common	Class_7B_1	3885	298	13.00
ProjectB_dk	Class_8B_1	4245	334	12.70
ProjectB_b2b	Class_9B_1	10142	803	12.63
ProjectB_dk	Class_10B_1	7595	609	12.47
ProjectB_dk	Class_11B_1	3884	317	12.25
ProjectAB_common	Class_12B_1	2680	230	11.65
ProjectB_dk	Class_13B_1	7715	666	11.58
ProjectB_klarna	Class_14B_1	7250	679	10.67



Joonis 10. Projekti B tehnilise võla prioriteedi graafik

Projekti B tehnilise võla jaotis on üldjoontes sarnane projekti A võla jaotisele, projekti B puhul on pisut enam selgemalt eristuvad ekstreemumid ning graafiku maksimaalsed väärtused on suuremad, mis viitab mõnele tehnilise võla mõistes erakordselt suure võla hulgaga failile, graafiku paremal üleval servas asuvad punktid.

4.4 Tulemuste analüüs ning esitamine

4.4.1 Tulemuste kontrollimise vorm

Leitud tehnilise võla prioriteetsemate failide kontrollimine toimus poolstruktueeritud intervjuu vormis. Intervjueeritavad on viimase kahe aasta jooksul töötanud analüüsiks valitud projektidega ning on kursis nende projektide varasema käekäigu ning praeguse seisuga, töötades projektidega igapäevaselt.

Intervjuu käigus tegi intervjueerija märkmeid avalikult, nii et need olid intervjueeritavale pidevalt nähtaval ning intervjueeritaval oli võimalus sekkuda, kui talle tundus, et tema seisukohti on valesti mõistetud.

Järgnevas tabelis on ära märgitud isikud, kellega on tehtud intervjuud, millise projektiga on nad seotud, kui pikk on nende töökogemus tarkvara arendamise valdkonnas ning positsioon hetkel.

Valim on mittetõenäosuslik sihipärane kvootvalim [23] nii, et mõlema projekti tehnilise võla prioriteetide tulemusi hindab vähemalt 2 antud projektiga igapäevaselt töötavat arendajat. Intervjuu käigus esitatakse 7 üldist küsimust tehnilise võla ning sellega kokkupuute kohta ning seejärel palutakse hinnata nimekirja klasside loendist, mis on sünteesitud peatükis 4. Nimekirja hindamine toimub ülejäänud intervjuust eraldi, andes intervjueeritavatele aega süveneda nimekirja ning võimaldada tutvuda failide sisuga. Tagasisidena oodati iga faili kohta otsust, kas see peaks kuuluma tehnilise võla prioriteetide hulka või mitte. Negatiivse vastuse korral olid intervjueeritavad oodatud paari lausega selgitama, miks nende arvates valik sobiv ei ole.

Hindamine on jagatud kaheks: kas antud klassi kuulumisega prioritiseeritud nimekirja nõustutakse või mitte.

Tabel 5: Intervjueeritud arendajad

Intervjueeritava kood	Projekt	Töökogemus	Positsioon hetkel
D1	Projekt B	8 aastat	Backend developer
D2	Projekt A	6 aastat	Backend developer
D3	Projekt A	12 aastat	Backend developer
D4	Projekt A	7 aastat	Backend developer
D5	Projekt B	10 aastat	Backend developer

4.4.2 Intervjuu koostamine

Intervjuu käigus saadakse ülevaade intervjueeritava kogemustest tarkvara arenduse valdkonnas ning selgitatakse välja, kas intervjueeritaval on olnud varasemalt kokkupuuteid tehnilise võla mõistega. Vajadusel lastakse kirjeldada varasemaid kogemusi, nende puudumisel antakse lühiülevaade, mida selle intervjuu käigus mõistetakse tehnilise võla all. Lisaks uuritakse, kui tihti ning mil määral mõjutab tehnilise võla olemasolu projektis arendaja igapäevast tööd ning mil määral on arendaja ise loonud juurde tehnilist võlga. Küsimustiku teise osana presenteeritakse varasemalt projekti analüüsides SQALE meetodit kasutades saadud tulemusi ning palutakse intervjueeritaval anda hinnang, mil määral nõustutakse (või ei nõustuta) antud tulemustega. Intervjuu peamised küsimused olid ette määratud, kuid vajadusel esitatakse täpsustavaid küsimusi ning selgitusi. Oluline on, et enne tehnilist võlga puudutavate sisuliste küsimusteni jõudmist on töö autor ning intervjueeritaval ühene arusaam, mida antud intervjuu raames tehnilise võla mõiste all mõeldakse.

4.4.3 Intervjuu D1

Esimene intervjueeritav pole varasemalt tehnilise võla mõistega kokku puutunud, kuid pärast kirjeldust, mida antud intervjuu (ning sellega seotud magistritöö) käigus tehnilise võla all mõistetakse tunneb ära tehnilise võlaga kokkupuutepunktid igapäeva töös. Selgub, et arendaja kulutab oma igapäevaseid tööülesandeid täites kuni 50% (intervjueeritava vastus 35-50%) ajast tehnilise võlaga tegelemisele, mainides et tehnilisele võlale kulunud aeg on otseses sõltuvusest komponendist, mida parasjagu käesolev tööülesanne puudutab. Samas kui tehnilise võlaga kokkupuutumise sagedust hindab arendaja vaid 2 palli väärtusele 10-st. Intervjueeritav on ise teadlikult tehnilist võlga juurde tekitanud, peamine põhjus on kiire loomuga ülesanded, näiteks jooksvad kriitilised haldusülesanded, kus lahenduse leidmine probleemile kaalub üles arhitektuurilise korrektsuse. Samas kaasneb võla võtmisega ainult mõnikord dokumenteerimine. Projektis varasemalt (enne antud arendaja projekti lülitumist) eksisteeriva tehnilise võla tekkepõhjuseks arvab arendaja eelnevate programmeerijate puudulikke teadmisi antud raamistikust.

Hinnang SQALE meetodil äriiindeksi järgi prioritiseeritud klassidele:

Nõustub: Class_1B_1, Class_2B_1, Class_7B_1, Class_8B_1, Class_12B_1, Class_13B_1

Ei nõustu:

- Class_3B_1 - ei tuvastanud märkimisväärset tehnilist võlga
- Class_5B_1 - sisaldab võla elemente, kuid on prioriteetsemaid klasse
- Class_6B_1 - sisaldab võla elemente, kuid on prioriteetsemaid klasse
- Class_9B_1 - ei tuvastanud märkimisväärset tehnilist võlga
- Class_11B_1 - sisaldab võla vähesel määral, peaks olema madalam prioriteet
- Class_14B_1 - ei tuvastanud märkimisväärset tehnilist võlga
- Class_4B_1 - tehniline võlg märkimisväärne, kuid moodulina tegu vähemtähtsa osaga süsteemist
- Class_10B_1 - tehniline võlg märkimisväärne, ei kuulu prioriteedi alla

Peamine põhjus mittenõustumisel oli asjaolu, et projektis on kriitilisemaid kohti, mida peaks prioritiseerima enne valitud klasse. Mõneti üllatav oli asjaolu, et mõnes failis ei tuvastanud arendaja praktiliselt mingeid puudujääke.

4.4.4 Intervjuu D2

Intervjueeritav on varem kuulnud tehnilise võla mõistest, kuid sisulisse poolde süvenenud ei ole. Pärast teemaarendust tehnilise võla teemal selgub, et olenevalt tööülesandest kulub intervjueeritaval tehnilise võlaga seotud tegevusteks 10-15% ajast. Tehnilise võlaga kokkupuutumise sagedust hindab arendaja hindega 4/10. On teadlikult tekitanud juurde tehnilist võlga, peamine põhjus on võimaliku refaktooriimisega seotud aja puudumine ning oht, et muudetav süsteemis osa muutub ebastabiilsemaks - selle põhjendusena tõenäoliselt ühiktestide puudumine legacy koodis. Teadliku tehnilise võla võtmisele eelneb konsulteerimine teiste arendajatega ning kommentaar JIRA-s.

Projektis varasemalt eksisteerinud tehnilise võla tekkepõhjuseks pakub arendaja varasemate arendajate hoolimatuse või vajalike tehniliste teadmiste puudumise.

Hinnang prioritiseeritud klassidele:

Nõustub:

Class_1A_1, Class_3A,_1 Class_5A,_1 Class_6A_1, Class_8A_1, Class_10A_1, Class_11A_1, Class_12A_1

Ei nõustu:

- Class_1A_1 - ei ole piisavalt prioriteetne klass, kuigi sisaldab võlga
- Class_2A_1 - tehnilist võlga minimaalselt
- Class_4A_1 - sisaldab tehnilist võlga, kuid on tõenäoline, et seda klassi ei muudeta mitte kunagi tulevikus
- Class_7A _1- sisaldab tehnilist võlga, kuid pole prioriteetne
- Class_9A_1
- Class_13A_1 - sisaldab vähesel määral võlga, ei peaks kuuluma kõige prioriteetsemate hulka
- Class_14A _1- ei sisalda märkimisväärset kogust tehnilist võlga

Peamiseks mittenõustumise põhjuseks oli ka intervjuu D2 puhul asjaolu, et projektis on olemas klasse, milles sisalduva tehnilise võla vähendamisega tuleks tegeleda enne väljapakutud nimekirja.

4.4.5 Intervjuu D3

Intervjuu käigus selgus, et arendaja on kursis tehnilise võla mõistega ning see ühtib enamjaolt antud töös ning intervjuus kasutatud käsitlusega. Arendaja sõnul puutub ta tehnilise võlaga kokku sagedusega 7/10 ning hinnanguliselt kulub tal tehnilise võlaga

tegelemise peale 30%-45% kogu ajast olenevalt komponendist, mille kallal hetkel töötatakse. Arendaja on teadlikult võtnud uut tehnilist võlga ning enamasti eelneb sellele konsultatsioon meeskonnakaaslastega ning vajadusel kogu meeskonna teavitamine e-maili teel. Arendaja arvates on varasemalt olemasolev võlg peamiselt teadmatusest ning raamistiku mittetundmisest põhjustatud.

Hinnang prioritiseeritud klassidele:

Nõustub: Class_2A_1, Class_6A_1, Class_7A_1, Class_9A_1, Class_13A_1, Class_15A_1

Ei nõustu: Class_1A_1, Class_3A_1, Class_4A_1, Class_5A_1, Class_8A_1, Class_9A_1, Class_10A_1, Class_11A_1, Class_12A_1, Class_14A_1

Kõikide failide mittenõustumist põhjendab intervjueeritav faktiga, et tegu on mitmes projektis kasutatavate moodulitega, mille haldamisega ei tegele antud meeskond. Seepärast pole antud hinnangu puhul ka eraldi välja toodud põhjendust iga klassi kohta. Arendaja sõnul on esmatähtis tegeleda moodulitega, mis on välja arendatud spetsiaalselt selle projekti käigus. Mooduleid küll kasutatakse, kuid enamasti n-ö teekidena, mida projekti poolt muudetakse suhteliselt harva.

4.4.6 Intervjuu D4

Varasemalt pole intervjueeritav tehnilise võla mõistega kursis. Pärast täiendavaid selgitusi leiab intervjueeritav, et puutub tehnilise võlaga kokku hinnanguliselt tihedusega 5/10, ning kogu tema ajast kulub umbes 20-30% tehnilise võlaga tegelemisele. Olles ise tehnilist võlga ka teadlikult juurde tekitanud, leiab arendaja, et enamasti on tehnilise põhjuse tekkepõhjuseks ajapuudus mõne konkreetse ülesande lahendamisel või pole olemasoleva võla vähendamiseks eelarvet ette nähtud. Projektides varasemalt sisalduva tehnilise võla põhjuste kohta pole arendajal kindlat arvamust.

Hinnang prioritiseeritud klassidele:

Nõustub: ClassA_6,_1 ClassA_7_1, ClassA_9_1, ClassA_13_1, ClassA_15_1

Ei nõustu:

- ClassA_1_1 - mõningane tehniline võlg, meeskond ei vastuta mooduli eest
- ClassA_2_1 - tehnilist võlga esineb, kuid ei tohiks kuuluda prioriteetsemate hulka
- ClassA_3_1 - tehnilist võlga peaks vähendama mooduli haldaja
- ClassA_4_1 - autogenereeritud kood, mis on suures osas staatiline, muutmise tõenäosus on väike
- ClassA_5_1 - võlg esineb, kuid koodi tõenäoliselt enam projektis ei kasutata
- ClassA_8_1 - tehnilist võlga peaks vähendama mooduli haldaja
- ClassA_10_1 - funktsionaalsust ei kasutata, antud moodulist kasutatakse väikest osa
- ClassA_11_1 - funktsionaalsust ei kasutata, antud moodulist kasutatakse väikest osa
- ClassA_12_1 - funktsionaalsust ei kasutata, antud moodulist kasutatakse väikest osa
- ClassA_14_1- mõningane tehniline võlg, meeskond ei vastuta mooduli eest

4.4.7 Intervjuu D5

Arendaja pole varasemast tuttav tehnilise võla mõistega. Igapäevatöös hindab ta tehnilise võlaga kokkupuutumise tiheduseks 7/10. Intervjueeritav kulutab keskmiselt 10% oma ajast tehnilise võlaga seotud probleemide lahendamisele, tunnistab, et tal on keeruline tehnilist võlga ära tunda ning eelneva koodi paremaks muutmine ei kuulu otseselt tema tööülesannete hulka. Tehnilist võlga on juurde loodud peamiselt situatsioonides, kus on vajadus teha kiireloomulisi parandustöid kasutades tihti arhitektuuriliselt mitte kõige korrektsemaid lähenemisi. Arendaja tunnistab, et arutamine meeskonnakaaslastega pole alati tulus, kuna tihtipeale esineb eriarvamusi. Minevikus kogunenud tehnilise võla kohta annab arendaja hinnangu, mis viitab peamise põhjusena teadmatusele või hoolimatusele.

Hinnang prioritiseeritud klassidele:

Nõustub: ClassB_7_1, ClassB_10_1, ClassB_13_1

Ei nõustu:

- ClassB_1_1 - sisaldab vähesel määral tehnilist võlga, ei ole piisavalt kriitilises seisus
- ClassB_2 _1- sisaldab tehnilist võlga, kuid ei tohiks olla prioritiseeritud
- ClassB_3 _1- ei sisalda piisavalt tehnilist võlga et olla prioritiseeritud
- ClassB_4_1 - sisaldab tehnilist võlga, kuid ei tohiks olla prioritiseeritud
- ClassB_5_1 - ei sisalda piisavalt tehnilist võlga et olla prioritiseeritud
- ClassB_6_1 - ei sisalda piisavalt tehnilist võlga et olla prioritiseeritud
- ClassB_8_1 - sisaldab tehnilist võlga, kuid ei tohiks olla prioritiseeritud
- ClassB_9_1 - ei sisalda piisavalt tehnilist võlga et olla prioritiseeritud
- ClassB_11_1 - ei sisalda piisavalt tehnilist võlga et olla prioritiseeritud
- ClassB_12_1 - ei sisalda piisavalt tehnilist võlga et olla prioritiseeritud
- ClassB_14_1 - ei sisalda piisavalt tehnilist võlga et olla prioritiseeritud

Hinnang prioriteetidele võib autori hinnangul olla mõjutatud intervjueeritava oskusest tehnilist võlga ning koodiga seotud probleeme ära tunda. Põhjenduseks on lisaks ka tõsiasi, et paljude klasside puhul ei ole probleeme esinenud, kuna meeskonna poolt muudetakse antud koodi harva või praktiliselt mitte kunagi, ning väidetavalt leidub projektis koodi, mida tuleks prioritiseerida enne antud nimekirja.

4.4.8 Kokkuvõte intervjuudest

Intervjuude baasilt selgub, et tehniline võlg on antud projektides aktuaalne - küsitletud arendajatel kulub võlaga tegelemisele (intressimaksetele) 15-50% kogu tööajast. Küll aga selgub SQALE meetodil prioritiseeritud ajakulu/ärilise väärtuse suhte määrad ei ühti projektiga tegelevate arendajate arvamusega tehnilise võlaga tegelemise prioriteetidega. Projekti A puhul kattus SQALE analüüsi teel saadud tulemus ligi 30% (5/15) ulatuses arendajate arvamusega prioritiseerimisest(vt tabel). Tehnilise võla tuvastamise osas oldi nõus – antud klassid sisaldasid olulisel määral tehnilist võlga. Projekti B puhul oli tulemus 21.4% (3/12) mõõtes saadud tulemustest(vt tabel). Projekti B puhul esines ka teatavaid möödarääkivusi tehnilise võla tuvastamise osas, vajalik oli eraldi täpsustused, miks ja kuidas on mõned prioritiseeritud probleemsed klassid tulemina saadud. Intervjuudest selgub, et otstarbekas oleks analüüsitava tarkvara valides kasutada dimensioonina mitte ainult hiljuti muudetud mooduleid, vaid lisaks filtreerida tulemust kasutades ekspertarvamust. Antud töös käsitletud valikus olevad moodulid olid viimase 2 aasta jooksul küll aktiivselt arenduses, kuid neid kasutati, kui teeke ning nende eest vastutavad olid teised meeskonnad.

Eranditult kõik intervjueeritavad on teadlikult tehnilist võlga ise juurde loonud ning peamine põhjus selleks on olnud ajapuudus, kuna muudatus on oma loomult ajakriitiline. Samuti märkisid kõik intervjueeritavad ära lisanduva tegevusena meeskonnakaaslastega tehnilise lahenduse arutamise, erinevused olid dokumenteerimisega seoses – enamus (4/5) intervjueeritavatest tegi sellekohase märke projektijuhtimiseks kasutatavasse keskkonda. Samas tunnistati, et mõnikord ununeb dokumenteerimine ning ühel juhul ei saadud meeskonnakaaslastega arutades kokkuleppele loodava lahenduse suhtes.

Tabel 6: Projekt A kattuvus arendajate hinnanguga

	Arendaja D2	Arendaja D3	Arendaja D4
Class_1A	Nõustub	Ei nõustu	Ei nõustu
Class_2A	Ei nõustu	Nõustub	Ei nõustu
Class_3A	Nõustub	Ei nõustu	Ei nõustu
Class_4A	Ei nõustu	Ei nõustu	Ei nõustu
Class_5A	Nõustub	Ei nõustu	Ei nõustu
Class_6A	Nõustub	Nõustub	Nõustub
Class_7A	Ei nõustu	Nõustub	Nõustub
Class_8A	Nõustub	Ei nõustu	Ei nõustu
Class_9A	Ei nõustu	Nõustub	Nõustub
Class_10A	Nõustub	Ei nõustu	Ei nõustu
Class_11A	Nõustub	Ei nõustu	Ei nõustu
Class_12A	Nõustub	Ei nõustu	Ei nõustu
Class_13A	Ei nõustu	Nõustub	Nõustub
Class_14A	Ei nõustu	Ei nõustu	Ei nõustu
Class_15A	Ei nõustu	Nõustub	Nõustub

Tabel 7: Projekt B kattuvus arendajate hinnanguga

	Arendaja D1	Arendaja D5
Class_1B	Nõustub	Ei nõustu
Class_2B	Nõustub	Ei nõustu
Class_3B	Ei nõustu	Ei nõustu
Class_4B	Ei nõustu	Ei nõustu
Class_5B	Ei nõustu	Ei nõustu
Class_6B	Ei nõustu	Ei nõustu
Class_7B	Nõustub	Nõustub
Class_8B	Nõustub	Ei nõustu
Class_9B	Ei nõustu	Ei nõustu
Class_10B	Ei nõustu	Nõustub
Class_11B	Ei nõustu	Ei nõustu
Class_12B	Nõustub	Ei nõustu
Class_13B	Nõustub	Nõustub
Class_14B	Ei nõustu	Ei nõustu

Intervjuudest tehnilise võla tekkepõhjusi analüüsides on konkurentsitult kõige rohkem mainitud tekkepõhjusena praktilisest vajadusest tulenev tehniline võlg. Siinkohal on huvitav nähtus, et arendajad pakuvad varasemalt projektis esinenud tehnilise võla tekkepõhjuseks teadmatust või raamistiku mittetundmist, samas ise tehnilist võlga juurde tekitades on mainitud peamine põhjus praktiline vajadus. Tehnilise võla klassifitseerimiseks kasutame siinkohal peatükis 2.3 välja pakutud kriteeriume.

Tabel 8: Tehnilise võla tekkepõhjused intervjuudest

	D1	D2	D3	D4	D5
Praktiline vajadus	1	1	1	1	1
Prioritiseerimine	0	0	0	0	0
Suhtumine / Hoiak	0	1	0	0	0
Teadmatus / eksimused	1	1	1	0	1

4.5 Tulemuste täiustamine

Kuna esmasel analüüsil saadud tulemused ei langenud kokku arendajate arvamusega, jäeti eksperthinnangut kasutades kõrvale moodulid, mis küll on loodud antud ettevõtte poolt, kuid mida kasutatakse justkui teeke ning projektiga tegelevate meeskondade poolt igapäevaselt ei arendata. Siinkohal mainib autor, et esmase analüüsi tulemused olid korrektsed tehnilise võla tuvastamise suhtes, kuid mitte antud projekti kontekstis. Kuna see ei korreleeru töö eesmärkidega, oli vajalik arendajatepoolne hinnang, milliseid moodulid tuleks esialgsest tulemusest kõrvaldada. Pärast valiku tegemist viidi läbi uus analüüs, mille tulemusena saadi järgmise järgnevad tulemused:

Tabel 9: Projekt A tehnilise võla prioriteet (iteratsioon 2)

Id	Ärilise tähtsuse suhtarv	Leevendusaeg minutites	SBI
Class_1A_2	1310	57	22.98
Class_2A_2	4795	376	12.75
Class_3A_2	4470	430	10.39
Class_4A_2	2685	260	10.32
Class_5A_2	2040	205	9.95
Class_6A_2	5844	617	9.47
Class_7A_2	3900	451	8.64
Class_8A_2	3560	422	8.53
Class_9A_2	2635	313	8.41
Class_10A_2	8112	1124	7.21
Class_11A_2	2442	361	6.76
Class_12A_2	1865	159	11.72
Class_13A_2	3450	348	9.91
Class_14A_2	2300	208	11.05
Class_15A_2	1280	112	11.42

Tabel 10: Projekt A tehnilise võla prioriteet (iteratsioon 2)

Id	Ärilise tähtsuse suhtarv	Leevendusaeg minutites	SBI
Class_1B_2	2590	96	27.26
Class_2B_2	2300	102	22.54
Class_3B_2	2610	155	16.8
Class_4B_2	3910	280	13.96
Class_5B_2	2685	260	10.3
Class_6B_2	7715	666	11.58
Class_7B_2	3980	407	9.77
Class_8B_2	3900	404	9.65
Class_9B_2	6462	714	9.05
Class_10B_2	3900	451	8.64
Class_11B_2	9346	1040	8.98
Class_12B_2	4125	535	7.71
Class_13B_2	3585	484	7.40
Class_14B_2	15336	1899	8.07
Class_15B_2	5915	628	9.51

Pärast tulemuste saamist küsiti taas arendajate arvamust prioriteetide suhtes. Küsitlus viidi läbi sarnaselt eelpool mainitud intervjuude käigus toimunud küsimustega. Projekti A puhul ei olnud arendajad nõus 1 prioritiseeringuga 15-st ehk nõustuti 95% määraga pakutud tehnilise võla prioriteetidega.

Tabel 11: Projekt A parandatud tulemused

	Arendaja D2	Arendaja D3	Arendaja D4
Class_1A_2	Nõustub	Nõustub	Nõustub
Class_2A_2	Nõustub	Nõustub	Ei nõustu
Class_3A_2	Nõustub	Nõustub	Nõustub
Class_4A_2	Ei nõustu	Nõustub	Nõustub
Class_5A_2	Nõustub	Nõustub	Nõustub
Class_6A_2	Nõustub	Nõustub	Nõustub
Class_7A_2	Nõustub	Nõustub	Nõustub
Class_8A_2	Nõustub	Nõustub	Nõustub
Class_9A_2	Nõustub	Nõustub	Ei nõustu
Class_10A_2	Nõustub	Nõustub	Nõustub
Class_11A_2	Nõustub	Nõustub	Nõustub
Class_12A_2	Nõustub	Nõustub	Nõustub
Class_13A_2	Nõustub	Nõustub	Nõustub
Class_14A_2	Ei nõustu	Ei nõustu	Ei nõustu
Class_15A_2	Nõustub	Nõustub	Ei nõustu

Tabel 12: Projekt B parandatud tulemused

	Arendaja D1	Arendaja D5
Class_1B_2	Nõustub	Ei nõustu
Class_2B_2	Nõustub	Ei nõustu
Class_3B_2	Nõustub	Ei nõustu
Class_4B_2	Nõustub	Nõustub
Class_5B_2	Nõustub	Ei nõustu
Class_6B_2	Nõustub	Nõustub
Class_7B_2	Nõustub	Nõustub
Class_8B_2	Nõustub	Nõustub
Class_9B_2	Nõustub	Nõustub
Class_10B_2	Nõustub	Nõustub
Class_11B_2	Nõustub	Nõustub
Class_12B_2	Nõustub	Nõustub
Class_13B_2	Nõustub	Ei nõustu
Class_14B_2	Nõustub	Nõustub
Class_15B_2	Nõustub	Nõustub

Projekti B puhul kõikus tulemus olulisel määral, arendaja D1 oli nõus kogu nimekirjaga, kuid arendaja D5 leidis, et 5 klassi puhul ei langeks tema arvamus prioritiseerimisel kokku analüüsi tulemustega. Siinkohal peab ära mainima, et arendaja D5 puhul on mainitud eelneva intervjuu käigus (peatükk 4.4.7), et talle valmistab tehnilise võla äratundmine probleeme. Projekti B suhtes loeb autor prioritiseerimise õnnestumise määraks 66% ehk 10/15.

Autori hinnangul sobivad parandatud tulemused tehnilise võla prioritiseerimiseks ning selle baasilt on võimalik luua protsesse, milles käigus pidevalt analüüsides on võimalik efektiivselt tehnilist võlga vähendada.

4.6 Tehnilise võlaga seotud protsessid ettevõttes

Käesoleval momendil ei tegele uuritud meeskonnad süstemaatiliselt tehnilise võla haldamise ega prioritiseerimisega. Tehnilise võla haldus, mis puudutab võla tagasimaksmist on suures osas juhuslik ning seotud parasjagu käsil olevate arendusülesannetega – tehnilise võla vähendamine ei ole eraldiseisev ülesanne.

Protsesside kontrollimise faasi peamine ülesanne on parandamise faasis väljatöötatud kasu aktuaalsena hoidmine võimalikult pika ajaperioodi vältel. Selle jaoks on vajalik

protsesside standardiseerimine ning dokumenteerimine[36]. Antud töö raames võib standardiseerimise all näha ettevõtte sisese poliitika kehtestamist, mis näeb ette tehnilise võla vähendamist vastavalt väljatöötatud prioritiseerimise mehhanismile. Samuti ka automatiseerimist sellisel tasemel, et tulemuste saamiseks piisaks vaid analüüsi käivitamise otsusest.

Teiseks tuleb välja töötada seirekava, millega kontrollida, kas ning mil määral on protsess edukas olnud. Selle tarbeks pakub töö autor välja järgneva kontrollmehhanismi: pärast prioritiseerimist ning nendele vastavate klasside refaktoorimist käivitatakse prioritiseerimine uuesti ning tulemuseks peab olema erinev prioriteet võrreldes eelmise iteratsiooni alustamisega, võttes arvesse süsteemi osi, mida parasjagu refaktooriti.

Töö autor paneb ette määrata kindlaks protsess, kuidas on meeskondadel võimalik igapäevaselt prioritiseerida ning vähendada tehnilist võlga projektide kriitilistes punktides. Protsess koosneb järgmistest etappidest:

1. Iga iteratsiooni (sprinti) lisatakse arendusülesanne, mis seisneb tehnilise võlaga seotud koodi refaktoorimises - Projektijuht

2. Projekti kood analüüsitakse antud töös kasutatud SQALE meetodil, leides huvipakkuvamad kohad tehnilise võla mõistes – Süsteem

3. Antud nimekirjast tehakse valik, mida refaktoorida konkreetse iteratsiooni käigus – Arendajad

4. Arendusülesanne piiritletakse (*time-box*) mingi hulga tundidega

5. Teostatakse arendusülesanne iteratsiooni (sprindi) käigus- Arendajad

6. Kontrollimiseks (samuti ka järgmiste iteratsioonide alguses) analüüsitakse projekti uuesti ning saadakse uus nimekiri prioriteetidest. - Süsteem

7. Kontrollitakse, et uus prioriteetide nimekiri ei oleks sama: viimati valitud komponentide tehniline võlg peaks olema langenud – Arendajad, Projektijuht

Protsesside jälgimise plaanina pakub töö autor välja kogu projekti SQALE kvaliteediindeksi järgimise ning dokumenteerimise. SQI peaks olema vähenenud iga

projekti iteratsiooniga. Võimalusel tuleks luua automatiseeritud analüsaator, mis analüüsib projekti viimaste iteratsioonide SQI väärtust, ning on võimeline hoiatama, kui selle trend peaks muutuma kasvavaks.

5 Kokkuvõte

Käesolev töö andis teoreetilise ülevaate tehnilise võla mõistest, tehnilise võla tekkepõhjustest ning tehnilise võlaga seotud riskidest. Lisaks anti teoreetiline ülevaade tehnilise võla prioritiseerimiseks kasutatavast meetodist SQALE ning sellega seotud suhtarvudest ning indeksitest.

Töö käigus analüüsiti kahte pikaajalist tarkvaraprojekti SonarQube nimelise tööriistaga ning leiti projekti klasside prioriteetne järjestus, mille alusel peaks hakkama toimuma tehnilist võlga sisaldavate projekti osade parandamine. Saadud tulemusi kontrolliti projektiga töötavate arendajatega läbi viidud poolstruktureeritud intervjuudega. Esialgsed tulemused ei rahuldanud töö kirjutajat ning hindamisele kuuluva tarkvara valimit korrigeeriti arendajate kaasabil. Parandatud meetodiga saadud prioriteedid kontrolliti taaskord küsides arendajate ekspertarvamust, teistkordne tulemus oli rahuldav. Projekte analüüsides leiti vastus küsimusele, kui suur on tehniline võlg nendes projektides praegusel momendil, leitud tulemused on kasutatavad projektide omavahelises võrdluses või mõnest järgnevast projektist ülevaate saamiseks.

Lisaks leiti vastused küsimustele tehnilise võla tekkepõhjuste kohta antud projektide juures. Intervjuudest selgus, et enamik arendajate poolt teadlikult juurde tekitatud tehnilisest võlast on juurde tekkinud praktilistel kaalutlustel ehk aja või ressursi puudumisel.

Töö viimases osas pakuti välja ideid protsesside loomiseks ning kontrollmeetodid, mida saab kasutada sisendina ettevõttes tehnilise võla tuvastamiseks, prioritiseerimiseks ning süstemaatiliseks vähendamiseks.

Kasutatud kirjandus

[1] The WyCash portfolio Management System, Ward Cunningham, 26.03.1992

[2] A systematic mapping study on technical debt and its management Zengyang Lia,∗, Paris Avgerioua, Peng Liangb, The Journal of Systems and Software 101 (2015) 193–220, [Online] ACM Digital Library (15.02.2017)

[3] Eric Allman, Managing Technical Debt, Queue, v.10 n.3, March 2012, [Online] ACM Digital Library (15.02.2017)

[4] An exploration of technical debt, The Journal of Systems and Software 86, 1498-1516, [Online] Science Direct (15.02.2017)

[5] Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study. Carlos Fernández-Sánchez, Juan Garbajosaa, Agustín Yagüea, Jennifer Pereza. The Journal of Systems and Software, Volume 124, February 2017, Pages 22–38, [Online] Science Direct (15.02.2017)

[6] Using Automatic Static Analysis to Identify Technical Debt, Antonio Vetro, Proceedings of the 34th International Conference on Software Engineering, June 02 - 09, 2012, Pages 1613-1615, [Online] IEEE/IET Electronic Library (IEL) (15.02.2017)

[7] Value-Based Technical Debt Model and Its Application, Marek G. Stochel, Mariusz R. Wawrowski, Magdalena Rabiej, ICSEA 2012 : The Seventh International Conference on Software Engineering Advances

[8] Managing Technical Debt with the SQALE Method, Jean-Louis Letouzey and Michel Ilkiewicz, IEEE Software ,Volume 29 Issue 6, November 2012, Pages 44-51, [Online] IEEE/IET Electronic Library (20.03.2017)

[9] The SQALE Method for Managing Technical Debt, [WWW], http://www.sqale.org/wp-content/uploads/2016/08/SQALE-Method-EN-V1-1.pdf (20.03.2017)

[10] Costs and obstacles encountered in technical debt management – A case study. Y. Guo et al. / The Journal of Systems and Software 120 (2016) 156–169, [Online] Science Direct (15.02.2017)

[11] A Threshold Based Approach to Technical Debt, Robert J. Eisenberg ACM SIGSOFT Software Engineering Notes, March 2012 Volume 37 Number 2, [Online] ACM Digital Library, 20.03.2017

[12] Comparing four approaches for technical debt identification,Nico Zazworka, Antonio Vetro',Clemente Izurieta, Sunny Wong, Yuanfang Cai, Carolyn Seaman,Forrest Shull ,Software Qual J (2014) 22:403–426, [Online] SpringerLink (20.03.2017)

[13] Help: log, [WWW] https://www.mercurial-scm.org/repo/hg/help/log (20.03.2017)

[14] Manual - Documentation - Zend Framework, [WWW] https://framework.zend.com/manual/1.11/en/coding-standard.naming-conventions.html (20.03.2017)

[15] Technical Debt, [WWW], https://docs.sonarqube.org/display/SONARQUBE52/Technical+Debt (20.03.2017)

[16] Eberhard Wolff and Sven Johann, IEEE Software, Volume: 32, Issue: 4, July-Aug. 2015, (02.), [Online] IEEE Xplore Digital Library (20.03.2017)

[17] TechnicalDebtQuadrant, [WWW], https://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html (08.04.2017)

[18] Prioritizing technical debt, [WWW], https://rishidean.com/2013/06/21/prioritizing-technical-debt/, Rishi Dean (08.04.2017)

[19] How to prioritize your technical debt, [WWW], https://blog.codacy.com/how-to-prioritize-your-technical-debt-35b5ce7ece9b (08.04.2017)

[20] The SQALE Quality and Analysis Models for Assessing the Quality of Ada Source Code, Thierry Coq, Jean-Pierre Rosen, 2009, [Online] SpringerLink (20.03.2017)

[21] Viigipuu R, Meetmeid tarkvara hallatavuse parendamiseks, magistritöö . Tallinna Tehnikaülikool 01.06.2015

[22] gist:ee0162a0074fd11dc6d8031cb2690989, [WWW] (08.04.2017) https://gist.github.com/anonymous/ee0162a0074fd11dc6d8031cb2690989

[23] Valimi moodustamine | Sotsiaalse Analüüsi Meetodite ja Metodoloogia õpibaas, [WWW], http://samm.ut.ee/valimid (08.04.2017)

[24] R Alidarso, Measuring Quality Improvements After Stimulating Software Quality Awareness Among Developers, magistritöö Delft University of Technology, 2012

[25] Prioritizing Design Debt Investment Opportunities, Nico Zazworka , Carolyn Seaman , Forrest Shull, Proceedings of the 2nd Workshop on Managing Technical Debt, May 23-23, 2011, [Online] ACM Digital Library (16.04.2017)

[26] Defining the decision factors for managing defects: a technical debt perspective, Will Snipes , Brian Robinson , Yuepu Guo , Carolyn Seaman,, Proceedings of the Third International Workshop on Managing Technical Debt, p.54-60, 05.06.2012, [Online] IEEE Xplore Digital Library (16.04.2017)

[27] Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study, Antonio Martini, Jan Bosch, Michel Chaudron, 2015, [Online] Science Direct (16.04.2017)

[28] Toward Measuring Defect Debt and Developing a Recommender system for their prioritization, Shirin Akbarinasaji, Doctoral Symposium on Empirical Software Engineering, 10.21.2015, [Online] CEUR Workshop Proceedings (16.04.2017)

[29] A Case Study on Effectively Identifying Technical Debt, Nico Zazworka, Rodrigo O. Spínola, Antonio Vetro, 16.04.2013, [Online] ACM Digital Library (16.04.2017)

[30] How technical debt turns into technical bankrupcy, [WWW], http://robertvbinder.com/how-technical-debt-turns-into-technical-bankruptcy/ (27.04.2017)

[31] Supporting Technical Debt Cataloging with TD-Tracker Tool, Lucas Borante Foganholi, Rogério Eduardo Garcia, Danilo Medeiros Eler, Ronaldo Celso Messias Correia, and Celso Olivete Junior, 2015, [Online] DOAJ Directory of Open Access Journals (29.04.2017)

[32] MSC09-C. Character encoding: Use subset of ASCII for safety - SEI CERT C Coding Standard - CERT Secure Coding Standards, [WWW] https://www.securecoding.cert.org/confluence/display/c/MSC09-C.+Character+encoding%3A+Use+subset+of+ASCII+for+safety (29.04.2017)

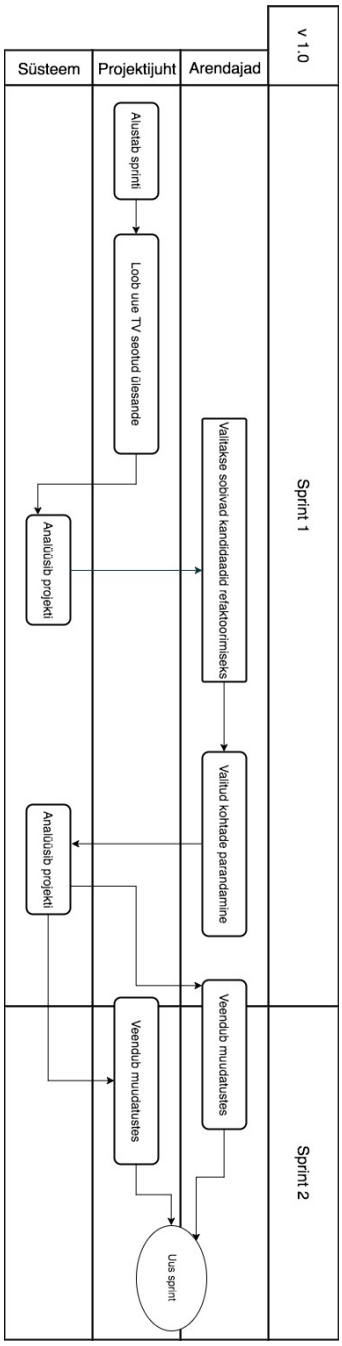
[33] PSR-2: Coding Style Guide, [WWW], http://www.php-fig.org/psr/psr-2/ (29.04.2017)

[34] Robert Cecil Martin , Clean Code, 2008, lk 53-74

[35] CWE-259: Use of Hard-coded Password (2.11), [WWW], http://cwe.mitre.org/data/definitions/259.html (29.04.2017)

[36] K. Desai Deepali, Six Sigma, 12. 2010

Lisa 1 - Protsesside visualiseering



Joonis 11. Tehnilise võla haldusega seotud protsessid

Lisa 2 - SonarQube reeglite muudetud prioriteedid

Tabel 13: SonarQubes töö käigus muudetud reeglid

Reegel	Vaikimisi väärtus	Muudetud väärtus
Funktsioonide kognitiivne keerukus ei tohi olla liiga kõrge	Major	Critical
Suurendamise (++) ja vähendamise (--) operaatoreid ei tohiks kasutada funktsiooni parameetrite sees ega loogilistes operaatorites (tingimuse osas)	Minor	Major
Read ei tohiks lõppeda tühikuga	Major	Minor
Read ei tohiks olla liiga pikad	Major	Minor
"exit(...)" ja "die(...)" funktsioone ei tohiks kasutada	Critical	Major
Funktsiooni nähtavus peab alati defineeritud olema	Major	Minor
Konstandid nagu „true”, „false” ja „null” peavad alati olema väikeste tähtedega	Major	Minor
Boolean väärtusi tagastavad meetodi peaksid algama "is" või "has" eesliitega.	Minor	Major