TARTU ÜLIKOOLI

SOTSIAALTEADUSTE VALDKOND

NARVA KOLLEDŽ

ÕPPEKAVA „INFOTEHNOLOOGILISTE SÜSTEEMIDE ARENDUS“

Denis Volkov

LÄHTEKOODI FAILIKOGU ANALÜÜSIMISE, STRUKTUREERIMISE JA OTSINGU INFOSÜSTEEMI LOOMINE

Diplomitöö

Juhendaja assistent Andre Sääsk

NARVA 2018

SISUKORD

SISSEJUHATUS

Käesolev töö on tehtud ühe tarkvaraarendusfirma tellimusel (edaspidi tellija). Tellimuse sisuks oli luua *infosüsteem*[1], mis osaliselt automatiseeriks tellija tarkvaraarendajate tööd seotud lähtekoodi otsingu ja analüüsimisega – infosüsteemi kasutamine annab võimaluse tarkvaraarendajale otsida lähtekoodi klassid ja meetodid automaatselt. Infosüsteemi loomine on aktuaalne, sest tellija tegevus eeldab suuremahulise lähtekoodi failikogu töötlemist.

Lõputöö eesmärgiks oli luua süsteem, mis täidaks eelnimetatud nõudeid. Selleks, et maksimaalselt täpselt täita tellija ootusi, on projekti arendamises osalenud üks tellija esindaja teise meeskonna liikmena[2].

Lõputöö ülesanded:

- Tellija lähtekoodi kogu analüüsimise alamsüsteemi arendamine;
- *Andmebaasi* genereerimise alamsüsteemi arendamine, mis koosneb objektidest, milles salvestatakse failidest leitud informatsioon;
- Andmebaasi kasutajaliidese loomine.

Failikogus otsitakse tarkvaraarendajale vajalikud *klassid* ja *meetodid*, mida ta parandab või mille põhjal loob uued andmestruktuurid.

Otsingusüsteemi ja mugava kasutajaliidese puudumine põhjustab tarkvaraarendajatele suurt ajakulu andme töötlemise aja pikenemise tõttu. Sellepärast on vaja luua infosüsteem, mis võimaldab valida lähtefailid, analüüsida ja töödelda nendes sisalduvaid andmeid, registreerida kõike leitud klasse ja meetodeid ning nende ühendusi.

Valmis süsteem peaks vastama järgmistele tingimustele:

- Lihtne tarkvara kasutusele võtmine ja sellega töötamine;
- Kasutajasõbralik liides;
- Windows ja MacOS operatsioonisüsteemides käivitamine;
- Võimalus otsida sõna või selle osa, mida sisaldab otsitav klassi või meetodi nimi;
- Otsitud koodi elemendi asukoha näitamine;
- Otsitud koodi elementide struktuuri näitamine hierarhilise puu kujul;

[1] Kaldkirjaga on märgitud mõisted, mille definitsioon on toodud esimeses osas asuvas sõnastikus.

[2] Iga meeskonna liikme panuseid projekti saab näha GiTHub-i repositooriumist (Code Analyzer, Commits 2017).

- Arendusprotsessi korraldamine niisugusel viisil, et otsinguliides ei näitaks analüüsitava keele süntaksi eripärasid.

Kui võtta arvesse kõiki eespool loetletud nõudeid ja omadusi, siis loodud tarkvara säästaks tööaega, vähendades käsitsi tehtud otsinguid ja parandaks ettevõtte klienditeeninduse kvaliteeti ja kiirust. See lubab teha järeldust infosüsteemi arendamise otstarbekuse kohta.

Tellija nõuete lahendamiseks kasutati järgmiseid tehnoloogiaid:

- *Programmeerimiskeel Java*;
- Integreeritud arenduskeskkond (IDE) Eclipse;
- *Kompileerimissüsteem* Maven;
- *Kaughoidla* korraldamiseks kasutati keskkonda GitHub;
- Testimiseks kasutati teeki *JUnit*;
- Konfiguratsioonifailide korraldamiseks kasutati *XML* keelt;
- Arendusmeetoditeks valiti *agiilsed arendusmeetodid SCRUM* ja *XP* (osaliselt).

Töö tulemusena saab valmis rakendus, mis aitab tellijal optimeerida arendajate tööaega, automatiseerides tarkvara moodulite lähtekoodi otsimise ja analüüsimise protsesse. See võimaldab arendajal pühendada rohkem aega nende väljatöötatud koodi kvaliteedile ning paremini taaskasutada varem loodud koodi.

Lõputöö kirjalik osa koosneb kolmest peatükist. Esimeses peatükis on defineeritud töös kasutatavad mõisted. Teises peatükis autor loetleb valitud arendustehnoloogiaid ja tingimusi, mis mõjutasid tehtud valikuid. Kolmas peatükk kirjeldab projekti ülesande püstituse tausta, tarkvaraprojekti struktuuri ja tulemusi.

1 PROJEKTIS KASUTATUD MÕISTED

Infosüsteem

Infosüsteem on andmetöötlussüsteemi tarkvara osa, mis sisaldab andmebaasi ja kasutajaliidest.

Andmetöötlussüsteem on süsteem, mis teeb sisendandmetega mitmesuguseid matemaatilisi operatsioone, eesmärgiga muuta need informatsiooniks, kasutajale vajalikule väljundandmete kujule. Viimane võib olla nii heli, video, graafika, arvude kui ka teksti kujul. (Petuhhov, Andmettlusssteem 2011)

Andmebaas

Andmebaas on andmekogu, mis on salvestatud vastavalt konkreetsele andmestruktuurile, mille manipuleerimine toimub vastavalt andmete modelleerimise vahendite eeskirjadele.

Programmeerimiskeel Java

Java on populaarne platvormist sõltumatu programmeerimiskeel. Java sünnitajaks (1990) oli Sun Microsystems ning nimeks anti OAK. Nimi Java anti talle hoopis viis aastat hiljem (1995), kui keelele tehti erinevaid parandusi. Loodud keel võttis eeskuju C++ keelelt ja seepärast võite kasutamisel tabada end mõttelt, et need keeled on ju päris sarnased. (Metshein 2018)

Automaatne kompileerimise süsteem

Kompileerimise automatiseerimise süsteem (Build automation system) – projekti koostamise vahend, mis automatiseerib kompileerimist, värskendamist, testimist ja dokumenteerimist.(rohkem sellest teemast Clark 2004)

Hoidla

Hoidla on rakendustarkvara juurde kuuluva info andmebaas, mis sisaldab autori nime, andmeelemente, sisendeid, protsesse, väljundeid ja nendevahelisi suhteid.

JUnit

JUnit on teek tarkvara testimiseks Java-keeles.

Teek

Programmeerimises nimetatakse teegiks valmiskompileeritud alamprogramme, mida programm saab kasutada. Need alamprogrammid, mida kutsutakse ka mooduliteks, salvestatakse objektkoodidena. Teegid on eriti kasulikud sageli kasutatavate alamprogrammide salvestamiseks, sest siis pole neid vaja käsilolevasse programmi otse sisse kirjutada. (Vallaste, library(2) 2000)

XML

XML on laiendatav märgistuskeel, see on suvaliste andmete struktureerimiseks mõeldud märgistuskeel, mis loodi eesmärgiga võtta see veebis kasutusele HTML'i asemel. Nimelt osutus HTML oma fikseeritud elementide ja atribuutidega paljude ülesannete jaoks liialt piiratuks. (Vallaste, XML 2000)

SCRUM

Scrum on iteratiivne ja kasvava populaarsusega agiilse tarkvara arendamise raamistik. Scrum keskendub projektijuhtimisele, kus on raske pikalt ette planeerida. Scrum koosneb sprintidest, mis jäävad tavaliselt ühe nädala kuni ühe kuu vahele. Igal sprindi päeval toimub arendajatel lühikoosolek, kus arutatakse seniseid saavutusi ning eesmärke. (rohkem sellest teemast Scrum Alliance 2018)

Agiilsed arendusmeetodid

Agiilsed arendusmeetodid on tarkvara arendamise viis, mis on orienteeritud iteratiivse arengu kasutamisele, nõuete dünaamilisele kujunemisele ja nende rakendamisele. Agiilne arendusega seotud meetodid: ekstreemprogrammeerimine, DSDM, Scrum, FDD. (rohkem sellest teemast Agile Alliance 2018)

XP

Ekstreemprogrammeerimine ehk XP on tuntumaid agiilmeetoodeid. XP-s viiakse sammud läbi äärmuslikult (ekstreemselt - siit meetodi nimetus) lühikestena, võrreldes klassikaliste arendusmudelitega - esimene sammude läbimistsükkel võib olla päevad või nädal pikk, samas kui klassikalistes mudelites kestab see kuid ja aastaid. Enne kodeerimist kirjutatakse automatiseeritud testid, mida tarkvara peab läbima, seejärel programmeeritakse paarides (so kaks programmeerijat ühe arvuti taga kodeerivad ühte programmilõiku - nn "paarisprogrammeerimine"). Kui valminud kood läbib testid, on programmeerimise samm antud iteratsioonis lõpetatud. (Petuhhov, Inkrementaalne arendusmudel 2011)

Süsteemiarenduse elutsükkel

Süsteemiarenduse elutsükkel (Systems Development Life Cycle) (ka tarkvaraarenduse elutsükkel) on protsess, mille käigus luuakse uus või muudetakse vana tarkvarasüsteemi, samuti mudelid ja meetodid, mida inimesed kasutavad süsteemide arendamiseks. Tarkvara kui toode on süsteemiarenduse väljund. Süsteemiarenduse protsess koosneb nii toote projekteerimisest (disainist) kui toote valmistamisest. (Petuhhov, Elutsükli faasid 2011)

Inkrementaalne arendus

Inkrementaalne arendus on etapiviisiline ja ajagraafikut järgiv strateegia, kus süsteemi erinevaid osi arendatakse erinevatel aegadel ja erineva kiirusega ning kui üks osa valmis saab, integreeritakse see juba valmis süsteemiga. (Petuhhov, Inkrementaalne arendusmudel 2011)

Iteratiivne arendus

Iteratiivne arendus on nö muutmisstrateegia, kus nähakse ette olemasolevate süsteemi osade ümbertegemist ja parandamist. Alternatiivne strateegia oleks planeerida tegevused selliselt, et kõik tehtaks õigesti esimesel katsel. (Petuhhov, Inkrementaalne arendusmudel 2011)

Refaktoriseerimine

Refaktoriseerimine (refactoring) on koodi ümberkujundamine, koodi muundamine, algoritmide samaväärne värskendamine – programmi sisemise struktuuri muutmise protsess, mis ei mõjuta selle välist käitumist ja mille eesmärk on hõlbustada programmi töö mõistmist. (Fowler 1999: 9)

Git

Git on projekti moodulite kaughoidlasse integreerimise ja versioonihalduse tarkvara.

Puhaskood

Puhaskood on arenduse viis, et loodud kood oleks selge ja läbipaistev. (Martin 2008: 28-36)

Jupats

Jupats, tupikprogramm – alamprogramm, mis ei tee midagi muud, kui annab teada enda olemasolust ja parameetritest, mida ta aktsepteerib. Jupatseid kasutatakse harilikult kohahoidjatena (placeholder) teiste programmide jaoks, mis pole veel valmis. Jupats sisaldab just niipalju koodi, et teda saaks kompileerida ja linkida programmi teiste osadega. (Vallaste, stub(1) 2000)

Prototüüp

Prototüüp – programmi eelnõu, töötlemata või katseversioon. Arendatakse, et testida rakendustele pakutavate mõistete, arhitektuuri- ja/või tehnoloogiliste lahenduste sobivust ja esitada tarkvara kliendile arenguprotsessi varases staadiumis. (Petuhhov, Prototüüpimine 2011)

Musta kasti testimise meetod

Musta kasti meetodil vaadeldakse süsteemi kasutajale nähtavaid osi (kasutajaliidest), süvenemata koodi (siit nimetus „must kast"). Testjuhtumid koostab testija (peamiselt kasutuslugude põhjal), kes testid ka läbi viib.

Voog

C-keele puhverdatud sisend/väljund teegifunktsioonide juures on voog seotud faili või seadmega, mis oli avatud käsuga fopen. Tähemärke võib voost lugeda (voogu kirjutada) ilma nende tegelikku allikat (sihtpunkti) teadmata ja läbipaistev puhverdamine on tagatud teegirutiinidega. (Vallaste, stream(2) 2000)

Autor kasutas Java keelt, kuid see termin on saamasugune Java ja C keeltes.

Kehtestama

Kehtestama (commit) – muudatusi püsivateks salvestama (Vallaste, commit 2000)

Pärilus

Pärilus on klassi omaduste vaikimisi laiendamine alamklassile teadmuse hierarhilisel esitamisel. (Vallaste inheritance(1) 2000)

Turvaauk

Turvaauk on arvutiprogrammi või -süsteemi niisugune omadus, mida selle loomise käigus kas ei mõeldud korralikult läbi, ei osatud ette näha, tehti hooletult või otsustati ignoreerida, ning mille kaudu saab sedasama süsteemi kuritarvitada. (Arvutikaitse: 2018)

Klass

Java programmikeeles nimetatakse klassiks tüüpi, mis defineerib teatud liiki objekti teostuse (realisatsiooni). Klassidefinitsioon määrab ära eksemplari ning antud klassi muutujad ja meetodid, samuti klassi poolt realiseeritavad liidesed ja antud klassi vahetu ülemklassi. Kui ülemklass ei ole ilmutatud kujul kirjeldatud, siis loetakse see objektiks (Vallaste, class 2000)

Meetod ehk reegel

Objektorienteeritud keeltes nimetatakse reegliks protseduuri või rutiini, mis on seotud ühe või mitme klassiga. Antud klassi objekt teab, kuidas tegutseda, näit. ennast printida või luua endast uus eksemplar, mitte et funktsioon (näit. printimine) teaks, kuidas erinevat liiki objektiga ümber käia. Erinevad klassid võivad defineerida ühe ja sama nimega reegleid (s.t. reeglid võivad olla polümorfsed). Terminit "reegel" kasutatakse nii nimega operatsiooni, näit. "PRINT" kohta kui ka koodi kohta, mida antud klass pakub selle operatsiooni teostamiseks. (Vallaste, method(2) 2000)

Sprint

Sprint on iteratsiooni nimetus SCRUM'is. Meeskonna ülesannete valimise ja täitmise protsess töötamisel SCRUM'i tehnoloogia alusel. Iga sprindi lõpus peaks arendatav toode olema sellises vormis, millega klient saaks töötada. (rohkem sellest teemast Scrum Alliance 2018)

2 PROJEKTIS KASUTATUD TEHNOLOOGIAD

Projekti loomisel kasutati tehnoloogiaid, mis sobisid kõige paremini eesmärkide saavutamiseks ja olid projektiarendajatele huvitavad. Konkreetse tehnoloogia valimisel hinnati selle rakendamise mugavust tellija vajadusi arvestades. Allpool on loetletud valitud tehnoloogiad ja tingimused, mis mõjutasid tehtud valikuid.

2.1 Programmeerimiskeskkond ja keel

Programmeerimisekeel Java

Java on kaasaegne, platvormist sõltumatu, objektorienteeritud programmeerimiskeel, mis toetab tehnoloogiaid, mida arendajad kavatsesid kasutada.

Programmeerimiskeel pidi olema platvormist sõltumatu, kuna lõpptoote tingimuste seas oli nii MacOS'i kui ka Windowsi baasil tarkvara kasutamine.

Praktiliste kogemuste omandamine Java-keele rakendamisel autorile on esmatähtis ja oluline, kuna autor õpetab programmeerimise aluseid Java baasil.

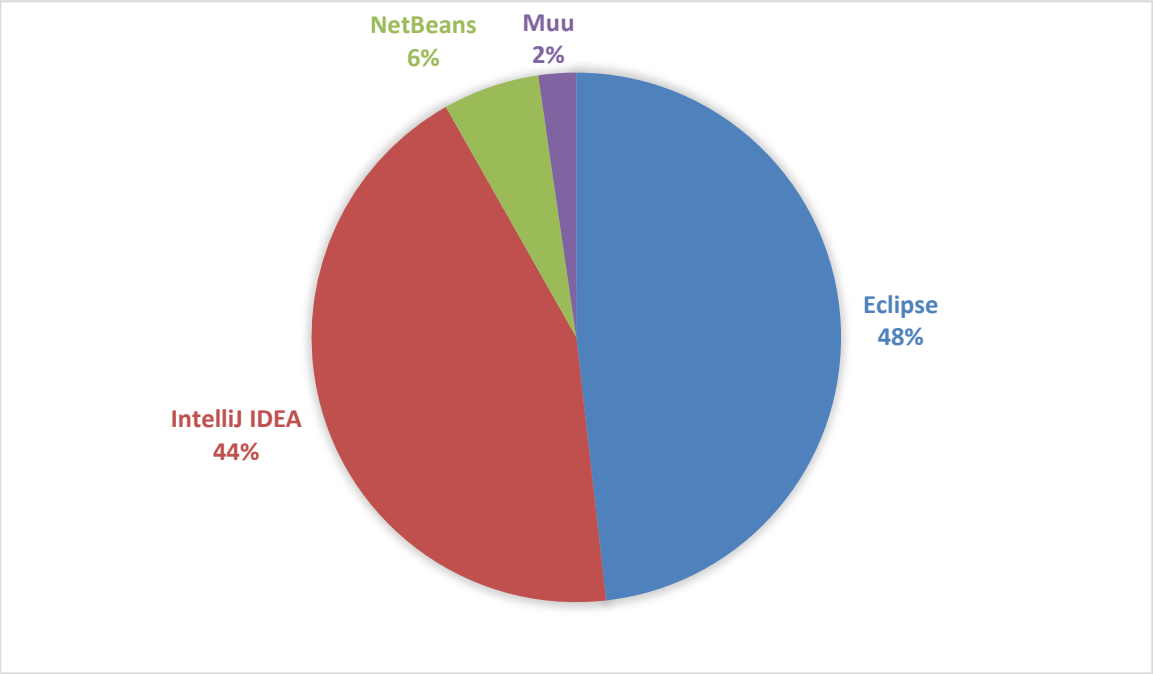
Java-programmeerimiskeele lõpliku valikut mõjutas tellija nõue, et oli vaja kasutada keelt, millega tellija on tuttav, kuna ta osales ka infosüsteemi loomises ja oli nii tellija kui ka arendaja.

Integreeritud programmeerimiskeskkond (IDE) Eclipse

Integreeritud programmeerimiskeskkonda valiti NetBeans, IntelliJ IDEA ja Eclipse vahel.

Töö autori hinnangul on hetkel kõige produktiivsem IDE IntelliJ IDEA, kuna see tunneb kindlalt sisestatud koodi konteksti. Kahjuks aga projektis seda kasutada ei saanud, kuna kommertsprojektide arendamiseks nõuab see iga-aastast makset. Kuna autor töötab korraga nii kommerts- kui haridusprojektidega, siis otstarbekas oli valida selline IDE, mida oleks võimalik kasutada tasuta mõlemas valdkonnas.

Mis puudutab IDE-d NetBeans, siis autoril on negatiivne kogemus selle kasutamisel, sest selle esimesed versioonid ei lubanud muuta automaatselt genereeritud koodi, see oli ebamugav ja viis ülemäära keeruliste koodi elementide loomisele graafilise kasutajaliidese loomisel.



Joonis 1 IDE-de kasutamise statistikat

Seega kokkuvõttes, valiti IDE Eclipse, kuna see on kõige populaarsem tasuta IDE Java arendamiseks ning autor kasutab seda oma igapäevatöös. Joonis 1 näitab erinevate IDE-de kasutamise statistikat (Paraschiv 2016).

Maven

Arendamise protsessi alguses ei teadnud projektiarendajad täpselt, milliseid teeke tuleks kasutada, seega otsustati kasutada kompileerimise automatiseerimissüsteemi, et teekide uuenduste puhul poleks probleeme põhiprojektiga. Valik oli Maven või Ant. Maven valiti, kuna see süsteem järgib teekide värskendusi ise. Teekide vastastikuste sõltuvuste olemasolu korral on Antiga töötamine keeruline ja nõuab vanade teekide kustutamist või uute lisamist käsitsi, mis võib põhjustada vigu.

2.2 Süsteemiarenduse elutsükli mudelid ja arendusmeetodid.

Selles peatükis kirjeldatakse arenenud tarkvaratoote *elutsükli* mudelit.

Inkrementaalne arendusmudel

Kasutati *inkrementaalset arendust,* sest meeskonna liikmetel polnud mingit võimalust oma tegevust ette planeerida, oli vaja kasutada niisugust mudelit, milles igaüks saaks oma koodiga tegeleda ja hiljem süsteemi lisada.

Iteratiivne arendus

Iteratiivset arendust kasutati seoses pideva vajadusega projekti muudatusi teha. Kuna koodi mahu suurendamine paratamatult viib uute vigade tekkimisele süsteemi, mida ei olnud üksikute

moodulite testimise ajal, siis nõuaks projekt pidevat *refaktoriseerimist*. Iteratiivne arendus võimaldab selle vajadust oluliselt vähendada, mis lihtsustab ja kiirendab tarkvara arendust.

Agiilsed arendusmeetodid

Selles osas kirjeldatakse agiilseid arenduse meetodeid, mida kasutati antud töös. Selgitatakse, milliseid valitud meetodite elemente kasutati.

SCRUM

Projekti arendajate meeskond kasutas osaliselt SCRUM tehnoloogiat siis, kui see oli otstarbekas.

SCRUM elemendid:

1. Projekti soovide nimekiri (Project backlog / user story list)

Kasutuslugusid ei lisatud eraldi nimikirja, kuid mõnikord arutati suuliselt.

2. *Sprindi* soovide nimekiri (Sprint backlog / task list)

Pärast iga kasutusloo arutelu koostati võimalike ülesannete nimekiri kasutusloo baasil.

3. Sprindi planeerimine

Sprindi ülesanded jagati meeskonna liikmete vahel.

4. Sprindi langustrendi graafik (Burndown chart)

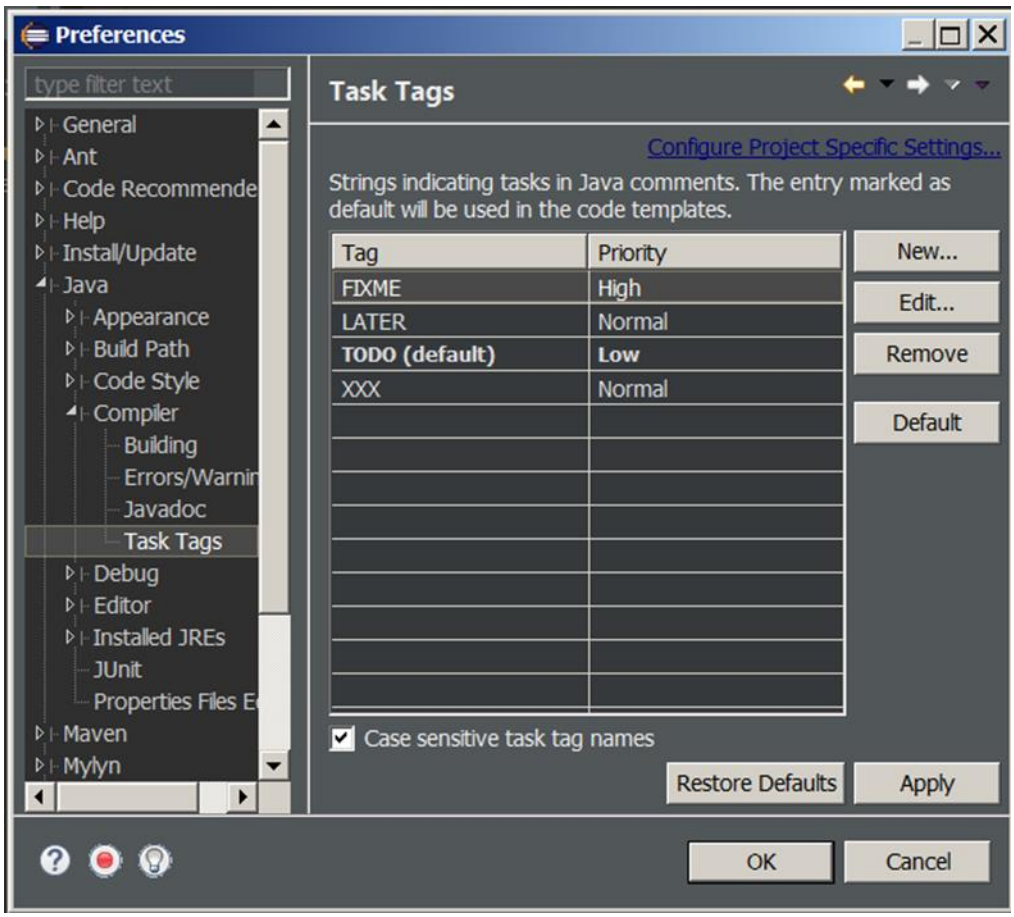
SCRUMi lauda ja sprindi langustrendi graafikut töös ei kasutatud, kuna tööl ei olnud jäika ajakava.

5. Igapäevased koosolekud

Meeskonnal ei olnud võimalust pidada igapäevasid koosolekuid. Projekti arendamine toimus vabal ajal, aga meeskonna osalejate töögraafikud ja vaba aeg olid erinevad. Igapäevaste koosolekute asemel kasutati *Git kehtestamist*, Eclipse'i tegumimärgendeid (vt **Ошибка! Источник ссылки не найден.**), *puhaskoodi* ja lühikesi selgitusi Skype'i kaudu.

6. Retrospektiivne koosolek

Retrospektiivsed koosolekud viidi läbi Skype'i videokonverentside teel.



Joonis 2. Eclipse'i tegumimärgendite seadistusmenüü

Ekstreemprogrammeerimine (XP)

XP elemendid:

1. Väljaannete kava (release plan).

Seda elementi töös ei kasutatud, kuna ei olnud vaja jälgida kindlat ajakava.

2. Iteratsioonide planeerimine.

Sama, mis eelmises punktis.

3. Igapäevased koosolekud.

Seda elementi töös ei kasutanud, kuna ei olnud võimalust korraldada igapäevaseid koosolekuid (vt SCRUM punkt (5)).

4. Metafooride süsteem.

Koodis on mitmeid reegleid, milles meeskonna liikmed arendamise käigus kokku leppisid. Näiteks, kui projektis leiti klassi, mille nimi sisaldas sõna "Tabel", siis see tähendas klassi analüüsitud lähtekoodist, kui aga sõna "Field", siis tähendas see analüüsitud klassi meetodit ja nii edasi.

5. Tellija töökohal.

Kuna töö tellija oli samaaegselt arendusmeeskonna liige, siis eemaldas see probleeme, mis on tavaliselt seotud valdkonna võimaliku teadmatusega.

6. Testimine enne arendamist.

Mõne projekti klassi jaoks enne arendustööde alustamist kirjutati JUnit teste, mis võimaldasid täpsemalt mõista, kuidas arendatav klass peaks töötama.

7. Paarisprogrammeerimine.

Kui töögraafik lubas, võis meeskond kasutada paarisprogrammeerimist. Selle jaoks kasutati Skype'i (v7.40.0.103) või TeamViewer'it (v12.0.75813) töölaua jagamiseks. Skype'is on märkimisväärseid puudusi – tarkvaraarendajal pole võimalust oma töölauda näidata, kui tal on ühendatud rohkem kui kolm kuvarit. Sellel juhul oli vaja riistvara välja lülitada ja operatsioonisüsteemi taaskäivitada. Lisaks Skype ei luba kaugtöölaua juhtimist, vaid ainult töölaua näitamist. TeamViewer-i puudus on konverentsi piiratud kestus – mõnikord on vaja konverentsi taaskäivitada tasuta versiooni piirangute tõttu.

8. Positsioonide muutmine.

Meeskonnal ei olnud selget plaani positsioonide muutmiste osas, aga kuna mõlemad liikmed olid huvitatud oma kutseoskuste taseme tõstmisest, siis aeg-ajalt tegevusala muudeti.

9. Kodeerimisleping (Coding Agreement).

Arengus kasutati puhaskoodi loomise põhimõtteid, mis lihtsustas ülalmainitud positsioonide muutmist.

10. Sagedane integratsioon.

Sagedane integratsioon saavutati tänu sellele, et iga töösessiooni lõpus, olenemata koodi valmidusastmest, lisati see põhiprojekti. Koodi lõpetamata osa oli varjatud *jupatsitega* või välja kommenteeritud. Kavandatud globaalsete projekti muudatuste puhul testiti funktsionaalsust kohalike hoidlate, eraldi harude või *prototüüpide* abil.

Prototüüpimine

Projekti arendamisel kasutati prototüüpimist mitmel korral:

- Kasutajaliidese muutmise demonstreerimiseks;
- Koodi analüüsi algoritmide muutmiseks, kui koormustestimine näitas ebapiisavat aega suure hulga andmete töötlemiseks;

- samuti muude väikeste ülesannete jaoks (näide: loodi väike programm, mis näitab, kuidas rippmenüü töötab, ja ainult pärast seda lisati vastavad muudatused põhiprojekti).

Peale testimist prototüübid kustutati, aga põhiprojektis tehti muudatusi prototüüpide testimise käigus saadud kogemuste põhjal ja lisati kasulik kood.

2.3 Unifitseerimise vahendid: XML

Arendatav tarkvara analüüsib suletud programmeerimiskeele lähtekoodi. Analüüsitud keele süntaktiliste konstruktsioonide peamisi sõnu ja kirjeldusi kasutatakse otsinguks ja analüüsimiseks. Võtmesõnad ja konstruktsioonid, nagu ka kogu analüüsitud keel, on ärisaladus, seetõttu need on salvestatud eraldi faili. Sõltuvalt olukorrast saab konfiguratsioonifaili asendada: testide ja demonstratsioonide jaoks kasutatakse ühte faili ning infosüsteemi tegelikuks toimimiseks teist.

Konfiguratsioonifaili loomine ja programmi struktuuri muutmine vastavalt sellele oli täiendav ülesanne infosüsteemi arendamisel.

Arengu alguses kasutati konfiguratsioonifaili, mis sisaldas teksti koos sõnade loendiga ja konstruktsioonide kirjeldusega. Kuid lihtsa tekstifaili kasutamine tingis vajaduse analüüsida seda kindlas järjekorras, samuti mõelda välja keeleelementide salvestamisele oma vormingut. Hiljem otsustas meeskond, et XML on sobivam ja professionaalsem lahendus selle ülesanne lahendamiseks, mis pakub suuremat paindlikkust konfiguratsioonifaili kasutamisel.

2.4 Testimine

Testimise tüübid:

1. Moodultestimine

Kuna XP metoodika tähendab ekstremaalselt kiiret arengutsükli kordamist, sageli arendatava mooduli all mõeldakse ühe klassi või isegi uue meetodi lisamist. Uue mooduli testimist viis läbi selle arendaja. Enne projekti integreerimist pidi tarkvaraarendaja veenduma, et moodul töötas korrektselt, selleks kasutati JUnit-i teste.

Testi näide:

```
package code_analyzer.source_analyze;

import static org.junit.Assert.assertTrue;

import java.io.BufferedReader;

import java.io.FileNotFoundException;
```

```
import java.io.StringReader;
import org.junit.Test;
public class SourceFileReaderTest {
	@Test
	public void getNextCharTest() throws FileNotFoundException {
SourceFileReader sourceFileReader = new SourceFileReader(new CodeSource(new BufferedReader(new StringReader("abcdef")), "teststream"));
		boolean res = false;
		int charCount = 5;
		int achar = -1;
		for (int i = 0; i < charCount; i++) {
			achar = sourceFileReader.getNextChar();
		}
		assertTrue("expected to be 'e', but is " + achar, achar == 'e');
		achar = sourceFileReader.getNextChar();
		assertTrue("expected to be 'f', but is " + achar, achar == 'f');
	}
}
```

See test kirjutatakse, lisades meetodi getNextChar() SourceFileReaderi klassi, seda meetodit kasutatakse lähtekoodi ridade loomiseks. Testrida "abcdef" testimise ajal loetakse viies ja järgmine sümbol. Test on edukalt läbitud, kui tähed "e" ja "f" on loetud.

2. Integratsioontestimine

Integratsioontestimine viidi läbi prototüüpides või hoidla eraldi harudes. Sellise testitüübi eesmärk on tuvastada vigu, mis tekivad, kui uus moodul ja teised, temaga seotud moodulid, vahetavad andmeid.

3. Süsteemtestimine

Süsteemtestimine viidi läbi kasutuslugude põhjal. Et *musta kasti meetodit* jäljendada, arendajad kasutasid ainult kasutajaliidest koodi vaatamata.

4. Regressioontestimine

Pärast projekti uue koodi lisamist viidi läbi JUnit-i testide kontroll. Kui testi tulemused olid negatiivsed, siis taastati projekt eelmise versiooni seisuga, aga uus moodul viimistleti ning viidi läbi korralikum integratsioontest.

5. Jõudlus- ja koormustestid

Jõudlustestimisel otsingusüsteem analüüsis suurt hulka tekstiandmeid. Analüüsitud testfail sisaldas umbes 30 000 rida. Mõõtes aega, mis kulus, et analüüsida testandmeid, arendajad otsustasid kas on mõistlik rakendada koodi refaktoriseerimist.

Näiteks: projekti esimeses versioonis loeti fail reahaaval, seejärel ühendati need read suurde andmekogusse, mida hiljem jagati koostisosadeks keele süntaksi põhjal. Koormustest leidis märkimisväärseid viivitusi faili analüüsimisel. Selle tulemusena meeskond otsustas parandada analüüsimehhanismi suurema tootlikkuse saavutamiseks. Valiti meetodit, mille puhul sümbolite lugemiseks failisisu käsitleti *voona* (sellel juhul faili ei loeta sisse täielikult, vaid osaliselt selle analüüsimise käigus). Testid näitasid jõudluse kasvu, töötlemisaeg muutus umbes kaks korda väiksemaks.

6. Kasutatavuse testid

Graafilist liidese kvaliteeti ja selgust kontrolliti kasutajate peal, kes ei olnud seotud projektiga. Nendele anti lihtsaid ülesandeid, näiteks:

- Laadige määratud failidega kaust analüüsisüsteemi.
- Leidke andmebaasis olev klass või meetod täpsustatud nimega.
- Leidke infot, milline fail sisaldab määratud nimega klassi kirjeldust.

Kui kasutajatel olid raskused ülesande täitmisel, siis tellija nõusolekul liides muudeti vastavalt kasutajate soovidele.

7. Vastuvõtutest

Tellija kasutas lähtekoodi sisaldavate failikogusid projekti aktsepteerimiseks. Test näitas, et kõik tarkvarafunktsioonid, mis olid kavandatud projekti alguses, said realiseeritud.

8. Koodi läbivaatus

Regulaarse koodikontrolli tulemus oli mõnede klasside ja objektide nimede muutus ning kasutamata koodielementide ja kommentaaride identifitseerimine ja nende edasine kustutamine.

2.5 Versioonihaldustarkvara: GitHub

Git-i rakendamine oli oluline meeskonna osalejatele, et oleks võimalik paralleelselt töötada, integreerida uued moodulid projekti, luua testide jaoks eraldi harud, jälgida erinevaid versioone ja

statistikat. Nende eesmärkide saavutamiseks kasutati ressurssi aadressil GitHub.com; mis võimaldab luua tasuta hoidlaid mitteäriliste projektide jaoks, lisada uusi kasutajaid koodi redigeerimise õigustega, igaüks saab vaadata projekti läbi ja taotleda luba projektis osalemiseks. Kui projekt muutub äriliseks, on võimalus tasuda iga-aastast litsentsitasu, ja siis saavad projekti vaadata ainult need kasutajad, kellele on antud juurdepääs projektile. Kuna käesolevas töös kirjeldatud tarkvara on ehitatud selliselt, et seal ei sisaldu ärisaladusi, siis projekti jaoks valiti tasuta hoidla.

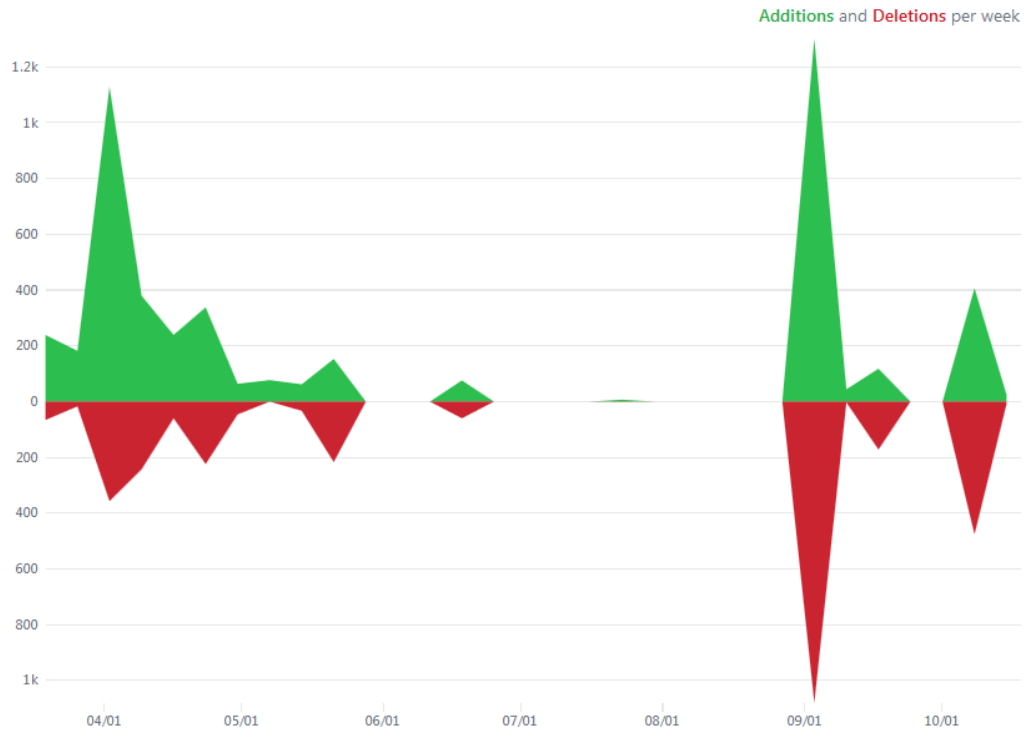
2.5.1 Projekti statistika

Ошибка! Источник ссылки не найден. näitab lisatud ja eemaldatud koodi suhet.(Code Analyzer, Code frequency 2016) Roheline värv tähistab lisatud koodi kogust, punane – kustutatud koodi. Septembris toimus peamise hoidla ja külgharu liitmine, seega on lisatud kood peaaegu võrdne kustutatud koodiga, sest GitHub eemaldab põhiprojekti ja lisab selle uuesti.

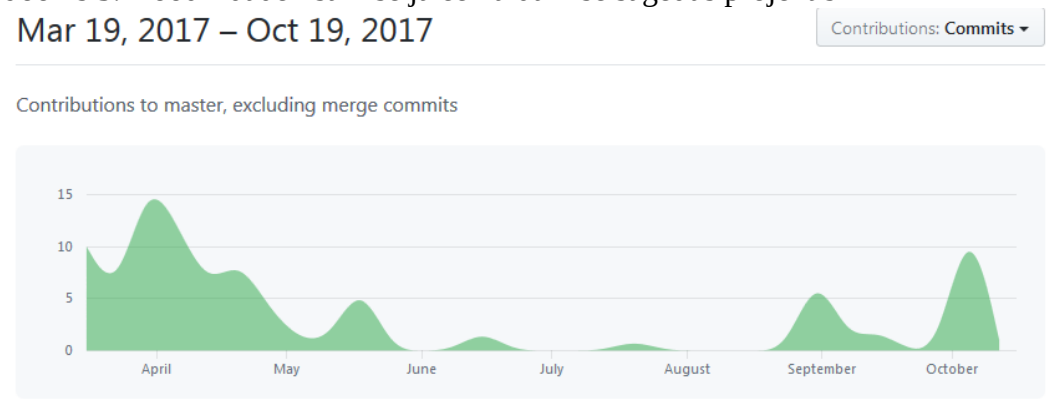
Ошибка! Источник ссылки не найден. näitab igakuist projekti täiendamiste arvu.(Code Analyzer, Contributors 2016) Nagu näha, projekti peaaegu ei arendatud suvel, kuna projektis osalejad olid puhkustel ja püüdsid arvutiga mitte aega veeta.

Ошибка! Источник ссылки не найден. näitab projekti meeskonna liikmete panust ning lisatud ja eemaldatud koodide ridade arvu.(Code Analyzer, Contributors 2016) Statistika ei ole päris õiglane, sest hoidla testharude ühendamisel GitHub lisab kõik liidetava haru koodi read ühendamist läbi viiva arendaja kustutatud koodi statistika alla.

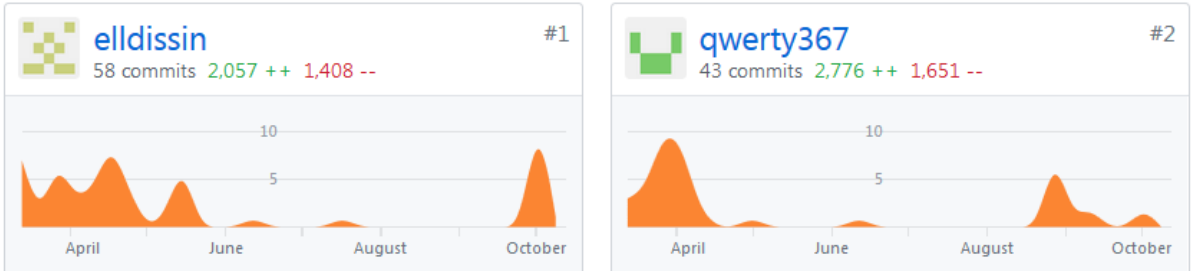
Graafik hüüdnimega elldissin kuvab selle infosüsteemi tellija statistikat, hüüdnimega qwerty367 – lõputöö autori statistikat.



Joonis 3. Koodiridade lisamise ja eemaldamise sagedus projektis



Joonis 4. Projekti täienduste tihedus kuu lõikes



Joonis 5. Projekti meeskonna liikmete panused projekti

2.5.2 Panuste näited

Projekti alguses mõlemad arendajad tegelesid tekstifailide analüüsimise klasside loomisega.

Näide ühest esimestest autori panustest:

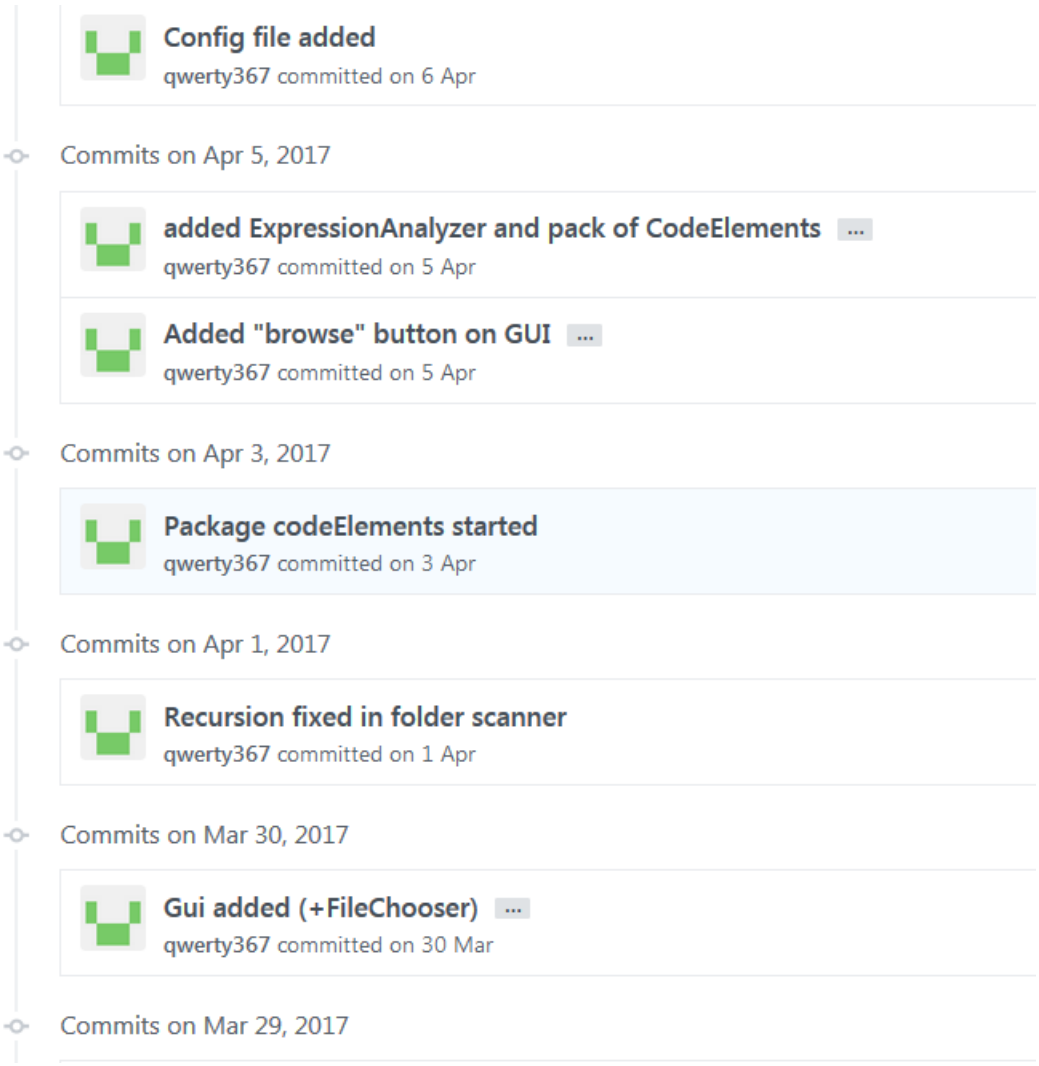
FileMaker added

```
+package code_analyzer;
+import java.io.BufferedWriter;
+import java.io.FileWriter;
+import java.io.IOException;
+public class SourceFileMaker {
+	SourceFileMaker() {
+		String fileSource = SourceFileReader.readFileContent("file.txt");
+		String newText = "\r\n" + "ururu"; // System.getProperty("line.separator")
+		try {
+			BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter("result.txt"));
+			bufferedWriter.write(fileSource + newText);
+			bufferedWriter.close();
+		} catch (IOException e) {
+			e.printStackTrace();
+		}
+	}
+}
```

See kood lihtsalt loeb faili file.txt teksti muutujasse fileSource, pärast seda luuakse fail result.txt, millele lisatakse muutujas fileSource olev tekst, järgmisele reale üleminek ja tekst “ururu”.

Kood on tehtud selleks, et näidata teisele projekti arendajale kuidas salvestada ja lugeda infot failidest, sest varem ta ei olnud sarnase ülesandega kokku puutunud.

Edasi autor keskendus graafilise kasutajaliidese arendamisele, kuid mõnikord osales ka failide analüüsimise meetodite arendamises.



Joonis 6. Panuste pealkirjade näited

Ошибка! Источник ссылки не найден. näitab mitut panuste pealkirja.(Code Analyzer, Commits 2017)

Kuna projekti arendajad kasutasid osaliselt XP tehnoloogiat, siis on raske täpselt määratleda, kes täpselt täitis mis ülesandeid – nad vahetasid mõnikord rolle, vaatasid üle ja muutsid üksteise koodi, samuti kasutasid ka paarisprogrammeerimist. Samas aga GitHub-i keskkond võimaldab tuvastada iga autori täpset panust kuni rea täpsuseni.

3 PROJEKTI TAUST, STRUKTUUR JA TULEMUSED

3.1 Projekti taust

Selles osas kirjeldatakse projekti ülesande püstituse tausta, tellija ootusi ja turu-uuringu tulemusi.

3.1.1 Tellijapoolne ülesande kirjeldus

Selle infosüsteemi tellija on tarkvaraarendaja, kes loob tarkvara ärisektorile, kasutades spetsiifilist suletud programmeerimiskeelt. Lähtekood on salvestatud failikogus. Uue tarkvara mooduli arendamisel ja uute klasside juurutamisel tarkvaraarendajale on vajalik lähtekoodi analüüsida juba loodud osade olemasolu kohta, kontrollida, et teine tarkvaraarendaja ei loonud enne klasse teiste projektide jaoks, mida saab kasutada uue mooduli arendamisel.

Arendamise eripära on selles, et tellija peab lähtekoodi mitmeid versioone haldama samaaegselt, ning selle koodi struktuur võib mitmes versioonis oluliselt erineda ja kõiki neid asjaolusid tuleb arvesse võtta.

Kuna tellija poolt kasutatud programmeerimiskeel on väga kitsa suunitlusega, on töövahendite komplekt selle kasutamiseks üpriski piiratud, sest siin kehtib lihtne reegel – mida väiksem on keele kasutajaskond ja kasutusala, seda vähem on selle jaoks loodud valmis tööriistu ja teeke. Seetõttu kulutavad tellija tarkvaraarendajad palju aega failide käsitsi avamiseks ja otsimiseks nende koodis sobivaid klasse ja meetodeid. Selle kitsa koha katmiseks tekkiski tellijal idee luua otsingusüsteem klasside ja meetodite leidmiseks.

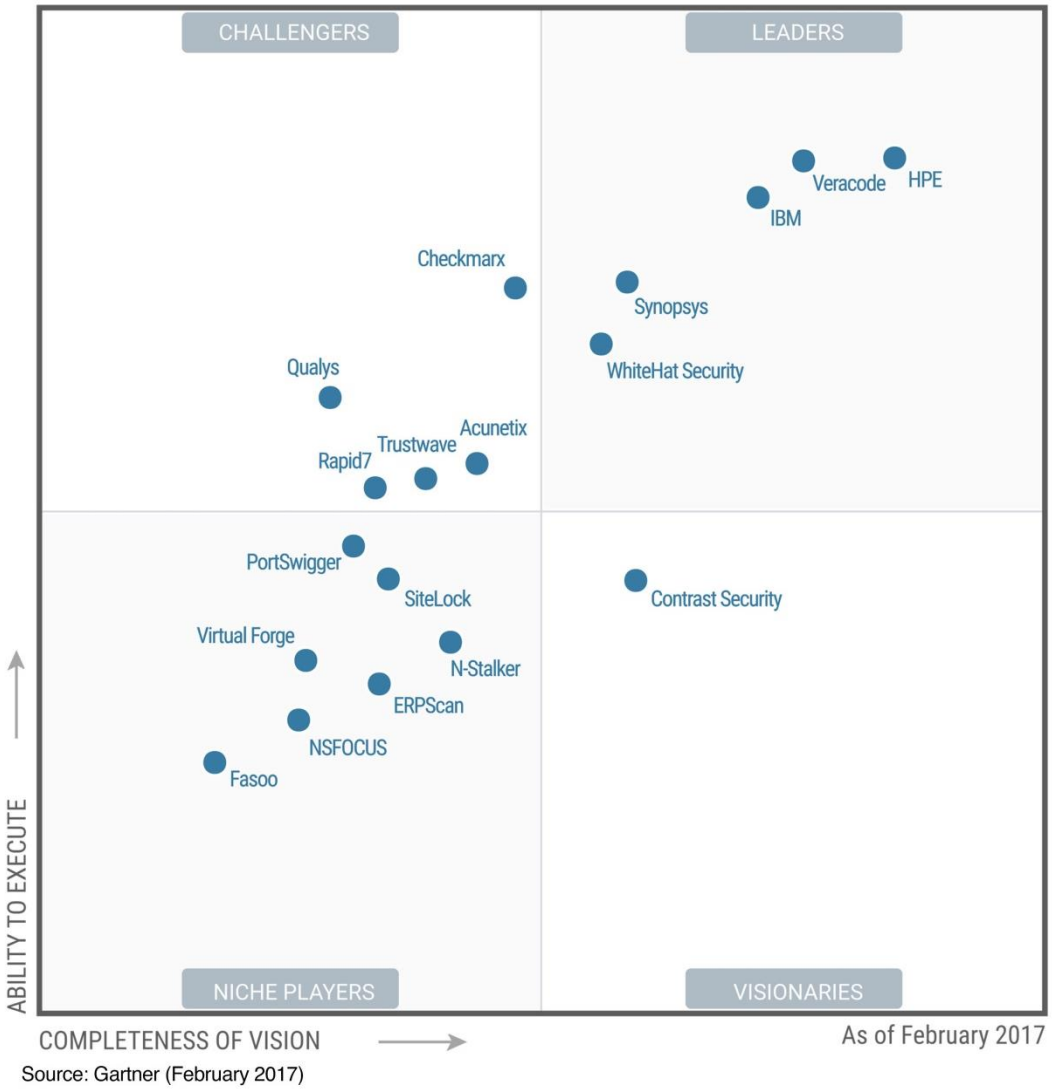
3.1.2 Tellija nõudmised otsingusüsteemi kohta

Tellija näeb oma tegevusi tellitavas otsingusüsteemis järgmiselt:

1. Programmi käivitamine (Windows või MacOS operatsioonisüsteemis).
2. Vajaliku failikoguga kausta valimine failide analüüsimiseks (tellija firmas on korraga kasutusel mitmed failikogude versioonid erinevate projektide tarbeks, sealhulgas ajutiste versioonidega ja juurdepääsu piirangutega).
3. Programm peab andma struktureeritud teavet analüüsitavates failides kasutatud klasside ja meetodite kohta.
4. Peab olema valik klasside või meetodite nimede otsimisel.
5. Koodi elemendi valimisel on vaja anda teavet selle omadustest ja failist, kus asub selle koodi elemendi kirjeldus.

3.1.3 Valmislahenduste turuanalüüs

Tellija ülesande lahendamiseks tehti väike turu-uuring selleks, et teada saada, kas saab kasuta mingeid valmislahendusi. Turul on olemas palju erinevat koodi analüüsimise tarkvara. Näiteks: HP Fortify Static Code Analyzer, Checkmarx CxSAST, IBM Security AppScan Source, «Эшелон» AppChecker, PT Application Inspector, InfoWatch Appercut, Digital Security ERPScan, Solar inCode ja palju muid. **Ошибка! Источник ссылки не найден.** näitab ettevõtteid, mis on ülemaailmse turu liidrid koodi analüsaatorite arendamisel.(Zumerle 2017)



Joonis 7. Ülemaailmse koodi analüüsimise tarkvara tootjate jaotus turul

Koodi analüsaatorite põhiülesanne on leida koodi *turvaaugud* ehk koodi struktuuri vigu. Mõned kirjeldatud rakendused võivad automaatselt genereerida tarkvaradokumentatsiooni, st täita nõutavat tellija ülesannet. Kuid kõiki neid suurepäraseid rakendusi, kahjuks, ei saa kasutada, kuna need ei toeta tellija ettevõtte poolt kasutatavat programmeerimiskeelt. Tellija ettevõtte arendab oma programmeerimiskeelt ning firma ise loob kõik arendusvahendid, mis suudavad selle keelega

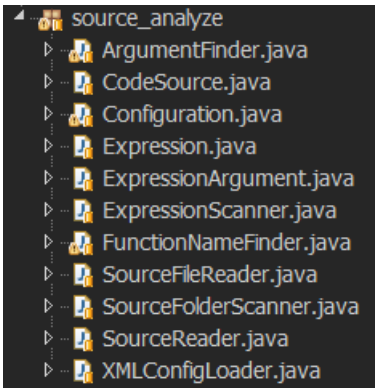
töötada, ja see oli peamine põhjus, miks oli vaja luua veel ühte lähtekoodi analüüsimise programmi.

3.2 Tarkavaraprojekti koodi struktuur

Selles osas kirjeldatakse projekti koodi struktuuri seisuga 20.10.2017. Kuna projekt pidevalt areneb, siis oli vaja valida mingi punkt, milleni projekti kirjeldatakse diplomitöös. Selleks punktiks oli valitud seis, millel projekt rahuldas kõiki tellija esialgseid nõudmisi. Hiljem, projekti arendamise käigus, tekkisid tellijal uued ideed ja soovid, aga need jäävad sellest tööst välja. Loogiliselt on projekt jagatud kolmeks paketiks: gui, db_elements ja source_analyze. Allpool on loetletud pakettides sisalduvad klassid ja nende lühikirjeldus.

3.2.1 Pakett source_analyze

Selles alamosas on kirjeldatud lähtekoodi failikogu analüüsiga seotud klassid (vt Joonis 8 ja Tabel 1).



Joonis 8. Koodi analüüsiga seotud klassid

Nende klasside funktsioonid on järgmised:

- Failide nimekirja loomine valitud kausta ja selle alamkaustade lugemiseks.
- Lähtekoodi sümbolite voo lugemine failist;
- Mitmest failist saadud sümbolite voogude konstrueerimine ühe tervikuna.
- Analüüsitava koodistruktuuri tüüpide määramine;
- Andmebaasi elementide loomine ja andmete lisamine andmebaasi;

Paketti source_analyze klassides kasutatakse teeke .util, .io ja .xml.

Tabel 1 Paketti source_analyze klassid ja eesmärgid

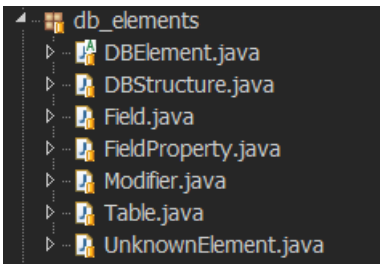
Klass	**Ülesanne**

ArgumentFinder	Koodi tuvastamise mehhanismi element, mis otsib, kas avaldises sisalduval meetodil on olemas parameetrid. Kasutatakse ExpressionArgument klassi objektide loomiseks.
CodeSource	Luua abstraktne esitus mitmest failist saadud lähtekoodi voost.
Configuration	Abiklass, mis loeb XML-faili ja lisab selles leitud elemendid oma staatilistesse väljadesse.
Expression	Loob andmebaasi elemente.
ExpressionArgument	Analüüsitud avaldise argument, mis sisaldab täiendavat teavet.
ExpressionScanner	Klass, mis skaneerib lähtekoodi ja saab keele avaldisi üksteise järel Expression klassi objektide kujul.
FunctionNameFinder	Koodi tuvastamise mehhanismi element, mis määrab avaldises sisalduva meetodi nime.
SourceFileReader	Üksiku faili sisu lugemine sümbolite voona.
SourceFolderScanner	Failide nimekirja loomine valitud kaustas ja selle alamkaustades.
SourceReader	Mitmest lähtekoodi voost saadud terviku lugeja, mis võimaldab vaadata mitme faili lähtekoodi kui ühte.
XMLConfigLoader	XML-faili laadija

3.2.2 Pakett db_elements

Paketis db_elements asuvad klassid, mis on seotud andmebaasi elementide loomisega, vt **Ошибка! Источник ссылки не найден.** ja Tabel 2.

Infosüsteem analüüsib lähtekoodi ja loob andmebaasi. Andmebaas sisaldab infot lähtekoodi iga leitud elemendi kohta – tüüp, nimi, failinimi, reanumber, – kus antud tuvastatud koodi element on leitud ja mis teistele elementidele see kuulub. Andmebaas koosneb Java-objektide elementidest.



Joonis 9 Andmebaasi elementide loomisega seotud klassid

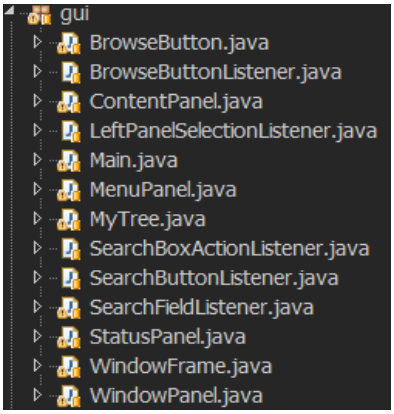
Tabel 2 Paketti db_elements klassid ja eesmärgid

Klass	**Ülemklass**	**Ülesanne**
DBElement		Andmebaas koosneb selle klassi objektidest, kõik lähtekoodi analüüsitud elementide tüübid on andmebaasis erinevad DBElement klassi alamklassid.
DBStructure		Põhiklass, mis kujutab endast failikogus leitud koodi struktuuri abstraktse mudeli. Selle klassi meetodid lisavad andmebaasi uusi elemente oma nimekirjadena (ArrayList).
Field	DBElement	Selle klassi objekte kirjutatakse andmebaasi, kui failide hulgast on leitud analüüsitava keele meetodi kirjeldus.
FieldProperty	DBElement	Selle klassi objekte praegu projektis ei kasutata. Edaspidi hakatakse selle klassi objekte kasutama lähtekoodis leiduvate meetodite atribuutide hoidmiseks.
Modifier	DBElement	Selle klassi objekte praegu ei kasutata, aga eeldatakse, et seda võib tulevikus vaja minna.
Table	DBElement	Selle klassi objekte kirjutatakse andmebaasi, kui failide hulgast on leitud analüüsitava keele klassi kirjeldus.

UnknownElement	DBElement	Selle klassi objekte tagastatakse, kui analüüsimise algoritm ei tuvasta koodielementi (kasutatakse jupatsina, et väärtust Null mitte tagastada).

3.2.3 Pakett gui

Selles paketis asuvad graafilise kasutajaliidese loomisega seotud klassid (vt Joonis 10 ja Tabel 3).



Joonis 10 graafilise kasutajaliidese loomisega seotud klassid

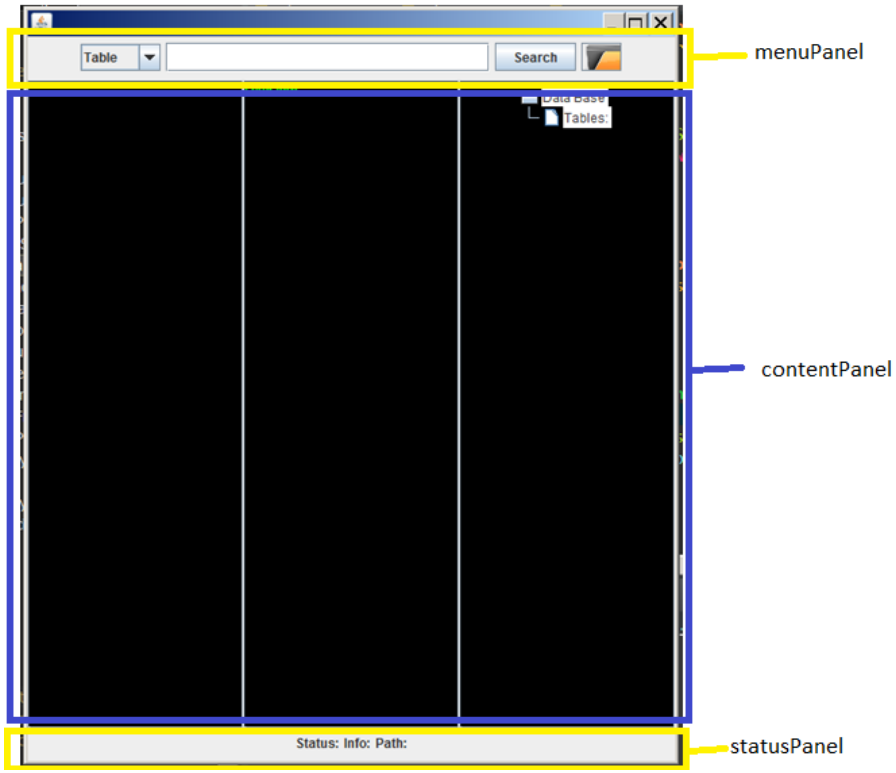
Enamik paketi gui klassidest on liideste ja klasside *pärijad* teekidest swing, util ja awt vt Tabel 3.

Tabel 3 Paketti gui klassid ja eesmärgid

Klass	**Ülemklass / Liides**	**Ülesanne**
BrowseButton	Klass JButton	Loob nupu lähtekoodi kausta valimiseks.
BrowseButtonListener	Liides ActionListener	Võimaldab valida lähtekoodi kausta, käivitab koodi analüüsimise algoritmi, loob andmebaasi leitud objektidega.
ContentPanel	Klass JPanel	Loob paneeli, vt Joonis 11
LeftPanelSelectionListener	Liides ListSelectionListener	Võimaldab valida vasakpoolsest paneelist elemendi, kuvab valitud elemendi kohta teabe keskpaneelile ja StatusPanelile, vt

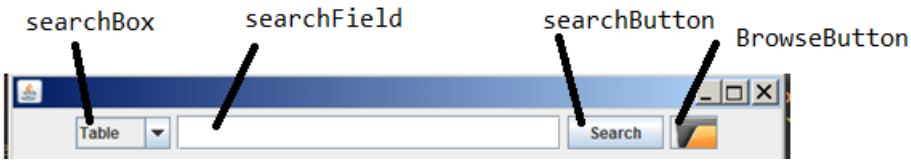
		Joonis 13.
Main		Rakenduse põhiklass.
MenuPanel	Klass JPanel	Loob paneeli vt Joonis 11.
MyTree	Klass JPanel	Loob hierarhilist puud parempaneelil, vt Joonis 13.
SearchBoxActionListener	Liides ActionListener	Võimaldab valida rippmenüüst otsingu variandi kas klasside või meetodite järgi, vt Joonis 12
SearchButtonListener	Liides ActionListener	„Search“ nupule klõpsatamisel otsib andmebaasis elementide nimesid, mis kattuvad SearchField sisestatud tekstiga.
SearchFieldListener	Liides KeyListener	Sama nagu SearchButtonListener, aga töötab “Enter” vajutamisel.
StatusPanel	Klass JPanel	Loob paneeli vt Joonis 11.
WindowFrame	Klass JFrame	Loob programmi akna.
WindowPanel	Klass JPanel	Loob paneeli, millele on lisatud MenuPanel, ContentPanel ja StatusPanel, vt Joonis 11.

Programmiaken on jagatud kolmeks põhipaneeliks (vt. Joonis 11).



Joonis 11 Programmiaken põhipaneelid

menuPanel sisaldab 4 elementi: rippmenüü otsinguks klasside/meetodite järgi, teksti otsinguvälja, nuppu "Search" ja nuppu "Browse", nagu Joonis 12 näidatud. Otsingut saab teha nii nupuga "Search" kui ka klaviatuuri nupuga “Enter”.



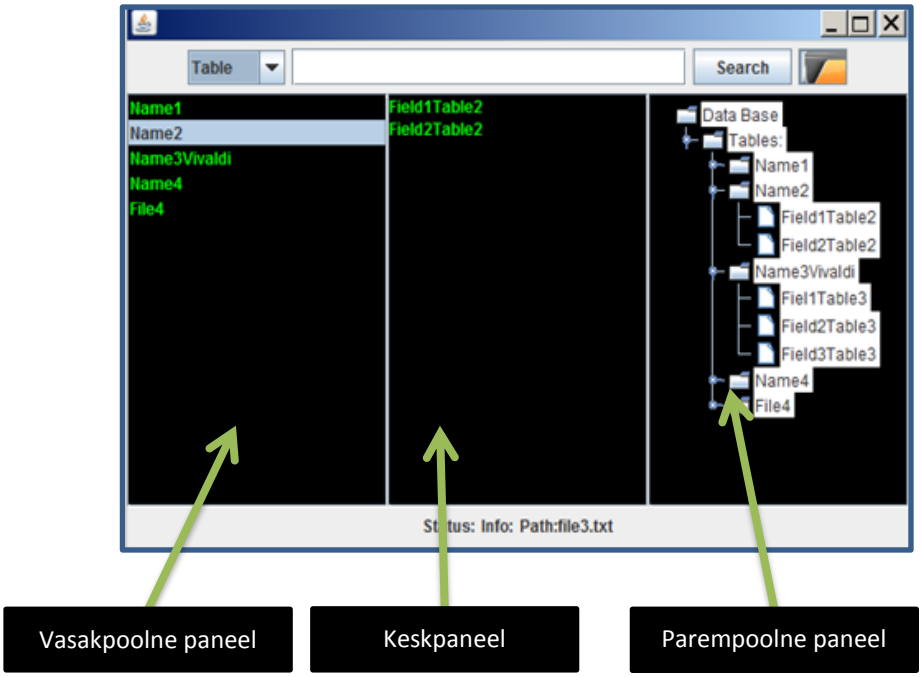
Joonis 12 menuPanel elementid

contentPanel on jagatud kolmeks osaks nagu Joonis 13 näidatud.

Vasakpoolset paneeli kasutatakse kas leitud klasside täieliku loendi või leitud meetodite täieliku loendi kuvamiseks, olenevalt menüüriba rippmenüü valikust.

Keskpaneeli kasutatakse selleks, et kuvada meetodeid, mis kuuluvad klassile, mida kasutaja valis vasakpoolsest paneelist. Kui aga vasakpoolsel paneelil on meetodite loend, siis meetodi valimisel näidatakse keskpaneelil klassi, milles asub valitud meetodi kirjeldus.

Parempoolses paneelis kuvatakse hierarhilist puud, mis koosneb failikogus leitud klassidest ja meetoditest.



Joonis 13 contentPanel jagamine

StatusPanel sisaldab teavet failinimest ja reast, kus leiti kasutaja poolt valitud koodi element.

3.2.4 Rakenduse UML-diagramm

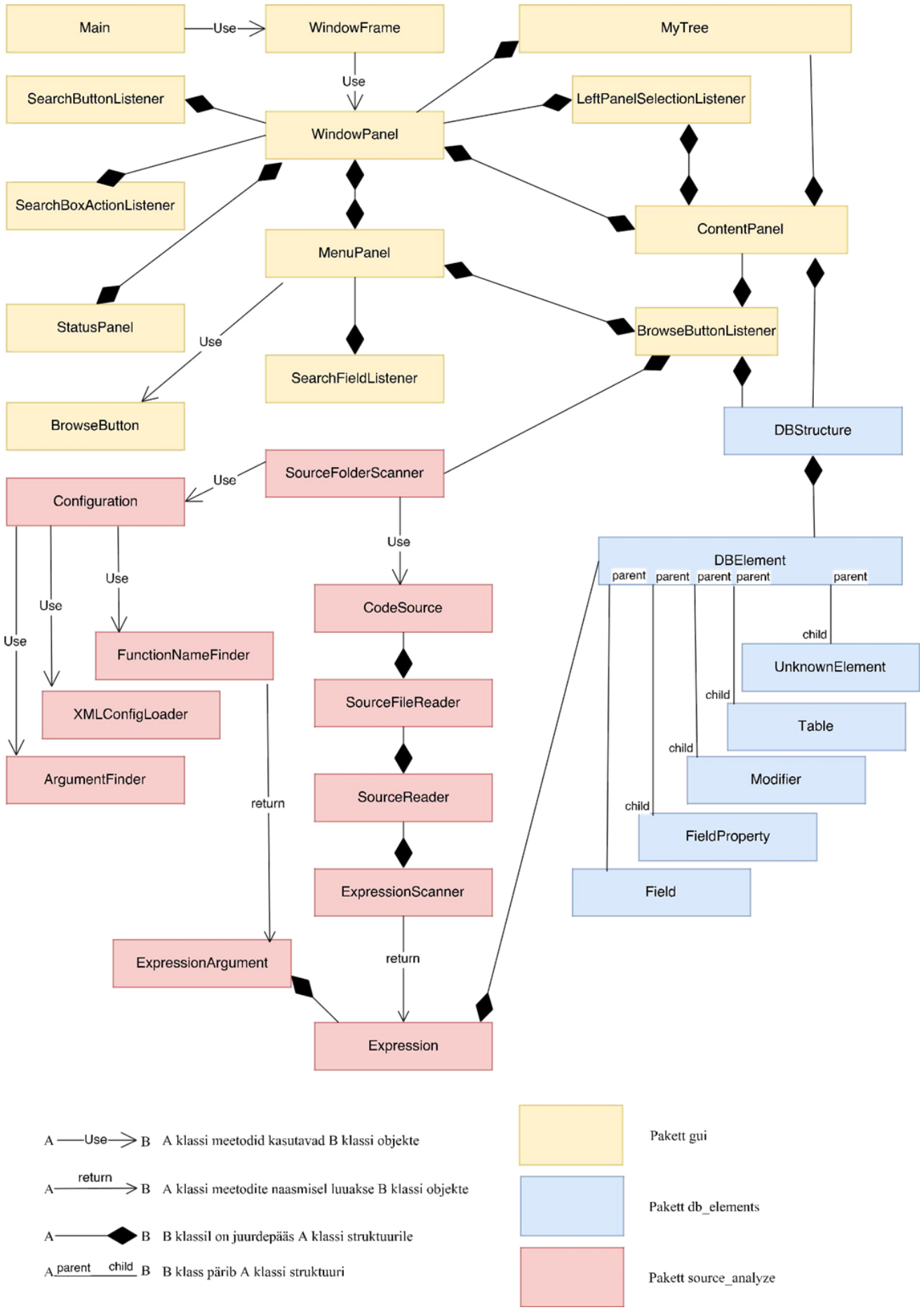
Ошибка! Источник ссылки не найден. ja tabelid 1-3 näitavad projekti loodud klasside seoseid, mille konstrueerimisel autor kasutas .jar failide analüsaatorit Code2UML ja disainivahendit www.draw.io. Lisateavet iga klassi kohta leiab lisadest (vt Lisa1).

Diagramm näitab suhteid erinevate pakettide vahel:

Kui kasutaja valib lähtekoodiga kausta nupu "Browse" abil, käivitatakse analüüsi algoritmid. Diagrammil seda peegeldab side BrowseButtonListener ja SourceFolderScanner vahel.

Analüüsimeetodid loovad Expression klassi objekte, mida teisendatakse andmebaasi elementideks, vt side Expression ja DBElement vahel.

Graafilise liidese elemendil on juurdepääs andmebaasi struktuurile selle kuvamiseks, vt side ContentPanel ja DBStructure vahel.



Joonis 14 Rakenduse üldine UML-diagramm

3.3 Tulemused

Alates septembrist 2017 tellija kasutab infosüsteemi, mis oli arendatud lõputöö käigus. Seni on tagasiside olnud positiivne. Koodi failikogudes otsimise aeg on märgatavalt vähenenud.

Infosüsteem aitab ülesande analüüsi etapil kontrollida, kas failikogus on olemas ülesande täitmiseks vajalikud klassid.

Arengufaasis on võimalus kasutada seda tarkvara, et uurida millised failid kirjeldavad üht või teist arendatava programmi osa, et tutvuda sellega üksikasjalikumalt, mis on kasulik koodi erinevate osade arendamisel erinevate arendajate poolt.

Katsetamisetapil on infosüsteem kasulik testijale, kuna see võimaldab lähtekoodifaile avamata otsida, millised meetodid kuuluvad klassidesse ja ette valmistada testi plaani. Kui sel viisil kompileeritud test ei tööta, siis tarkvaraarendaja ei andnud korralikult nimesid meetoditele ja/või klassidele.

Tarkvaratoe etapil on mugav seda tarkvara kasutada siis, kui arendaja saab kliendilt infot ilmuva vea kohta ja peab leidma, millises failis asub seda probleemi tekitava meetodi kirjeldus.

Tulevikus hakkab tarkvara arendajate meeskond lisama projektile võimalust klasside otsimiseks graafilise liidese nime järgi (st võimalust valida liidese akna nime ja saada kõike klasse ja meetodeid, mis on seotud selle liidesega).

Magistritööna plaanib autor valida universaalse arenduskeskkonna loomist, mille aluseks saab selle projekti source_analyze paketti kasutada.

KOKKUVÕTE

Antud lõputöö tulemus on valmis rakendus, mis vastab kõigile ainevaldkonna nõuetele.

Enne töö alustamist uuriti tellija töökorraldust ja avastati automatiseerimise puudumist. Saadud teabe alusel loodi otsingusüsteem, mis pakub vahendeid lähtekoodi failikogus klasside ja meetodite otsimiseks ja analüüsimiseks.

Kasutajaliidese loomisel rakendati kasutaja mugavuse huvides erinevaid Java-komponente.

Rakenduse loomise käigus lahendati mitmeid probleeme:

- loodi mitu andmestruktuuri esituse varianti;
- kood loodi sellisel viisil, et mõlemad meeskonna liikmed saaksid sellest aru ühtemoodi;
- koodi analüüsi algoritmide optimeerimine vähendas analüüsi aega;
- saab näha mitte ainult faili, milles analüüsitav koodielement asub, vaid ka konkreetset tekstirida, milles see on leitud.
- graafilise liidese elementide optimeerimine võimaldas teha palju kiiremaks paneelide *kerimist*.

Otsingusüsteemi loomine ja arendamine on täielikult õnnestunud. Selle töö otstarbekuse kõige olulisem kriteerium on see, et seda tarkvara kasutatakse juba mitu kuud tellija firmas ja see aitab tarkvaraarendajatel oma igapäevaseid ülesandeid paremini täita.

SUMMARY

DEVELOPMENT OF INFO SYSTEM FOR SOURCE CODE DATABASE ANALYSING, STRUCTURING AND SEARCHING

Present work is done by the order of a software development company (hereafter the custumer). The order was to develop an info system that partially woult automate the custumer developers work, which is connected with the source code search and analysis – the use of the info system allows the development team to search for the source code classes and methods automatically. The creation of the info system is topical because the custumer's activity implies a large scale labor processing of the source code.

The objective of this thesis was to develop an application which helps the custumer to optimize the developers working time by automation of the software source code searching and analyzing processes.

The written part of the thesis consists of three chapters. In the first chapter are defined the terms used in the work. In the second chapter the author lists chosen development technologies and conditions that influensed the choise. The third chapter describes the background of raised project tasks, the structure of the software project and the results.

The result of the given thesis is a ready application, which meets all the requirements of the subject area.

Before the start of work the custumer's work organization was investigated and the absence of automation was found. On the basis of received information was developed a search engine (searching system) that offers tools/means for searching and analyzing the classes and methods inside the source code database.

The search engine creation and development was a complete success. The most important criterion of the expediency of this work is the fact that this software is being used by the custumer for many months and it helps the developers to better perform at their daily tasks.

KIRJANDUS

Paraschiv, Eugen 2016. The Market Share of Java IDEs in Q2 2016. Baeldung
http://www.baeldung.com/java-ides-2016
(viimati vaadatud 07.04.2018).

Code Analyzer 2016. Code frequency. GitHub.
https://github.com/elldissin/code_analyzer/graphs/code-frequency
(viimati vaadatud 08.04.2018).

Code Analyzer 2016. Contributors. GitHub.
https://github.com/elldissin/code_analyzer/graphs/contributors
(viimati vaadatud 08.04.2018).

Code Analyzer 2017. Commits on Apr 11, 2017. GitHub.
https://github.com/elldissin/code_analyzer/commits/master?before=26fa558322543ac6d3fd35ad3dbc6c60aa447209+105 (viimati vaadatud 08.04.2018).

Metshein koolitusportaal 2018. Java – Sissejuhatus ja paigaldamine.
https://www.metshein.com/unit/java-sissejuhatus-ja-paigaldamine/
(viimati vaadatud 12.04.2018).

Zumerle, Dionisio 2017. Magic Quadrant for Application Security Testing. Gartner.
https://www.gartner.com/doc/3623017/magic-quadrant-application-security-testing
(viimati vaadatud 08.04.2018 nõuab registreerimist).

Petuhhov, Inga 2011. Infosüsteemi hankimine, arendus ja teostamine. Andmetöötlussüsteem.
http://www.e-uni.ee/e-kursused/eucip/arendus/111_andmettlusssteem.html
(viimati vaadatud 08.04.2018).

Petuhhov, Inga 2011. Infosüsteemi hankimine, arendus ja teostamine. Inkrementaalne arendusmudel.
http://www.e-uni.ee/e-kursused/eucip/arendus/1222_inkrementaalne_arendusmudel.html
(viimati vaadatud 08.04.2018).

Petuhhov, Inga 2011. Infosüsteemi hankimine, arendus ja teostamine. Prototüüpimine.
http://www.e-uni.ee/e-kursused/eucip/arendus/1224_prototpimine.html
(viimati vaadatud 08.04.2018).

Petuhhov, Inga 2011. Infosüsteemi hankimine, arendus ja teostamine. Elutsükli faasid.
http://www.e-uni.ee/e-kursused/eucip/arendus/121_elutskli_faasid.html
(viimati vaadatud 08.04.2018).

Clark Mike. Pragmatic Project Automation. Dallas, 2004.

Scrum Alliance 2018. Learn About Scrum.
https://www.scrumalliance.org/learn-about-scrum (viimati vaadatud 08.04.2018).
Agile Alliance 2018. Key Agile Concepts.
https://www.agilealliance.org/agile101/ (viimati vaadatud 08.04.2018).
Fowler Martin. Refactoring. Improving the Design of Existing Code. Vestford, 1999. lh 9.
Robert Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Bergen, 2008, lh 28-36.
Vallaste, Heikki 2000. library(2). e-teatmik
http://vallaste.ee/sona.asp?Type=UserId&otsing=2137 (viimati vaadatud 08.04.2018).
Vallaste, Heikki 2000. stub(1). e-teatmik
http://vallaste.ee/sona.asp?Type=UserId&otsing=759 (viimati vaadatud 08.04.2018).
Vallaste, Heikki 2000. stream(2) . e-teatmik
http://vallaste.ee/sona.asp?Type=UserId&otsing=1970 (viimati vaadatud 08.04.2018).
Vallaste, Heikki 2000. commit. e-teatmik
http://vallaste.ee/sona.asp?Type=UserId&otsing=2366 (viimati vaadatud 08.04.2018).
Vallaste, Heikki 2000. inheritance(1). e-teatmik
http://vallaste.ee/sona.asp?Type=UserId&otsing=1169 (viimati vaadatud 08.04.2018).
Vallaste, Heikki 2000. XML. e-teatmik
http://vallaste.ee/sona.asp?Type=UserId&otsing=342 (viimati vaadatud 08.04.2018).
Vallaste, Heikki 2000. class. e-teatmik
http://vallaste.ee/sona.asp?Type=UserId&otsing=1961 (viimati vaadatud 08.04.2018).
Vallaste, Heikki 2000. method (2) . e-teatmik
http://vallaste.ee/sona.asp?Type=UserId&otsing=1946 (viimati vaadatud 08.04.2018).
Arvutikaitse 2018. Turvaaugud. Infoturvalisuse teeviit.
http://www.arvutikaitse.ee/arvutikaitse-algtoed/turvaaugud/ (viimati vaadatud 08.04.2018).

LITSENTS

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks.

1. Mina Denis Volkov, annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Lähtekoodi failikogu analüüsimise, struktureerimise ja otsingu infosüsteemi loomine", mille juhendaja on Andre Sääsk.

1.1. Reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. Üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Narvas, 10.12.2018